Poznan University of Technology
Faculty of Mechanics and Transport
Chair of Virtual Engineering

Technical Report 02/2017

The Fluidic Pinball — A Toolkit for Exploring Multiple Input Multiple Ouput Flow Control (Version 1.0)

Bernd R. Noack & **Marek Morzyński**

May 2017

ul. Piotrowo 3, PL 60-965 Poznań, Poland

Abstract

The fluidic pinball is software package for multiple-input multiple-output flow control. This package allows the user to explore control design for a complex dynamics needing only minutes of computational Laptop time. The goal is to stabilize the two-dimensional wake behind three cylinders. The wake is changed by cylinder rotation, i.e. 3 independent inputs. The incompressible flow is computed with an in-house Navier-Stokes solver UNS3 employing a finite element method and implicit integration [1]. This solver has been employed for numerous wake investigations [2, 3, 4] and control studies [5, 6, 7]. The control command, i.e. the rotation speeds of the cylinders, is computed in a matlab/octave script which runs in parallel. Wake stabilizing actuation mechanisms include low-frequency forcing [8], high-frequency forcing [9], base-bleeding, boat tailing, stagnation point and circulation control. The challenge for control design is that many successful actuation mechanisms are strongly nonlinear and based on frequency crosstalk [10, 11]. This 2nd edition of the 2016 report [12] provides more details about the code.

Keywords: Flow control, wake stabilization, direct numerical simulation

Contents

1 Introduction **1**
- 1.1 Wake stabilization as flow control benchmark . . . 1
- 1.2 Categories of wake stabilization strategies . . . 1

I Capabilities **5**

2 Control problem **6**
- 2.1 Flow configuration . . . 6
- 2.2 Dynamics . . . 6
- 2.3 Actuation, sensing and cost function . . . 7
- 2.4 Direct numerical solution . . . 8

3 Control example **9**

4 Package and prerequisites **11**

II Running the control example **12**

5 Execution **13**

6 Installation and trouble shouting **16**

7 Analysis **17**

III Exploring other control problems **19**

8 Changing the initial condition and the integration time **20**
- 8.1 Controlling the integration . . . 20
- 8.2 Re-starting the integration . . . 20
- 8.3 Changing the initial condition . . . 21

9 Changing the control law **22**

Bibliography **22**

A On the Navier-Stokes solver **26**
- A.1 How to run the program. A short introduction. . . . 26
- A.2 Main directories . . . 27
  - A.2.1 Code_Input . . . 27
  - A.2.2 LIBRARY . . . 31
  - A.2.3 pictures . . . 33
  - A.2.4 V3 . . . 33

A.3 Variables in the program 34
A.4 Structure of the program 36
A.4.1 call read_mesh 36
A.4.2 call read_bc 37
A.4.3 call read_control 37
A.4.4 call setIA 37
A.4.5 call read_restart_steady 37
A.4.6 call read_restart_unsteady 37
A.4.7 7777 CONTINUE 37
A.4.8 call pressure 37
A.4.9 call flow_controller 38
A.4.10 call set_kick 38
A.4.11 call update_unsteady 38
A.4.12 call set_bcV 38
A.4.13 8888 CONTINUE 38
A.4.14 call read_control 38
A.4.15 call read_material 38
A.4.16 call set_bcV 39
A.4.17 call assemble_matrix 39
A.4.18 call set_bc_RD 39
A.4.19 call load 39
A.4.20 call dsys 39
A.4.21 call update_internal 39
A.4.22 call set_bc_RD 39
A.4.23 call set_bcV 39
A.4.24 call resid 39
A.4.25 call write_restart_steady 39
A.4.26 call V2 39
A.4.27 call write_restart_unsteady 39
A.4.28 GO TO 39

B The control script **41**

1. Introduction

1.1 Wake stabilization as flow control benchmark

We propose our fluidic pinball configuration as new wake stabilization benchmark. This flow configuration is geometrically simple, yet comprises most known wake suppression mechanisms. Wake stabilization is a key benchmark for flow control applications since many decades [13] dating back to 1930 [14] and earlier.

The importance of wake control has several reasons. Wakes are a characteristic feature of virtually all bluff bodies, like cars, trucks, trains and buildings [15]. Yet, wakes are associated with undesirable forces and moments. Drag reduction of road vehicles has even become a cornerstone challenge due to the increasing need of the reduction of greenhouse gas emissions and corresponding fuel consumption. Aerodynamic drag represents over 65% of the total power expense [16, 17] at highway speeds. In particular, the low pressure in the wake resulting from the flow separation causes the form drag and constitutes an important portion of the aerodynamic drag for the bluff form vehicles. Hence, the manipulation of wake flow provides a great potential to achieve the drag reduction by increasing the base pressure. For buildings and bridges, unsteady drag forces may give rise to discomfort of the inhabitants, in particular if the accelerations exceed 0.5% of the gravitational acceleration [15, page 408]. The unsteady lift forces may be amplified by resonance by factors up to 1000 [15, page 407] and have lead to numerous destructions of bridges and chimneys.

In particular, wake stabilization around a circular or D-shaped cylinder has enjoyed large popularity as flow control benchmark. For these flows, the steady solution has engineering benefits like reduced drag and vanishing lift forces at potentially vanishing actuation penalty. At the same time, the stabilization of this steady solution allows the application of powerful linear control methods. In addition, wake stabilization is achieved by counteracting periodic vortex shedding. Thus, any active control method leads to phasor control, i.e. is characterized by two parameters, a phase shift with respect to the shedding and the gain of the actuation response. One of the most effective passive means to prevent vortex shedding is a suitable placed splitter plate [18]. Strykowski wires is another low-Reynolds number means [19]. A short yet highly informative summary of passive wake stabilization is provided in Section 2.5 of [15].

1.2 Categories of wake stabilization strategies

In the following, we summarize active wake stabilization mechanisms. First, we categorize the mechanisms by dynamic behavior and not be actuation means.

Phasor control: The earliest experimental wake stabilization [20] employs phasor feedback. The laminar wake around a circular cylinder is sensed downstream by a hot-wire sensor and actuated by two anti-phase load speakers in transverse direction. The measured phase is fed back with a phase shift and a gain, i.e. applying classical phasor control. Physically, the actuation effect can be obtained by a transverse oscillation of the cylinder. The control

has shifted the critical Reynolds number for the onset of vortex shedding from 45 to about 70.

Base bleed: Since over 50 years, steady base bleed into the wake region is reported as effective means for wake stabilization [21, 22]. The physical effect is similar to the splitter plate. The communication between the upper and lower shear layer is suppressed. Thus, an von Kármán vortex cannot occupy the whole near wake region but is pushed away in its infancy.

Boat tailing: The drag can also be reduced by shaping the wake region more aerodynamically, i.e. by vectoring the shear layer towards the center region. This has been performed by passive devices, like vanes [14], or active control via Coanda blowing [23, 24, 9].

High-frequency forcing: Another means of wake stabilization is high-frequency forcing for the energetization of the upper and lower shear layer. This energetization reduces the transverse wake profile gradients and thus the instability of the flow. Another effect of the shear layer excitation is the increase of the effective eddy viscosity on the von Kármán vortices — another damping effect. This shear layer energetization may be achieved with vigorous cylinder rotation [25] or symmetric blowing at the trailing edge [26].

Low-frequency forcing: Also symmetric forcing at a lower frequency than natural vortex shedding may stabilize the wake [8]. In this case, *anti-symmetric* vortex shedding is 'confused' by a forced *symmetric* dynamics and a clock-work which is distinctly out of sync with the shedding period. The advantage of low-frequency forcing is a much lower energy consumption as compared to high-frequency forcing. Both stabilizing effects by high- and low-frequency forcing constitutes an example of frequency crosstalk [10] and is described by a low-dimensional generalized mean-field model [27].

A second categorization may be based on the predominant region of the actuation response — extending a review by Choi et al. [13].

Circulation control: The cylinder wake has also been stabilized using feedback and cylinder rotation [28]. The cylinder rotation changes the circulation around the cylinder. the underlying dynamics is phasor control. The word 'circulation control' is most commonly used for active lift increase of airfoils [29], but the physical principle for wake stabilizations remains the same.

Stagnation point control: The cylinder rotation also changes the location of the stagnation point, as well known for the Magnus effect. Shifting the stagnation point to the lower side gives rise to higher velocities on the upper side and thus larger vorticity production at the expense of the lower side. Evidently, the actuation effect of cylinder rotation implies both, stagnation point and circulation control. Yet, we can draw a clearer separation between both mechanisms for the fluidic pinball as discussed below.

Separation control: The separation line may be enforced by geometry, like the sharp trailing edge of a D-shaped cylinder. For smooth bodies, like the cylinder, the line of separation may be changed. Near critical Reynolds numbers $Re \leq 3 \cdot 10^5$, a tripping wire or many other perturbations may cause transition of the boundary layer and thus a significant delay of separation. This leads to a more narrow wake and a drag reduction over 50% [30].

Wake control: The manipulation of the shear layer and base bleed changes the wake dynamics and may thus inhibit vortex shedding.

A third categorization of wake control is based on the targeted flow state.

Steady solution: A complete stabilization of the steady Navier-Stokes solution comes with the advantage of a lower drag at vanishing actuation energy. In case of noise, this actuation energy increases with the noise level but can still be expected to be small. No experimentally realizable actuator has ever been designed which has enough authority to reach this goal at high Reynolds numbers. Yet, any actuation which elongates the vortex bubble — like for the steady solution — reduces the shear-layer curvature, thus reduces the transverse pressure difference across the shear layer and thus increases the base pressure.

Streamlined wake region: Aerodynamic boat-tailing makes the wake region more streamlined and thus helps in the pressure recovery of the wake region. Most effective drag reductions at high Reynolds numbers are based on boat tailing. The prize is the continuous need for actuation energy.

Potential flow: Eliminating the wake region leads to nearly complete pressure recovery. This has been achieved in a two-dimensional Navier-Stokes simulation of the cylinder flow with wall actuation [31]. Yet, the energy cost was almost as large as the significant drag reduction.

The degree of wake stabilization may be measured in different terms.

Net drag power saving: Arguably the most honest engineering measure is the net saving in parasitic drag power accounting for the actuation energy.

Drag reduction: Another measure is drag alone if actuation power is no issue.

Reduction of fluctuation level in lift: Same applications, like stability of bridges, may require the reduction of lift oscillation. There is no straight-forward assessment for the actuation energy, as changes in lift have no direct energetic benefit.

Length or volume of the backflow region: An increase of the wake region is typically associated with desirable features like lower fluctuation level or lower drag. The advantage is that the backflow region can easily be determined from time-averaged flow, e.g. PIV. However, boat tailing may *reduce* length and volume of the backflow region and reduces drag and fluctuation level. Thus, this performance measure is heavily biased towards favoring the steady solution as target.

Difference to steady solution: For linearized Navier-Stokes frameworks, the averaged distance between the actuated flow state and the targeted steady solution is a natural performance measure. Again, actuation penalization may be included.

Total fluctuation energy: Another performance measure is the total fluctuation energy of the flow. In case of vanishing level and no actuation, the steady solution is reached.

This 'tour de force' through the wake stabilization literature sets the stage for a good problem formulation employing the fluidic pinball in the coming chapters. In the first part (Chapters 2, 3, 4), the appetite is wetted by detailing the control problem, outlining one example and stating the few prerequisites for running the package. The second part (Chapters 5, 6, 7) deals with practical aspects of repeating the discussed control example. In the third part (Chapters 8, 9) the reader is invited to perform other control studies himself. The appendix (Chapters A, B) provides a detailed description of the Navier-Stokes solver and a listing of a control script.

Part I

Capabilities

2. Control problem

2.1 Flow configuration

In this section, the considered two-dimensional flow control problem is described. Three equal circular cylinders with radius R are placed in parallel in a viscous incompressible uniform flow with speed U_∞. The centers of the cylinders form an equilateral triangle with sidelength $3R$, symmetrically positioned with respect to the flow. The leftmost triangle vertex points upstream, while rightmost side is orthogonal to the oncoming flow. Thus, the transverse extend of the three cylinder configuration is given by $L = 5R$.

This flow is described in a Cartesian coordinate system where the x-axis points in the direction of the flow, the z-axis is aligned with the cylinder axes, and the y-axis is orthogonal to both. The origin $\mathbf{0}$ of this coordinate system coincides with the mid-point of the rightmost bottom and top cylinder. The location is described by $\boldsymbol{x} = (x, y, z) = x\boldsymbol{e}_x + y\boldsymbol{e}_y + z\boldsymbol{e}_z$, where $\boldsymbol{e}_{x,y,z}$ are unit vectors pointing in the direction of the corresponding axes. Analogously, the velocity reads $\boldsymbol{u} = (u, v, w) = u\boldsymbol{e}_x + v\boldsymbol{e}_y + w\boldsymbol{e}_z$. The pressure is denoted by p and the time by t. In the following, we assume a two-dimensional flow, i.e. no dependency of any flow quantity on z and vanishing spanwise velocity $w \equiv 0$.

The Newtonian fluid is characterized by a constant density ρ and kinematic viscosity ν. In the following, all quantities are assumed to be non-dimensionalized with cylinder diameter $D = 2R$, the velocity U_∞ and the fluid density ρ. The corresponding Reynolds number reads $Re_D = U_\infty D/\nu$. The Reynolds number based on the transverse length $L = 5D$ is 2.5 times larger. The non-dimensionalization with respect to the diameter appears a more natural description of the near-wake dynamics, while the transverse length is more natural for the far-wake. With this non-dimensionalization, the cylinder axes are located at

$$\begin{array}{llll} x_F &= -3/2\cos 30^\circ = 3\sqrt{3}/4 \approx -1.2990 & y_F &= 0 \\ x_B &= 0 & y_F &= -3/4 \\ x_T &= 0 & y_T &= +3/4. \end{array} \tag{2.1}$$

Here, and in the following, the subscripts 'F', 'B' and 'T'refer to the front, bottom and top cylinder. An alternate reference are the subscripts 1, 2, 3 for the front, bottom and top cylinder, respectively. The numbering is in mathematically positive orientation. We note that the choice of the origin and the non-dimensionalization with respect to the cylinder diameter gives rise to a simple description of the cylinder centers.

2.2 Dynamics

The incompressibility conditions reads

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

where ∇ represents the Nabla operator. The evolution is described by the Navier-Stokes equations,

$$\partial_t \boldsymbol{u} + \nabla \cdot \boldsymbol{u} \otimes \boldsymbol{u} = -\nabla p + \frac{1}{Re_D} \triangle \boldsymbol{u}, \tag{2.3}$$

where ∂_t and $\triangle$ denote the partial derivative and the Laplace operator. The dot '$\cdot$' and dyadic product sign '$\otimes$' refer to inner and outer tensor products.

Without forcing, the boundary conditions comprise a no slip-condition on the cylinder and a free-stream condition in the farfield:

$$\boldsymbol{u} = \boldsymbol{0} \text{ on the cylinder and } \boldsymbol{u} = \boldsymbol{e}_x \text{ at infinity.} \tag{2.4}$$

A typical initial condition is the unstable steady Navier-Stokes solution $\boldsymbol{u}_s(\boldsymbol{x})$.

2.3 Actuation, sensing and cost function

The forcing is exerted by rotation of the cylinders with circumferential velocities $b_1 = U_1 = U_F$, $b_2 = U_2 = U_B$ and $b_3 = U_3 = U_T$ for the front, bottom and top cylinder. The actuation command $\boldsymbol{b} = (b_1, b_2, b_3) = (U_1, U_2, U_3)$ is preferably used for control theory purposes following [10, 11] while (U_F, U_B, U_T) are more natural for physical discussions. The actuation is conveniently expressed with the vector cross product '$\times$':

$$\boldsymbol{u} = 2U_i \, \boldsymbol{x} \times \boldsymbol{e}_z \text{ on the } i\text{th cylinder} \tag{2.5}$$

The factor 2 counterbalances the non-dimensional radius $1/2$.

We like to refer to this configuration as *fluid pinball* as the rotation speeds allow to change the paths of the incoming fluid particles like flippers manipulate the ball of a real pinball. The front cylinder rotation may determine if the fluid particle passes by on the upper or lower side of the cylinder, while the top and bottom cylinder my guide the particle through the anus. Sorry, we mean through the interior of the cylinder configuration.

As experimentally realizable sensing, we propose 5 equidistantly placed u-velocity sensors, 2 lengths $L = 10R$ behind the rearward end $x = 1/2$ of the cylinder configuration, such that the inner three sensors have the heights of the three cylinder centers:

$$s_i = u\left(x_i^s, y_i^s, t\right), \quad x_i^s = 11/2, \quad y_i^s = (3/4)(3-i) \text{ for } i = 1, 2, \ldots, 5. \tag{2.6}$$

The sensor signal are comprised in the vector $\boldsymbol{s} = (s_1, \ldots, s_5)$, following the symbols of [10, 11].

One control optimization may be the minimization of the averaged parasitic drag power $\overline{J}_a$ with penalization of the averaged actuation power $\overline{J}_b$,

$$\overline{J} = \overline{J}_a + \overline{J}_b \stackrel{!}{=} \text{minimum.} \tag{2.7}$$

Let '$\boldsymbol{a}$' be a symbol or discretization of the velocity as flow state. Then, the sensor-based *multiple-input multiple-output* (MIMO) control problem has the structure

$$d\boldsymbol{a}/dt = \boldsymbol{F}(\boldsymbol{a}, \boldsymbol{b}) \tag{2.8a}$$

$$\boldsymbol{s} = \boldsymbol{G}(\boldsymbol{a}, \boldsymbol{b}) \tag{2.8b}$$

$$\boldsymbol{b} = \boldsymbol{K}(\boldsymbol{s}). \tag{2.8c}$$

Here, $\boldsymbol{F}$ comprises the actuated Navier-Stokes dynamics, $\boldsymbol{G}$ defines the measurement equation, and $\boldsymbol{K}$ the sensor-based feedback law minimizing the cost function. For this particular set of sensors, the measurement equation is linear and has no feedthrough term, $\boldsymbol{s} = \boldsymbol{C}\boldsymbol{a}$.

2.4 Direct numerical solution

Finally, the numerical solver is discussed. The chosen computational domain is bounded by the rectangle $[-6, 20] \times [-6, 6]$ and excludes the interior of the cylinders:

$$\Omega = \left\{(x, y) \colon \, -6 \leq x \leq 20 \wedge |y| \leq 6 \wedge (x - x_i)^2 + (y - y_i)^2 \geq 1/4, i = 1, 2, 3\right\}.$$

This flow domain is discretized on an unstructured grid with 4225 triangles and 8633 vertices. This discretization optimizes the speed of the numerical simulation while keeping the accuracy at acceptable level. The Navier-Stokes equation is numerically integrated with an implicit Finite-Element Method [3, 32] The numerical integration is third-order accurate in space and time.

3. Control example

In this section, we present a simple flow control example comprising an unforced transient from steady flow to periodic shedding and a forced transient exhibiting the boat-tailing suppression of vortex shedding. The characteristic shedding period is around 12.5 corresponding to a Strouhal number of $St_W = 0.2$ based on the reference length $W = 5R = 2.5$.

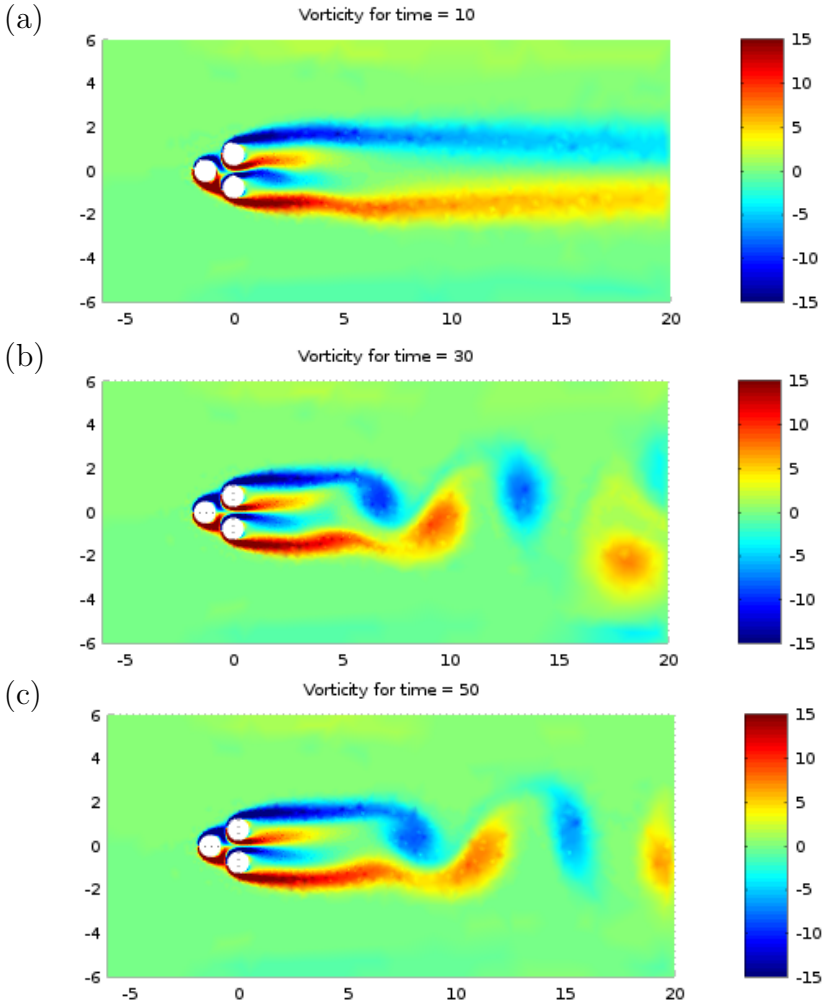

Figure 3.1: Vorticity snapshots of the fluidic pinball from the unforced transient. The figures show (a) an early state $t = 10$, an (b) an intermediate state $t = 30$ and (c) a converged state $t = 50$.

At $t = 0$, the initial condition is set to the steady solution. The front cylinder kick-starts vortex shedding with the expected period. The alternative for this kick-start is waiting forever for numerical errors to accumulate and to start vortex shedding. After about 4 shedding cycles, the top and bottom cylinder start to rotate at constant velocity 4 sucking fluid from the near wake and accelerating the fluid in the outer boundary layer. The corresponding actuation commands

read:

$$U_F = \begin{cases} 1/2 & t \le 6.25 \\ -1/2 & 6.25 < t \le 12.5 \\ 0 & \text{otherwise} \end{cases} \tag{3.1a}$$

$$U_B = -U_T = \begin{cases} 4 & t \ge 50 \\ 0 & \text{otherwise} \end{cases} \tag{3.1b}$$

Figure 3.1 illustrates the vorticity evolution of unforced vortex shedding. At time $t = 10$, the kick has slightly perturbed the initially steady base flow. At time $t = 30$, the vortex shedding is almost converged as evident from the visualization of a later instant $t = 50$. It may be noted that the pinball configuration shows some base bleeding already in the unforced case. This base bleeding elongates the vortex formation region as compared to a cylindrical body of same width.

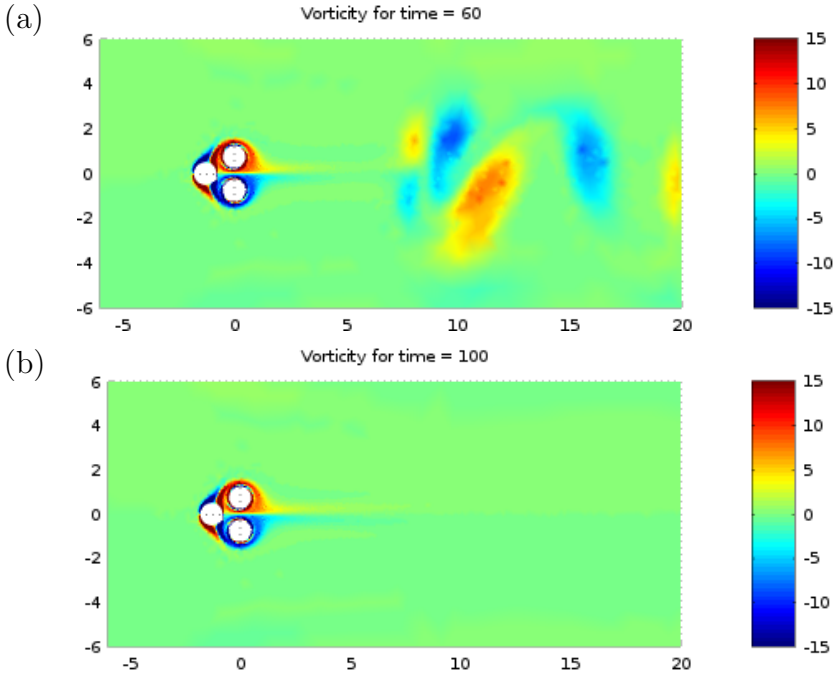

Figure 3.2: Vorticity snapshots of the fluidic pinball from the forced transient. The figures display (a) a state shortly after actuation $t = 60$ and and b) the converged and completely stabilized state $t = 100$.

From figure 3.2, the forcing at time $t \ge 50$ immediately prevents vortex shedding. The obstacle has only a single stagnation point leading to the annihilation of upper and lower vorticity stream. This annihilation gives rise to a nearly potential flow in the wake. The vortices of the von Kármán vortex street are wiped out by the oncoming flow.

The displayed computation with 100 convective time unites takes less than 26 minutes on a moderate Hewlett Packard Laptop of 2016 (Elitebook 820 G3 16Gb RAM, Core i7, proG6, 512 GB SSD). The Navier-Stokes solver has been compiled with Fortran 77 and optimization level 2 (-O2). The 1000 snapshots (10 per convective time unit) requires about 750 MB and 280 MB after gzip compression. The computation time can easily be decreased by level 4 Fortran compilation (-O4) and exploiting multi-threading.

4. Package and prerequisites

The direct numerical simulation of the fluidic pinball can be downloaded following the instructions of our webpage

`http://FluidicPinball.com`

The solver should work under following prerequisites.

- **64bit processor.** The included LAPACK libraries assume a 64bit processor. The pinball code also works on 32bit processors if these libraries are replaced.
- **Linux system.** The original pinball code should work smoothly under reasonably new versions of Linux without any new compilation of the sources. The code has also been successfully tested on Mac's operating system. We did not test the code under Windows and don't expect it to work. If the target computer runs with Windows,

  `http://www.Cygwin.com`

  may offer a solution. Our past experiences with Cygwin range from easy to long, painful installations. Running Linux-based programs on Cygwin may or may not work.
- **Octave (alternatively Matlab)** is a mandatory software for the pinball code. The control logic in `Control.m` is based on octave. Octave is a free software and hence our first choice. The installation can be performed with two commands:

  ```
  > sudo apt-get update > sudo apt-get install octave
  ```

  The first command ensures that the Linux software is updated. And the second command installs octave within seconds without prompting for decisions.
- **Fortran77 compiler.** A Fortran compiler needs to be installed in case the executable for the Navier-Stokes solver `UNS3` does not work. The installation can be performed like with octave.

  ```
  > sudo apt-get update > sudo apt-get install gfortran
  ```
- **File structure of the code.** As preview, the resulting main folder contains four directories

  ```
  Code_Input Code_Output Code_Source LIBRARY
  ```

  and three files

  ```
  UNS3 Control.m Visualization.m
  ```

  Following chapters will describe the files and their use in detail.

Part II

Running the control example

5. Execution

The pinball code can be executed in few steps.

Step 1: Start DNS code. Open a terminal from the main directory and start the DNS code with following command:

```
> ./UNS3
```

The code will start and stop after few seconds with following output:

```
> ./UNS3
 ========================
 FEM-Unsteady-Steady-Eigen CFD-Solver Version:2016
 ========================
        8633  nodes read for  Anzahl der Knotenpunkte und die x,y Koordinaten
        4225  elements read
         522  Boundary conditions
 last degree of freedom        19469
 IA matrix set
 Restart values read from Restart_steady
 No restart file Restart_unsteady
 Starting assembling
 Matrix assembled
 RESVx=   1.1048939541069558E-012   RESVy=   5.7820415122478153E-013
 ################################################
 RMAX=     1.1048939541069558E-012   TIME=   0.0000000000000000
 NCOUNT=           1
 ################################################
Enter fill-in  : Enter tolerance: Elements in A:     569107
       778      1557      2336      3115      3893      4672      5451      6230
 7008      7787      8566      9345     10123     10902     11681     12460
13238     14017     14796     15575     16353     17132     17911     18690     19469
Elements in L:     851587
Elements in U:    1258219
The residual norm has converged.
 Ndecompose=          20  Decompose= T
 RESVx=   2.4433023121589749E-014   RESVy=   5.0091058634576691E-014
 ################################################
 RMAX=     5.0091058634576691E-014   TIME=   0.0000000000000000
 NCOUNT=           1
 ################################################
 Plot for time=   0.0000000000000000       started
 Plot for time=   0.0000000000000000       ended
 Starting assembling
 Matrix assembled
 RESVx=   4.7908343958624755E-012   RESVy=   1.2567724638756772E-012
 ################################################
 RMAX=     4.7908343958624755E-012   TIME=  0.10000000000000001
 NCOUNT=           1
 ################################################
Enter fill-in  : Enter tolerance: Elements in A:     569107
       778      1557      2336      3115      3893      4672      5451      6230
```

```
 7008      7787      8566      9345     10123     10902     11681     12460
13238     14017     14796     15575     16353     17132     17911     18690     19469
Elements in L:     851587
Elements in U:    1258219
The residual norm has converged.
 Ndecompose=          20  Decompose= F
 RESVx=   2.4894217332339409E-014   RESVy=   6.0737717522015959E-014
 ################################################
 RMAX=     6.0737717522015959E-014   TIME=  0.10000000000000001
 NCOUNT=           1
 ################################################
 Plot for time=  0.10000000000000001       started
 Plot for time=  0.10000000000000001       ended
```

The DNS program will terminate the computation is researched.

Step 2: Start control script. Open another terminal from the main directory and start the control script with following command:

```
> octave Control.m
```

You can follow the computation of new time steps:

```
> octave Control.m
T =  0.30000
T =  0.40000
T =  0.50000
T =  0.60000
T =  0.70000
T =  0.80000
```

Step 3: Start visualization script. If you want to see an online visualization, open a third terminal from the main directory and start the visualization script with following command:

```
> octave Visualization.m
```

A window will pop up and show the latest vorticity field every few seconds.

Figure 5.1 illustrates how the DNS code and control script interact. When the DNS has integrated one time step

A) It writes the flow state in `Control_Input.dat`,

B) it creates an empty barrier file `Control_Barrier`.

In the meantime, the control script

C) is continually checking the existence of the barrier file.

D) When it discovers the barrier file, it reads `Control_Input.dat`.

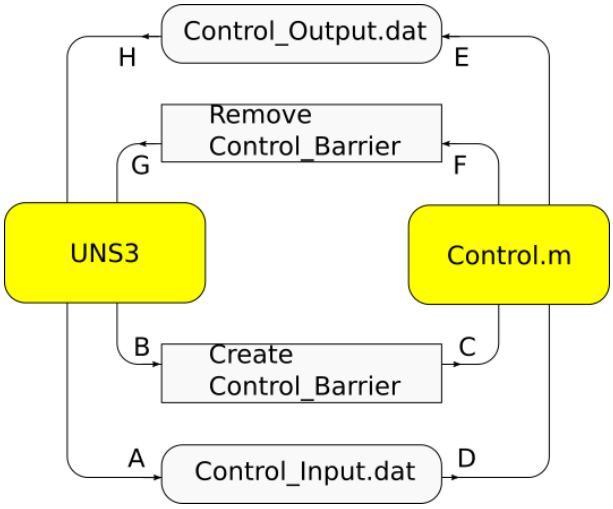

Figure 5.1: Interaction between DNS code and control script. For details see text.

The creation of a barrier file might seem unnecessary. However, it guarantees that the flow state is only read when completely written out. Otherwise, the reading of the control script may start during the writing of the DNS code which will interrupt the cycle with an error.

When the control script has computed the actuation commands,

E) it writes the actuation parameters in `Control_Output.dat`,

F) and removes `Control_Barrier` afterwards.

In the meantime, the DNS continually checks the existence of the barrier file.

G) When it detects no barrier file,

F) it loads the actuation parameters from `Control_Output.dat`, and starts the next integration step.

Now the control script waits for the barrier file and so on.

6. Installation and trouble shouting

After the download of the pinball code, as described in Chapter 4, the simulation can be performed following Chapter 5.

Under few conditions, additional steps may be necessary.

The control script `Control.m` is not working. Up to know, there has been no adverse incident with the octave control script on over 10 different computers. As alternative, the script may be started in an open window of octave. Note that there may be subtle differences between octave and Matlab commands. We do not regularly check Matlab compatibility.

The DNS code `UNS3` cannot be executed. In this case, the Fortran code may need to be re-compiled:

```
> cd Code\_Source
> rm *.o
> make
> mv UNS3 ..
> cd ..
```

The compilation takes about 10 seconds on a modern Laptop.

The DNS code cannot be re-compiled. Perhaps, the `LAPACK` library is not compatible with an old computer, e.g. 32bit processor. In this case, the `LIBRARY` folder needs to be replaced. Perhaps, the `LAPACK` library is not compatible with the Mac Operating System. This also requires a replacement of the same folder. In some cases the `LAPACK` library needs to be re-compiled. Some versions of the pinball, are equipped with the `LAPACK` folders for 32bit processors and for Mac OS.

If nothing works... In this case, send the main developer Marek `Morzynski@virtual.edu.pl` an email with the error listing.

7. Analysis

The numerical solution is stored in the directory `Code_Output`, i.e. where one would expect it. This directory contains following files after a typical completed run with 100 convective units and time step 0.1.

```
Grid.dat
Flow.dat
Flow.000000
Flow.000001
$\ldots$
Flow.001000
```

`Grid.dat` contains the grid, `Flow.dat` the latest computed snapshot, and `Flow.000000`, ... `Flow.0010000` the 0th to 100,000th snapshot respectively.

Grid.dat: The Grid has `xAMC` format [33]:

```
   8633           1           1           3
-1.7979795000000001       -3.2701563099999997E-002
-1.8300103000000001       -1.8026761700000001E-002
-1.7990500000000000        0.0000000000000000
...

     16900           3
         3           2           1
         1           2           6
...
```

The first integer of the first line is the number of nodes $N_\bullet = 8633$, followed by a listing of the x- and y-coordinates of all nodes

$$x_i \quad y_i \quad i \in \{1, \ldots, N_\bullet\}$$

The first integer of the next line $N_\triangle = 16900$ is the number of triangles of the mesh, followed by the indices of the grid nodes defining the triangles

$$i_l \quad j_l \quad k_l \quad l \in \{1, \ldots, N_\triangle\}$$

,

Flow.dat, Flow.000000, ... These snapshot files contain the velocity components and pressure values at all grid nodes:

$$u_i \quad v_i \quad p_i \quad i \in \{1, \ldots N_\bullet\}$$

The first lines of one snapshot reads

```
0.00000000000000000000E+00   0.00000000000000000000E+00   0.93189892642285332425E+00
0.32485090461131155282E-01  -0.12913678282816759210E-01   0.92080940230062868768E+00
0.00000000000000000000E+00   0.00000000000000000000E+00   0.94172613875945154760E+00
    ....
```

The authors have developed a number of tools for post-processing:

Visualization. The enclosed octave script `Visualization.m` computes and visualizes the vorticity field.

Forces on the three cylinders. On tool of Guy Yoslan Cornejo Maceda computes the instantaneous drag and lift forces on each cylinder from a snapshot.

POD modeling. Daniel Fernex is pushing `xROM`—a tool for POD modeling which can be downloaded from

`http://www.xROM.info`

Cluster-based reduced-order modeling. Eurika is pushing `CROM`, a tool for cluster-based reduced-order modeling [34]. More information is available from

`http://www.ClusterModeling.com`

Many more tools are available or will be available soon.

Part III

Exploring other control problems

8. Changing the initial condition and the integration time

In this chapter, we discuss changing the Navier-Stokes problem. First (Section 8.1), the specification of the integration time is discussed. In Section 8.2, we outline how a stopped integration may be continued. Finally, in Section 8.3, changes of the initial condition are addressed.

8.1 Controlling the integration

The integration time is controlled in the file `Code_Input/CONTROL`, i.e. in the ASCII file `CONTROL` in the directory `Code_Input`. A complete listing is below.

```
*Control data
0.0          T0
0.1          dt
100          Tmax
20           NDecompose each N time steps
.1           PlotTime   each N time steps
100          NMAX  Maximum number of iterations in the internal loop
1.0E-6       RRHSMAX Maximum residuum for Navier-Stokes
1.0e-5       REVMax Maximum residuum for the eigenvalue vector
0            ieig     1 - eigen  computations 0 - steady or unsteady computations
0            isteady  1 - steady computations 0 - unsteady computations
3            NEig    Number of eigenvalues to be calculated
1            shift= (nHz * 2 pi ) ^ 2 shift value in Hz
.false. .true.       .false.  .true.       kickme
0            Nvf Number of volume forces
.TRUE. CAYLEY
-0.1 Alfa1
0.5 Alfa2
0.0 Alfa3
10 NEig
.FALSE. Verbose
NOT IMPLEMENTED NOW ! .FALSE. symmetry BC on y=0
```

`T0` and `Tmax` control the initial and final time of the integration. `dt` is the time step for numerical integration. We ask the reader not to change the time step since too long time steps may lead to divergence of the algorithm and too short time steps imply inefficiency. We also ask the reader not to change any of the other parameters at this point. The appendix chapter A outlines more powerful features of the code for which the remaining parameters are employed.

8.2 Re-starting the integration

A stopped integration can easily be resumed. Set the initial condition with

```
> cd Code_Input
```

```
> mv Restart_Unsteady_T Restart_Unsteady
```

Restart_unsteady_T is the latest flow state saved from the unsteady Navier-Stokes simulation. The code will start from the time and flow archived in Restart_Unsteady, if it finds this file. Now re-start the simulation like a new one following Chapter 5. The code will continue the simulation from the latest time.

8.3 Changing the initial condition

The integration can start from several initial conditions archived in the directory Code_Input.

Unsteady snapshot: If there is a file Restart_unsteady in this directory, the code will choose this state as initial condition.

Steady solution: If there is no file Restart_unsteady, the code will search for a steady solution in Restart_steady_0100. The number 0100 refers to the Reynolds number with respect to an indivial cylinder and may, of course, be changed by the user.

Uniform flow: If there is neither an unsteady nor a steady solution to start with, the velocity field is initialized to be the oncoming flow.

The employed initial condition can be seen from the first few lines of the code listing. For instance, the output

```
> ./UNS3
 =========================
 FEM-Unsteady-Steady-Eigen CFD-Solver Version:2016
 =========================
        8633  nodes read for  Anzahl der Knotenpunkte und die x,y Koordinaten
        4225  elements read
         522  Boundary conditions
 last degree of freedom        19469
 IA matrix set
 No restart file Restart_steady
 Restart values read from Restart\_unsteady
```

shows that Restart_unsteady has been read and will be used, i.e. the integration will start from an unsteady flow snapshot.

Unfortunately, writing the initial condition files is not that easy. Each file contains 3 subsequent velocity fields to enable a third-order accurate time integration using the flow history.

If the desired initial condition is, say, at time $t = 35$ of a specific simulation, the integration may be stopped there: Set Tmax=35 in CONTROL or do a brute-force stop and archive Restart_unsteady_T from the Control_Input directory.

A more elegant way is to study the code in Appendix A and write a little program for creating Restart_unsteady.

9. Changing the control law

The control of the cylinders is commanded in the octave/Matlab script `Control.m`. A listing for the example of Chapter 3 is provided in Appendix B.

The control script performs following steps

(1) Initialization of the run (lines 1–3). When the simulation is started, the barrier file is removed and the time is set to 0. In the following, the correct time will be read from `Control_Input.dat`.

(2) Start an effective endless loop (line 7). Now, the script starts an effective endless loop.

(3) Wait for the barrier file (lines 12–16). Continue after the file is found to exist.

(4) Read the flow state (until lines 49). Then, the script reads the `Control_Input.dat` file. This files contains the grid, the flow snapshot, the (hitherto not used) volume force, etc. This information is needed for full-state feedback with the potential use of volume forces.

(5) Determine the cylinder rotation (lines 63–78). Evidently, these lines implement the control law of Chapter 3.

(6) Enforce the cylinder rotation as boundary condition (line 80–109). This appears unnecessarily complex, as the boundary condition may be computed in the Navier-Stokes solver. However, we gain actuation flexibility beyond cylinder rotations. For instance, we can impose blowing and suction on the cylinders, and an unsteady oncoming flow.

(7) Write `Control_Output.dat` (line 114–118). This file contains the imposed actuation, i.e. the new boundary condition and the volume forces (not used here).

(8) Remove `Control_Barrier` (line 122). This tells the Navier-Stokes solver that the integration can be resumed.

(9) Line 125 marks the end of the effective endless loop.

This control script encodes one example of an open-loop control. A small variation may yield any other open-loop control. For closed-loop control, the flow field `FFF` needs to be employed. Evidently, closed-loop control requires programming.

Bibliography

[1] M. Morzyński. Numerical solution of Navier-Stokes equations by the finite element method. In *In Proceedings of SYMKOM 87, Compressor and Turbine Stage Flow Path - Theory and Experiment*, pages 119–128, 1987.

[2] M. Morzyński and F. Thiele. Solution of the eigenvalue problems resulting from global non-parallel flow stability analysis. *Computer Methods in Applied mechanics and Engineering*, 169:161–176, 1991.

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

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

[5] J. Gerhard, M. Pastoor, R. King, B. R. Noack, A. Dillmann, M. Morzyński, and G. Tadmor. Model-based control of vortex shedding using low-dimensional Galerkin models. In *33rd AIAA Fluids Conference and Exhibit*, Orlando, Florida, USA, June 23–26, 2003, 2003. Paper 2003-4262.

[6] G. Tadmor, O. Lehmann, B. R. Noack, L. Cordier, J. Delville, J.-P. Bonnet, and M. Morzyński. Reduced order models for closed-loop wake control. *Philosophical Transactions of the Royal Society A*, 369(1940):1513–1524, 2010.

[7] B. R. Noack, M. Morzyński, and G. Tadmor. *Reduced-Order Modelling for Flow Control*. Number 528 in CISM Courses and Lectures. Springer-Verlag, Vienna, 2011.

[8] M. Pastoor, L. Henning, B. R. Noack, R. King, and G. Tadmor. Feedback shear layer control for bluff body drag reduction. *J. Fluid Mech.*, 608:161–196, 2008.

[9] D. Barros, J. Borée, B. R. Noack, A. Spohn, and T. Ruiz. Bluff body drag manipulation using pulsed jets and Coanda effect. *J. Fluid Mech.*, 805:442–459, 2016.

[10] S. L. Brunton and B. R. Noack. Closed-loop turbulence control: Progress and challenges. *Appl. Mech. Rev.*, 67(5):050801:01–48, 2015.

[11] T. Duriez, S. Brunton, and B. R. Noack. *Machine Learning Control — Taming Nonlinear Dynamics and Turbulence*. Number 116 in Fluid Mechanics and Its Applications. Springer-Verlag, 2016.

[12] M. Morzyński and B. R. Noack. PinBall — a Toolkit for Multiple-Input Multiple-Output Flow Control (Version 0.1). Technical Report 01/2016, Chair of Virtual Engineering, Institute of Combustion Engines and Transport, Poznan University of Technology, Poland, 2016.

[13] H. Choi, W.-P. Jeon, and J. Kim. Control of flow over a bluff body. *Ann. Rev. Fluid Mech.*, 40:113–139, 2008.

[14] G. Flügel. Ergebnisse aus dem strömungsinstitut der technischen hochschule danzig. In *Jahrbuch der Schiffbautechnischen Gesellschaft*, pages 87–113. Springer, 1930.

[15] W.-H. Hucho. *Aerodynamik der stumpfen Körper. Physikalische Grundlagen und Anwendungen in der Praxis*. Vieweg Verlag, Wiesbaden, 2002.

[16] W.-H. Hucho. *Aerodynamics of Road Vehicles*. Society of Automotive Engineers, 1998.

[17] R. C. McCallen, Kambiz Salari, J. M. Ortega, L. J. DeChant, B. Hassan, C. J. Roy, W. D. Pointer, F. Browand, M. Hammache, T. Y. Hsu, et al. Doe's effort to reduce truck aerodynamic drag–joint experiments and computations lead to smart design. In *AIAA Paper, 2014-2249*, 2004.

[18] A. Roshko. On the wake and drag of bluff bodies. *Journal of the Aeronautical Sciences*, 22:124–132, 1955.

[19] P.J. Strykowski and K.R. Sreenivasan. On the formation and suppression of vortex 'shedding' at low Reynolds numbers. *J. Fluid Mech.*, 218:71–107, 1990.

[20] K. Roussopoulos. Feedback control of vortex shedding at low Reynolds numbers. *J. Fluid Mech.*, 248:267–296, 1993.

[21] C. J. Wood. The effect of base bleed on a periodic wake. *Journal of the Royal Aeronautical Society*, 68(643):477–482, 1964.

[22] PW Bearman. The effect of base bleed on the flow behind a two-dimensional model with a blunt trailing edge. *Aeronautical Quarterly*, 18(03):207–224, 1967.

[23] D. Geropp. Process and device for reducing the drag in the rear region of a vehicle, for example, a road or rail vehicle or the like. United States Patent **US 5407245 A**, 1995.

[24] D. Geropp and H.-J. Odenthal. Drag reduction of motor vehicles by active flow control using the Coanda effect. *Exp. Fluids*, 28(1):74–85, 2000.

[25] B. Thiria, S. Goujon-Durand, and J. E. Wesfreid. The wake of a cylinder performing rotary oscillations. *J. Fluid Mech.*, 560:123–147, 2006.

[26] A. R. Oxlade, J. F. Morrison, A. Qubain, and G. Rigas. High-frequency forcing of a turbulent axisymmetric wake. *J. Fluid Mech.*, 770:305–318, 2015.

[27] D. M. Luchtenburg, B. Günter, B. R. Noack, R. King, and G. Tadmor. A generalized mean-field model of the natural and actuated flows around a high-lift configuration. *J. Fluid Mech.*, 623:283–316, 2009.

[28] B. Protas. Linear feedback stabilization of laminar vortex shedding based on a point vortex model. *Phys. Fluids*, 16(12):4473–4488, 2004.

[29] R. E. Walters, D. P. Myer, and D. J. Holt. Circulation control by steady and pulsed blowing for a cambered elliptical airfoil. Technical report, West Virginia University, Aerospace Engineering TR-32, 1972.

[30] G. Schewe. On the force fluctuations acting on a circular cylinder in crossflow from subcritical up to transcritical Reynolds numbers. *J. Fluid Mech.*, 133:265–285, 1983.

[31] J. Wu, L. Wang, and J.Z. Tadmor. Suppression of the von karman vortex street behind a circular cylinder by a traveling wave generated by a flexible surface. *J. Fluid. Mech*, 574:365–391, 2007.

[32] B. R. Noack, W. Stankiewicz, M. Morzyński, and P. J. Schmid. Recursive dynamic mode decomposition of transient and post-transient wake flows. *J. Fluid Mech.*, 809:843–872, 12 2016.

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

[34] E. Kaiser, B. R. Noack, L. Cordier, A. Spohn, M. Segond, M. W. Abel, G. Daviller, J. Östh, S. Krajnović, and R. K. Niven. Cluster-based reduced-order modelling of a mixing layer. *J. Fluid Mech.*, 754:365–414, 2014.

A. On the Navier-Stokes solver

The following sections provide indepth information about the Navier-Stokes solver and associated files. This chapter may be useful for readers who want to use the full capacity of the code, like computing the steady solution or performing a stability analysis.

A.1 How to run the program. A short introduction.

Compile the program with make in DNS-quadratic-2.0 directory.

Prepare the steady solution or the restart you want to start with. It can be done by setting appropriate Re number in Code_Input/PROPERTIES file and setting isteady to 1 in Code_Input/CONTROL.

Run uns3, copy Restart_steady_xxxx to Restart_steady in Code_Input. "xxxx" denote the actual Re number, Restart_steady_xxxx is the output file for the computations, the file without extension (Restart_steady) is the current restart file. Obtaining high Re can require few intermediate steps (Reynolds numbers), the usual CFD rules apply.

Switch to unsteady computations by setting isteady to 0. In some situations a kick can be needed, if unsteady computations involve rotation no kick is necessary. In all cases bare in mind possible consequences (e.g. starting incompressible flow motion from rest to some finite velocity in one (almost zero) time step is called water hammer !).

If there is any Restart_unsteady file, the solver reads in the unsteady restart. If it is not present the flow solver starts from the steady solution. If there is no steady solution the code assumes start from the rest. The convention of unsteady restarts is similar to steady ones, the output of each time step is written to Restart_unsteady_T which has to be copied to Restart_unsteady to be used to start computations.

Once all the input files are prepared, start in one terminal uns3 with ./uns3 and in another terminal window Octave or Matlab.

Introduce executable octave file oct.m (no semicolon at the end of line and no .m) into Octave window.

For Matlab use matl.m file exactely in the same way.

Files oct.m and matl.m contain extensive comments and are self-explanatory. All the flow control can be applied in these files.

Use SSD disk to write and read if possible to speed up the computations.

Output is written to "pictures" directory in ASCII. Names of the files Flow.xxxxxxx are related to time step, e.g. Flow.000122 denotes result for T=12.2. There is also a version of visualisation subroutine in x_m_v_t3.f writing in (old) flowfem.xxxx scientific notation, presently commented.

A.2 Main directories

A.2.1 Code_Input

Code_Input directory contains the files described in following subsections.

BOUNDARY6

All boundary conditions are now set in the BOUNDARY6 file. The example content of the BOUNDARY6 file:

Boundary conditions (Text line)

145

1125 1 1 1. 0.

1126 1 1 1. 0.

1263 1 1 1. 0.

1279 1 1 1. 0.

1280 1 1 1. 0.

1282 1 1 1. 0.

.....

The file reads as follows:

145 is the number nodes with prescribed boundary conditions. 1125 is the node number. First two integers (1 1) mean what boundary conditions are set. For example 1 1 means both x and y velocities are prescribed, 0 1 would mean only V_y velocity is prescribed.

Next two real numbers (1. 0.) mean the value of the (Dirichlet) boundary condition. The pair (1. 0.) means $V_x = 1$, $V_y = 0$.

The number of lines is equal to number of nodes with prescribed boundary conditions. Nodes on the boundary of the domain not listed in the BOUNDARY6 file are assumed to have stress-free boundary condition ("outflow boundary"). BOUNDARY6 denotes boundary conditions for 6-node triangle mesh, quadratic version of the program, for qubic FEM one should use BOUNDARY10 (10-node triangle).

GRID6

The GRID file has following structure:

Circular cylinder grid (any text)
2550
15. -4.375
15. -5.
14.375 -5.
14.5788202 -4.57416391
13.9538202 -4.57416391
13.75 -5.
13.125 -5.
13.4419699 -4.64355803
13.6457901 -4.21772099
.....
Connectivity (Any text)
1234
11 2 14 4 1 13
11 6 2 5 3 4
11 22 6 9 8 5
22 18 6 17 7 8
22 11 30 9 10 24
....
The first line is text, then (integer) the number of nodes (NDE), and pairs of x and y coordinates of the node. Further after NDE lines of coordinates a line with text, number of (TRIA6) elements and connectivity matrix (first 3 values in the line are corner nodes, further 3 midside nodes).

PROPERTIES

For present version of the code only Re is relevant. It is the third line of the file, $100.$ in the example below.

The remarks about penalty parameter and bulk viscosity are relevant only for penalty version of the code, not used in the present computations. For compatibility reasons the structure of the file is preserved.

Two lines of (arbitrary) text are followed by 3 real numbers denoting Reynolds number, penalty parameter (some large number of order $10^5 - 10^6$or more in case of poor convergence) and "bulk viscosity" - irrelevant for incompressible computations but significantly smoothing the

solution. "Bulk viscosity" should be kept between 0 (no smoothing) to 50% of Reynolds number (practically found optimum). The example file is as follows:

```
*Properties
* Re    eps    Bv
100.    10000.    50.
```

CONTROL

The file contains control values *for the computations*. Do not mix it with flow control. The file below shows the example values and their description. Not all values are used presently. The most important switches are isteady (0 means unsteady computation, 1 is used for steady solution) and kickme (.true. or .false. , triggers the "kick" according to set_kick.f subroutine). Eigensolver in this version is not incorporated (ieig = 0).

```
*Control data
0.0 T0
0.1 dt
100 Tmax
10 NDecompose each N time steps
.1 PlotTime each N time steps
100 NMAX Maximum number of iterations in the internal loop
1.0E-6 RRHSMAX Maximum residuum for Navier-Stokes
1.0e-5 REVMax Maximum residuum for the eigenvalue vector
0 ieig 1 - eigen computations 0 - steady or unsteady computations
0 isteady 1 - steady computations 0 - unsteady computations
3 NEig Number of eigenvalues to be calculated
1 shift= (nHz * 2 pi ) ^ 2 shift value in Hz
.false.  .true.  kickme
2 Nvf Number of volume forces
```

SOLVER

To learn exactly the parameters found in the SOLVER file one should refer to the documentation of the QMRPACK library. The file contains:

```
1 Algorithm
1.0D-6 convergence tolerance
```

```
500 Nlim - number of iterations
1.0 estimated norm for P&Q
1.0 estimated norm for V&W
1 Precon
250 fill-in
0.0 tolerance
1.0 SSOR parameter OMEGA - not used for ILU
*****************************************************************
Choices of algorithm :  1 = CPL
estimated norm for P&Q
estimated norm for V&W
2 = CPX
3 = QBG
4 = QMR
estimated matrix norm
5 = QMX
6 = TFX
1 = Left ILUT
2 = Right ILUT
3 = Two-sided ILUT
fill-in
tolerance
4 = Left SSOR
5 = Right SSOR
6 = Two-sided SSOR
SSOR parameter OMEGA
```

The parameters set here are to the authors knowledge a kind of optimum. One can experiment with "convergence tolerance" - the residuum of iterative linear equations solver and fill-in. The another parameter which could be change is fill-in. Fill-in is the parameter characterizing the width of the band of the preconditioning matrix. The larger fill-in the closer is the procedure to complete LU decomposition. It is better in subsequent iterations but costs more on the decomposition stage. For large problems memory limitations can force fill-in to be of order of 50 or even less. It should be remarked that already now the alternative solvers were used, including the parallel versions.

Restart files

All restart files are now ASCII files. The temporary, convergent restart file for unsteady computations is stored on each time step in "Restart_unsteady_T". To use it as a restart one copies it to the file "Restart_unsteady". Steady solution is stored in "Restart_steady_#####" where ##### is the Reynolds number. To use the steady solution one should copy "Restart_steady_#####" to "Restart_steady" file in Code_Input directory.

A.2.2 LIBRARY

Contains the library files for the solver. The directory contains files compiled for Linux machines:

- blas.a
- lapack.a
- libalgs.a
- liblinp.a
- libmisc.a
- libsplib.a

Compilation from source requires QMRPACK and LAPACK libraries (from NETLIB or morzynski@virtual.edu.pl on request).

The compilation of QMRPACK is done by running "setup" script. It is important to use -traditional (-pedantic) flag for gcc. The output from the "setup" in QMRPACK should look like:

```
Looking for the C preprocessor cpp...
Amazingly, I found it.
Enter any C preprocessor flags that might be needed [none]:  -traditional
(or try -pedantic)
Looking for the FORTRAN compiler f77...
I cannot find f77.
Try another name (y/n)?  [y]
Enter the correct name or location [f77]:  gfortran
Wow, I found the FORTRAN compiler.
Enter any FORTRAN compiler flags that might be needed [none]:
I'll make the header file for the makefiles...  OK.
.....
```

After compilation necessary libraries (files lib*.a) are created in the subdirectories of QMRPACK. They should be copied to the LIBRARY directory.
Compilation of LAPACK delivers blas-some-machine.a and lapack-something-some-machine.a libraries in LAPACK dir. The files should be copied as the files as blas.a and lapack.a to LIBRARY directory. LAPACK is tested to work with the following Makefile.in:

```
############################################################################
# LAPACK make include file.  #
# LAPACK, Version 3.0 #
# June 30, 1999 #
############################################################################
#
SHELL = /bin/sh
#
# The machine (platform) identifier to append to the library names
#
PLAT =
#
# Modify the FORTRAN and OPTS definitions to refer to the
# compiler and desired compiler options for your machine.  NOOPT
# refers to the compiler options desired when NO OPTIMIZATION is
# selected.  Define LOADER and LOADOPTS to refer to the loader and
# desired load options for your machine.
#
FORTRAN = gfortran
#FORTRAN = /opt/intel/compiler60/ia32/bin/ifc
#OPTS = -O4 -u -f -mt
#OPTS = -u -f -dalign -native -xO5 -xarch=v8plusa
OPTS =
DRVOPTS = $(OPTS)
NOOPT = -u -f
#NOOPT = -u -f -mt
LOADER = gfortran
#LOADER = /opt/intel/compiler60/ia32/bin/ifc
LOADOPTS =
```

```
#LOADOPTS = -f -dalign -native -xO5 -xarch=v8plusa
#
# The archiver and the flag(s) to use when building archive (library)
# If you system has no ranlib, set RANLIB = echo.
#
ARCH = ar
ARCHFLAGS= cr
RANLIB = echo
#
# The location of the libraries to which you will link.  (The
# machine-specific, optimized BLAS library should be used whenever
# possible.)
#
BLASLIB = ../../blas$(PLAT).a
LAPACKLIB = lapack$(PLAT).a
TMGLIB = tmglib$(PLAT).a
EIGSRCLIB = eigsrc$(PLAT).a
LINSRCLIB = linsrc$(PLAT).a
```

~

A.2.3 pictures

The output is written to the "pictures" directory in ASCII format. Names of the files Flow.xxxxxxx are related to time step, e.g. Flow.000122 denotes result for T=12.2.

For old convention files flowfem.#### are written, where #### is the time. One can save only some (each nth) time step, by setting PlotTime in the Code_Input/CONTROL file. The structure of the file is $(p,\, V_x,\, V_y)$ for subsequent nodes.

A.2.4 V3

There is a MF32TEC.f file and Makefile producing binary MF. This program should write ASCII TECPLOT file TEC.plt or is a hint how to construct visualisation for any commercial or Open Source visualisation programs. Note, in present version only the last file is visualised, namely Flow.dat.

A.3 Variables in the program

- ndf - number of degrees of freedom in the node
- nen - number of nodes in the finite element
- ndm - number of dimensions
- NDEmax - maximal allowed number of nodes in the program (Warning: it appears also in GM under the name NGridP !!)
- NDIM - number of DOFs (dimensions), NDIM=MDEmax*Ndm
- NELmax - maximal allowed number of elements
- NZMAX, NZLMAX, NZUMAX - Number of nonzero entries in the matrix, lower and upper part of LU decomposed matrix - important only for solver
- MatMax, MaxProp - maximum number of materials and properties, important only for structural solver
- MaxCon maximal number of elements connected in one node
- MaxEig - maximal number of eigenmodes allowed
- NvfP - maximum number of volume forces
- Nvf -number of volume forces in the actual problem
- NDE - number of nodes in the actual problem
- NGrid = NDE number of nodes in the GM, for historical reasons different name
- NEL - number of elements in the actual problem
- nbcV - number of boundary conditions (Dirichlet)
- nbcF - number of boundary RHS boundary conditions (loads) not used for flow solver
- NMAX - maximal allowed number of Navier-Stokes iterations
- shift, REVMax - parameters for eigencomputations
- ul() - local (element level) values of velocities
- xl() - local (element level) values of nodes coordinates
- F(),F1(),F2() vectors of unknowns (velocities) in time T,T-dt and T-2*dt. Only F(*,1) is used even if F(*,5) is reserved (for future purposes).
- U1(),V1() - velocities (the same as in F() written in different format)
- RD() residuum (global)

- IA(), JA(),ITMP() A() MAXN, MAXPQ, MAXVW, JT(), IDA(), ILMAT(), IU(NDIM+1), JI(NDIM), JLMAT(), JU(), AINV(), DR(), LMAT(), UMAT(), DN(), DS() VECS(), XZR() - matrices and vectors used by the linear equation solver.
- x() - nodal coordinates, also XX(), YY() contain (for historical reasons) nodal coordinates
- NOPP() - Pointers to the first degree of freedom in each node (global)
- NOPPL() - Pointers to the first degree of freedom in each node (local)
- MDF() - number of DOFs in each node, for penalty formulation identically equal 2
- MEN() number of nodes per element
- IX() connectivity matrix
- Neighbour() - matrix containing the neighbouring nodes (for the solver)
- AF() - FEM element matrix
- RDL() - local (element level) residuum vector
- VBC() - values of Dirichlet boundary conditions
- KBC() - first column is the node number of Dirichlet BC, the further two columns contain 1 or 0 depending on if the condition is active or not
- FBC(), LBC() - boundary condition in RHS, not used presently
- DMat(), MatGroup() - material (properties), only 3 values are important in flow case, in principle each element can have different values (structural solver)
- NEWMAT - logical value, determines if LU decomposition is done by solver
- DT - time step
- T - current time
- Tmax - maximum time for computations
- RRHSMAX - maximum value in the RHS vector
- RMAX - maximum error
- VBCo() - values of BC in the "laboratory" frame
- VBC() - actual values of BC in "body" frame
- vx(),vy(),vx1(),vy1(),vx2(),vy2() - local (element level) values of velocities in time T, T-dt, and T-2*dT
- vxl(),vyl(),vxo(),vyo() - local (element level) values of velocities used by disturbance equation

- VolX(),VolY() - local (element level) values of volume force
- VolXg(),VolYg() - global values of volume force
- Vol_X(), Vol_Y() the volume forces read in (to form circle, or any shape, several forces, given in all nodes - most of the value can be 0.0)
- PRESS() pressures computed by the Poisson solver on base of velocities
- Re Reynolds number
- eps - penalty parameter
- Bv - bulk viscosity
- kickMe - logical value to disturb the flow by blowing at cylinder surface

A.4 Structure of the program

Main program can be found in x_m_v_t3.f file.

A.4.1 call read_mesh

The routine reads mesh and connectivity matrix.

```
READ(Lgrid,*) NDE                ! Read number of nodes
READ(Lgrid,*) (x(j,n),j=1,ndm)       !  ndm =2
```

Coordinates are stored for historical reasons in x() and XX(),YY().

```
XX(i)=x(1,i)
YY(i)=x(2,i)
READ(Lgrid,*) NEL          ! Read number of elements
Read(Unit=InputLine, FMT=* ) (IX(k,i),k=1,MEN(i))
```

In IX() matrix the connectivity is stored.

```
if(j.le.3) MDF(IX(j,i)) = 3
if(j.gt.3) MDF(IX(j,i)) = 2
```

MDF(i) denotes number of DOFs at the node i, corner nodes have 3 DOFs (Vx,Vy,P), middle nodes have 2 DOFs (Vx,Vy).

A.4.2 call read_bc

The routine reads boundary conditions.

```
READ(Lboundary,*) nbcV              ! Read number of boundary conditions
read(Lboundary,*) (kbc(i,k),k=1,3),(VBCo(i,k),k=1,2)
```

Number of the node is stored in kbc(i,1), the switches in kbc(i,2) and kbc(i,3) determine if the condition is applied or not (conf. with the structure of BOUNDARY file in Code_Input directory). VBCo() stores the values of the (Dirichlet) BC. These are the initial values, the actual ones are modified in controller accordingly and stored in VBC()

A.4.3 call read_control

The routine reads control parameters for the Navier-Stokes solve, should not be mixed with control of the flow!

A.4.4 call setIA

Prepares the matrices for the QMRPACK solver.

A.4.5 call read_restart_steady

Obvious action, if data not present simply continue.

A.4.6 call read_restart_unsteady

Obvious action - note that if present, unsteady values overwrite steady, read in the previous subroutine, if data not present simply continue.

A.4.7 7777 CONTINUE

Time loop

```
T = T + DT
```

A.4.8 call pressure

Interpolates pressure to all 6 nodes of the triangle.

A.4.9 call flow_controller

Main entrance to the controller. The Flow Solver uns3 in the file xAcuator.f (subroutine flow_controler) writes binary data about grid, fields, time into Transfer_Octave and sets Barrier in file Barrier_Octave (not MPI Barrier, just creates empty file) in xActuator.f Fortran code tests if Barrier is there and waits till it disappear. Octave or Matlab notice appearance of Barrier and reads flow data, process them (flow control, here only setting BC for rotating cylinder) and writes binary output - Dirichlet BC in KBC and (eventually) Volume Forces, all to Output_Octave file, removing the Barrier after writing. Flow solver realizes that Barrier is gone, reads actuation and continues flow computation.

A.4.10 call set_kick

Introduction of the disturbance to the flow to trigger unsteady flow motion (if needed, set by kickme .true. in Code_Input/CONTROL)

A.4.11 call update_unsteady

Update of the unsteady loop for next dt step.

A.4.12 call set_bcV

Introduction of Dirichlet BC to the F() vector

A.4.13 8888 CONTINUE

Internal iteration (Navier-Stokes) loop

A.4.14 call read_control

The routine reads control parameters for the Navier-Stokes solve, should not be mixed with control of the flow, the parameters are read on each iteration step and in this way modified in the running program.

A.4.15 call read_material

Reads Re eps Bv, here only Re is relevant and used.

A.4.16 call set_bcV

A.4.17 call assemble_matrix

A.4.18 call set_bc_RD

Set zero to the residuum in rows where Dirichlet BC are set - to be sure that small round-off error is not accumulated

A.4.19 call load

Not used presently - sets RHS

A.4.20 call dsys

Solution of linear equation set with QMRPACK

A.4.21 call update_internal

Update of internal Navier-Stokes (Newton-Raphson) loop within the time step.

A.4.22 call set_bc_RD

A.4.23 call set_bcV

A.4.24 call resid

Computes residuum to decide about further iteration.

A.4.25 call write_restart_steady

A.4.26 call V2

Preparation of visualisation and writing file in "pictures" directory.

A.4.27 call write_restart_unsteady

A.4.28 GO TO

Depending on residuum:

```
GO TO 7777
```

or

```
GO TO 8888
```

or

```
stop
```

B. The control script

The listing of the control script for the displayed transient in Chapter 3 reads:

```
%***** Initialize and clean ***********************************************
T=0;
system ('rm -f Control_Barrier');

%***** Loop over time *************************************************
%***** Here is some T time set as the limit, it can be endless loo too.
while T < 1000

%***** Wait until the flow solver writes the data **********************
%***** and writes the Control_Barrier file *****************************
%***** NOTE: MATLAB has no "UNTIL" loop!
    while(1)
        if (exist('Control_Barrier', 'file') == 2 )
            break;
        end;
    end;

%***** Allow output to stdout (presently only time T is written)
more off;

%***** Read Fortran written files - watch markers written by "SEQUENTIAL"
 fid=fclose('all');
 fid=fopen('Control_Input.dat','rb');
 fseek(fid, 4, 'cof');

T   =fread(fid, 1, 'real*8')
DT  =fread(fid, 1, 'real*8');
NDE =fread(fid, 1, 'int32');
NEL =fread(fid, 1, 'int32');
NDEmax=fread(fid, 1, 'int32');
NELmax=fread(fid, 1, 'int32');
III   =fread(fid, 6*NELmax, 'int32');
IX     =reshape(III,[6,NELmax]);
NOPP   =fread(fid,NDEmax,'int32');
XX     =fread(fid,NDEmax,'real*8');
YY     =fread(fid,NDEmax,'real*8');
VolXg  =fread(fid,NDEmax,'real*8');
VolYg  =fread(fid,NDEmax,'real*8');
Nvf    =fread(fid, 1, 'int32');
NBC    =fread(fid, 1, 'int32');
III    =fread(fid, NDEmax*4, 'int32');
KBC    =reshape(III,[NDEmax,4]);
FFF    =fread(fid,NDEmax*3,'real*8');
VBCo   =reshape(FFF,[NDEmax,3]);
FFF    =fread(fid,NDEmax*3,'real*8');
VBC    =reshape(FFF,[NDEmax,3]);
FFF    =fread(fid,NDEmax*2*5,'real*8');
F      =reshape(FFF,[NDEmax*2,5]);
PRESS  =fread(fid,NDEmax,'real*8');

```

```
%----- Comment for programmers ----------------------------------------
% Equivalent (binary=fast)  write statement in flow solver
% one can skip unnecessary information in both, Fortran xActuator.f and matlab/octave files
%
%          write(60)T, DT, NDE,NEL,NDEmax,NELmax, IX, NOPP,
%     &                      XX, YY, VolXg, VolYg, Nvf,
%     &                      NBC,KBC,VBCo, VBC,F,PRESS,isteady
%
% Here we have full information about flow and can perform control
% Note: F(NOPP(I),1),F(NOPP(I)+1,1),PRESS(I) are Vx, Vy and P in node "i"
%----------------------------------------------------------------------

%***** Control law ***************************************************
%***** Initialize the control law ...
U1 = 0.0; % ... for the cylinder at the front
U2 = 0.0; % ... for the cylinder at the back, bottom
U3 = 0.0; % ... for the cylinder at the back, top
%***** Kick oscillation
if T<6.25
  U1 = 0.5;
elseif T<12.5
  U1 =-0.5;
end;
%***** Start strong boat tailing after t=100
if T>=50
  U2 = +4;
  U3 = -4;
end

%***** Implement control law in boundary nodes ************************
if  T==0.1
% printf ("first step\n");

elseif T >=0.2
%***** First cylinder, front
%***** R=0.5, (x_0,y_0) = (-1.5 * sqrt(3./4.) , 0),
       for j = 379:522  % 96 nodes on a circle
          i=KBC(j,1) ;  % node number
          VBC(j,1) = -2 * U1 * (YY(i) - 0)             ; % V_x
          VBC(j,2) =  2 * U1 * (XX(i) + sqrt(27./16.)) ; % V_y
       end

%***** Second cylinder, bottom right
%***** R=0.5,  (x_0,y_0) = (0, -3/4),
       for j = 235:378 % 96 nodes on a circle
          i=KBC(j,1);   % node number
          VBC(j,1) = -2 * U2 * (YY(i) + 0.75) ; % V_x
          VBC(j,2) =  2 * U2 * (XX(i) - 0.00) ; % V_y
       end

%***** Third cylinder, top right
%***** R=0.5,  (x_3,y_3) = (0, 3/4)
       for j = 1:144 % 96 nodes on a circle
           i = KBC(j,1);
          VBC(j,1) = -2 * U3 * (YY(i) - 0.75) ; % V_x
          VBC(j,2) =  2 * U3 * (XX(i) - 0.00) ; % V_y
```

```
107         end
108 %***** End of "elseif T>=0.2"
109 end
110
111 %***** Write out actuation (Dirchlet BC + volume forces) to flow solver
112 %***** NOTE to programmers: Fortran MUST read it with access="STREAM" !!!
113
114 fid = fopen('Control_Output.dat','w');
115 fwrite(fid,VBC,'real*8');
116 fwrite(fid,VolXg,'real*8');
117 fwrite(fid,VolYg,'real*8');
118 fid= fclose('all');
119
120 %***** Remove Control_Barrier set by flow solver **********************
121 %***** This will start the next integration step **********************
122 system ('rm -f Control_Barrier');
123
124 %***** End of while T<1000 ****************************************
125 end
```