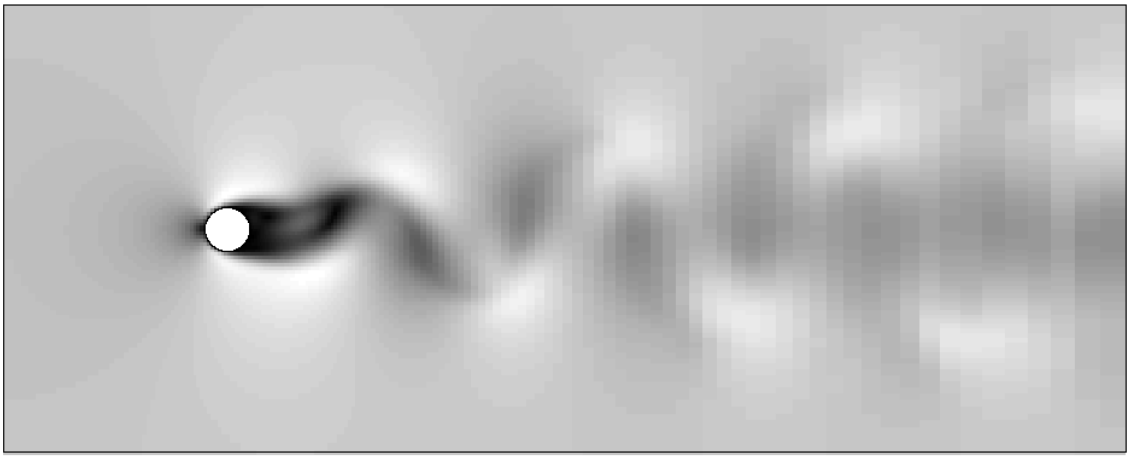
A numerical bifurcation analysis of flow around a circular cylinder

Maarten Duyff



Department of Mathematics

RuG

Master's Thesis

A numerical bifurcation analysis of flow around a circular cylinder

Maarten Duyff

University of Groningen
Department of Mathematics
P.O. Box 800
9700 AV Groningen

October 2006

Contents

Chapter 1

Introduction

Many physical phenomena can be described by partial differential equations (PDEs). Changing the physical parameter γ in these PDEs will cause changes in the behavior of the system. We are interested in changes of the stability of the state when the physical parameter γ is varied. (A state is stable if a small perturbation of the state disappears over time.) From these solutions we can make a *bifurcation diagram* where a characteristic quantity of the state is plotted as function of the physical parameter γ.
There are also ranges of values γ where the state is not stable. Close to the critical threshold γ_c of the physical parameter where the stability changes, systems become more sensitive, such that small perturbations may trigger some drastic changes in the state. The threshold γ_c is called a bifurcation point.
A nice example of a physical system that became unstable is the famous Tacoma Narrows suspension bridge. The bridge collapsed a few months after the opening. The forces of the wind on the bridge led to a transition between purely vertical oscillations and torsional behavior which lead to the collapse of the bridge. To prevent such situations, engineers can perform a bifurcation analysis on such systems.
A bifurcation analysis of a system can sometimes be done in a laboratory where a scaled model is build of the system. The physical parameter γ can be varied and one can see how small changes of the parameters effect the state of the system. This kind of analysis is often very expensive. Nowadays it is also possible to do the bifurcation analysis numerically by simulating a model with the computer. This is much cheaper and it is much easier to change some physical parameters.
A numerical bifurcation analysis can be done in two ways: by repeated simulations or by continuation. In the repeated simulation approach, we run the simulation code for different values of γ and look at the behavior of the system. The second way is to use a continuation technique which will give accurate results on the location of the bifurcation point γ_c.

1.1 The project

In this project we perform a numerical bifurcation analysis of a flow problem: the Navier-Stokes flow of a fluid around a circular cylinder. There is already a lot of literature available about this flow problem. At a low Reynolds number the flow will be time independent, a "steady state". When the Reynolds number is increased a Hopf bifurcation occurs, and the

flow becomes periodic. At higher Reynolds numbers, the well known "Von Karman vortex street" appears. Both states are fully two dimensional. The flow problem is explained in Section 1.2.
For the numerical bifurcation analysis we use PDEcont, a software library for time stepper-based bifurcation analysis of large scale systems, created by K. Lust [1]. For the simulation we use a three dimensional direct numerical simulation code (DNS) from the mathematical department of *Rijksuniversiteit Groningen* [2]. We use this code in a two dimensional mode. Both software codes are explained in some detail in Chapter 3 and in Section 1.3 we give a short summary of available software for numerical bifurcation analysis.
The goals of this project are performing a numerical bifurcation analysis of the flow problem and detecting possible weaknesses and difficulties in the combination of PDEcont and the DNS code. We hope that with the combination of PDEcont and DNS we can obtain results which are in good agreement with the results from literature summarized in Section 2.3.
After the introduction of the flow problem and the available software we explain the mathematical theories in Chapter 2. In Chapter 3 we explain the software used in detail and what we have to do for performing a bifurcation analysis on our flow problem. In Chapter 4 we explain how we did it. The results are given in Chapter 5 and in the end we have a final discussion in Chapter 6.

1.2 The flow problem

Let us now look at our flow problem. Consider an infinitely long circular cylinder placed perpendicular to a uniform open flow. When the fluid starts flowing, a wake appears behind the cylinder. In this project we only study the two dimensional behavior of the cylinder wake. Therefore it is sufficient to use the two dimensional Navier-Stokes equations for our flow problem. The incompressible *Navier-Stokes* equations describe the flow behavior in our problem. Despite the fact that in our flow problem there are no walls and the cylinder is infinitely long, we have to choose a calculation domain D large enough with proper boundary conditions. It is very important that the walls have no influence on the behavior of the wake. The two-dimensional Navier-Stokes equations are written as follows:

$$\begin{aligned} \nabla \cdot \boldsymbol{u} &= 0 \\ \frac{\partial \boldsymbol{u}}{\partial t} &= -(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \nabla p + \frac{1}{Re}\nabla^2 \boldsymbol{u}. \end{aligned} \tag{1.1}$$

The vector $\boldsymbol{u} = (u(x, y, t), v(x, y, t))$ denotes the velocity with components u and v in the x and y directions. Pressure is denoted by $p(x, y, t)$ and the time by t.
The domain D with C the cylinder is defined as

$$D := \{(x, y) | -a \leq x \leq -a, -a \leq y \leq b\} \backslash C(0, r).$$

The outflow boundary conditions are chosen such that there exists no stress at the outflow. In our project the following boundary conditions are used. On the cylinder surface a no-slip condition is applied. At the left boundary (the inflow), a Dirichlet condition ($u = 0, v = 1$) is applied. On the upper and lower boundaries $x = \pm a$ homogeneous Neumann conditions

$$\frac{\partial u}{\partial n} = 0, \tag{1.2}$$

$$\frac{\partial v}{\partial n} = 0 \tag{1.3}$$

are used. In this way we try to simulate a domain without these upper and lower walls. At the outflow boundary $y = b$ we use

$$\nu \frac{\partial u}{\partial y}(x, b) = 0 \tag{1.4}$$

$$\frac{\partial v}{\partial y}(x, b) = 0 \tag{1.5}$$

$$p(x, b) = 0. \tag{1.6}$$

The first condition (1.4), where ν denotes the kinematic viscosity, means physically that there is no viscous normal stress. This is physically a good choice, but mathematically there is not enough information to obtain a unique solution. For mathematical reasons an extra condition (1.5) is added. The pressure on the right boundary is set to zero (1.6). Due to these outflow conditions the flow has no stress leaving the domain. The boundary conditions are summarized in Fig. 1.1.

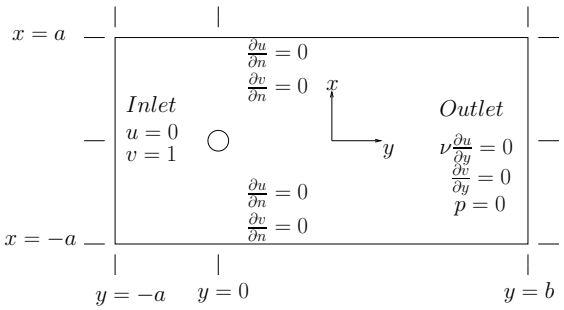


Figure 1.1: *Schematic view of the problem on the domain D with border Γ*

The *Reynolds number* is the most important parameter in this system. The Reynolds number is defined as

$$Re := U_\infty d/\nu \tag{1.7}$$

where U_∞ denotes the flow velocity far from the cylinder, d is the cylinder-diameter and ν is the kinematic viscosity. In our case the $U_\infty = 1$ at the inflow boundary. When this Reynolds number is increasing the behavior of the flow, affected by a disturbance, in the cylinder wake changes. In Fig. 1.2 (taken from [3]) it is illustrated how the flow changes. At a very low Reynolds number the flow is laminar, steady and does not separate from the cylinder (a). When $6 < Re < 46$ the flow separates from the cylinder but it is still steady and laminar (b). At $Re > 46$ the flow becomes time-periodic with the wake oscillating in space (c) and (d). The flow in figure (d) is called the "Von Karman vortex street". This behavior is still two dimensional with no velocity in the spanwise direction z. After some bifurcations the flow becomes turbulent (e) and the behavior is fully three dimensional. The Reynolds number Re is the natural branching parameter in this bifurcation problem.

The critical value of $Re \approx 46$ marks the transition from stationary solutions to time-periodic solutions as shown in Fig. 1.2(c)(d). This transition is called a supercritical *Hopf bifurcation*, because stable stationary solutions become unstable stationary solutions at this critical value

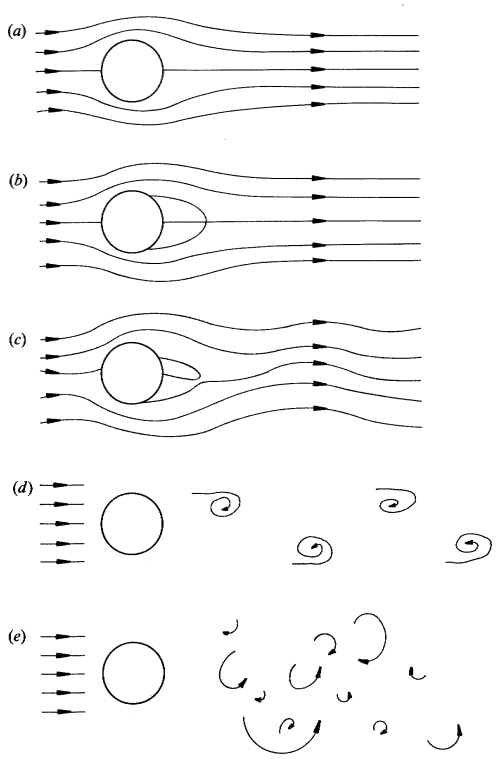


Figure 1.2: *Changes in the flow dynamics with increasing Reynolds number. Source:[3]*

of Re and stable periodic solutions arise for $Re > 46$. Changes in the periodic wake dynamics as a function of Re are quantified using shedding frequency measures. One often uses the non dimensional vortex shedding frequency or *Strouhal number*, defined as

$$St := fd/U_\infty. \tag{1.8}$$

For a range of Re above the first bifurcation point $Re \approx 46$ the wake is laminar and perfectly time-periodic. According to [4], the flow becomes periodic in the spanwise direction in the second bifurcation point at $Re \approx 189$. These two bifurcations are the most important ones, because they are visible in real life. For higher values of Reynolds, many more bifurcations will turn up and eventually the wake becomes chaotic, see Fig. 1.2 (e).

1.3 Software for numerical bifurcation analysis

There are many software packages available for numerical bifurcation analysis. Examples of software packages are AUTO [5], Locbif [6] and CONTENT [7]. These packages can analyze the stability of steady states and periodic solutions of autonomous ODEs. They are optimized for low-dimensional systems of ODEs. The calculation and memory cost are increasing very fast when the dimension of the problem increases. Moreover these packages are not optimized for the sparsity of the matrices which occur in the calculations.
Though a PDE can be transformed to a large system of ODEs, the use of these standard

packages often becomes prohibitively expensive. There are just a few software packages available for continuation and bifurcation analysis of solutions of PDEs. A well known package is PLTMG [8]. But this package is not capable of analyzing the stability of the calculated solution and can not detect some types of bifurcation points like the Hopf bifurcation. K.Lust developed a continuation software package PDEcont [1]. This is a time stepper-based continuation code for large systems. PDEcont can analyze the stability of steady-states as well as periodic solutions.

Chapter 2

The theory

Now we need some mathematical theory about stability. We have to investigate the stability of isolated steady states and isolated periodic solutions. In this project one of the goals is to do a stability analysis over a range of Reynolds numbers and try to find the first instability at $Re \approx 46$. In this chapter we give theory to determine the stability information corresponding to the solutions and a technique to compute solutions in a range of Reynolds numbers. With the stability information we can locate bifurcation points like the Hopf bifurcation.

2.1 Linear stability

Many physical models can be written as a system of autonomous ordinary differential equations (ODEs)

$$\frac{dy}{dt} = f(y, \gamma) \tag{2.1}$$

where $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$. $\gamma \in \mathbb{R}$ is a parameter. Let y^* be a steady state or a periodic solution ($y^*(t)$) of (2.1). Assume we apply a small perturbation u'_0 at $t = t_0$. To investigate the evolution of the perturbation in time, we substitute the perturbed solution $y^*(t) + u'(t)$ in (2.1). Using a first order Taylor expansion of $f(y^* + u', \gamma)$, we obtain the equation

$$\frac{du'(t)}{dt} = f_y(y^*(t); \gamma)u'(t), \qquad u'(0) = u'_0 \tag{2.2}$$

for the linear evolution of the perturbation $u'(t)$ in time. The matrix $f_y(y^*(t); \gamma) \in \mathbb{R}^{n \times n}$ operates on the perturbation $u'(t)$ which means this matrix is important for determining whether a solution is stable or not. The solution $u'(t)$ of (2.2) is

$$u'(t) = e^{\int_0^t f_y(y^*(\tau);\gamma)d\tau} u'_0. \tag{2.3}$$

In our flow problem we have to deal with two types of solutions, namely the steady state y^* and the periodic solution $y^*(t)$. The meaning of these solutions y^* and $y^*(t)$ is illustrated in Fig. 2.1. When a perturbation u'_0 to y^* converges to zero in time, y^* is called asymptotically stable. In the right figure there is also an unstable steady solution, marked with $\circ$. In our flow problem described in Section 1.2, we saw that at $Re \approx 46$ a transition takes place from a steady state to a time periodic state. The figure on the left illustrates the situation for $Re < 46$ and on the right the situation for $Re > 46$.

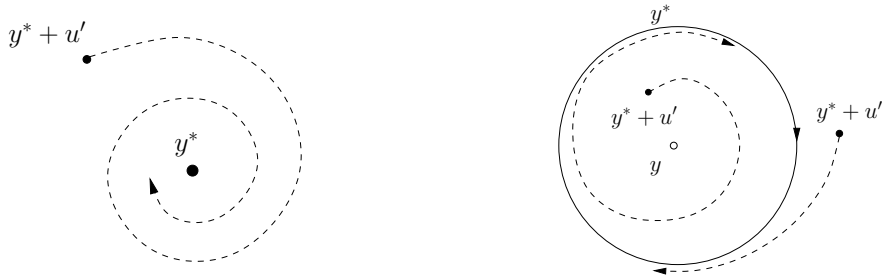


Figure 2.1: *Right: An isolated steady state y^* at $Re < 46$. Left: An isolated periodic state $y^*(t)$ at $Re > 46$.*

2.1.1 Steady-state solutions

When y^* is a steady-state solution, i.e., independent of time, the Jacobian f_y in (2.3) is also independent of time. Define $A \equiv f_y$ as the Jacobian of f. The Jacobian is evaluated at the solution y^*, given parameter γ. Then

$$u'(t) = e^{At}u'_0 \tag{2.4}$$

is the solution of (2.2). Here the eigenvalues λ of A give us information how $u'(t)$ evolves in time. With the eigenvalues of A we can determine whether the $u'(t)$ converges to zero in time or not. The following theorem gives the conditions on the eigenvalues for asymptotic stability.

Theorem 2.1.1 *Suppose $f(y)$ is two times continuously differentiable and $f(y^*;\gamma) = 0$. The real parts of the eigenvalues λ_i for $i = 1, \ldots, N$ of the Jacobian A evaluated at the stationary solution y^* determine stability in the following way:*
(a) $\Re(\lambda_i) < 0$ for all i implies asymptotic stability, and
(b) $\Re(\lambda_k) > 0$ for one or more values of k implies instability.

2.1.2 Periodic solutions

We are also interested in the stability of time periodic solutions $y^*(t)$. If the solution $y^*(t)$ is a periodic orbit with period T, then the following holds:

$$\varphi(y^*(t), T; \gamma) = y^*(t) \quad \forall t \tag{2.5}$$

where $\varphi(y, T; \gamma)$ is defined as the result of integrating (2.1) over a time interval of length T starting at y. Because the solution $y^*(t)$ is periodic in time t, the Jacobian f_y in (2.2) is not constant. We are interested in what happens with the perturbation $u'(t)$ after one period. Then from (2.3) follows

$$u'(T) = Mu'_0, \tag{2.6}$$

the perturbation at time T, where $M \in \mathbb{R}^{n\times n}$ is the monodromy matrix

$$M(y^*(T);\gamma) = e^{\int_0^T f_y(y^*(t);\gamma)dt}. \tag{2.7}$$

The eigenvalues of M give us the information about the evolution of u'.

The monodromy matrix can also be derived differently. Say $\varphi(y_0, \tau; \gamma)$ is a solution of (2.1). Thus

$$\frac{\partial \varphi(y_0, \tau; \gamma)}{\partial \tau} = f(\varphi(y_0, \tau; \gamma), \gamma). \tag{2.8}$$

Differentiating this with respect to y_0 gives us the matrix differential equation

$$\frac{\partial}{\partial \tau}\frac{\partial \varphi}{\partial y_0}\bigg|_{(y_0,\tau,\gamma)} = \frac{\partial f}{\partial y}\bigg|_{(\varphi(y_0,\tau,\gamma),\gamma)} \frac{\partial \varphi}{\partial y_0}\bigg|_{(y_0,\tau,\gamma)} \tag{2.9}$$

with initial condition

$$\frac{\partial \varphi}{\partial y_0}\bigg|_{(y_0,0,\gamma)} = I. \tag{2.10}$$

The solution at time T of the differential equation (2.9) is

$$\frac{\partial \varphi}{\partial y_0}\bigg|_{(y_0,T,\gamma)} = e^{\int_0^T f_y(\varphi(y_0,\tau;\gamma),\gamma)d\tau} = M(y_0, T, \gamma) \tag{2.11}$$

where M is again the monodromy matrix. Here M is written as the derivative of φ with respect to the state vector y. The eigenvalues μ of M are called Floquet multipliers. μ is defined as

$$\mu = e^{\sigma T}, \tag{2.12}$$

where σ is called the Floquet exponent and plays the same role as the eigenvalues λ of A. The eigenvectors of M are called Floquet modes. We give a theorem about the properties of the Floquet multipliers for stability.

Theorem 2.1.2 *Let $(y^*, T; \gamma)$ determine a periodic solution of (2.1). Let M be the monodromy matrix for this orbit. Let $\mu_1 = 1, \mu_2, \ldots, \mu_N$ be the N eigenvalues of M.*
Then:
(a) The periodic orbit determined by $(y^*, T; \gamma)$ is asymptotically stable if $|\mu_i| < 1$, $i = 2, \ldots, N$.
(b) The periodic orbit determined by $(y^*, T; \gamma)$ is unstable if $|\mu_i| > 1$ for at least one value of i.

A bifurcation occurs if a Floquet multiplier crosses the unit circle. There are three possible scenarios: Floquet multiplier $\mu_i = 1$, $\mu_i = -1$ and $\mu_i = e^{\pm i\sigma}$ [1].

If a solution y^* at a parameter value γ from system (2.1) is stable and γ is close to γ_c, then a small perturbation in the parameter γ can lead to another solution y. In our flow problem, when we are close to $Re = 46$ a small perturbation can lead to a transition to a periodic solution $y^*(t)$. Two complex conjugate eigenvalues cross the imaginairy axis and the real part becomes positive. A Hopf bifurcation occurs.
At the Hopf bifurcation the two complex conjugate eigenvalues are $\lambda = \pm\beta i$, where β denotes the initial angular velocity of the periodic solution. Thus $\beta = 2\pi f$ with f the frequency.

From the definition of the Floquet multiplier (2.12) the complex conjugate eigenvalues λ are mapped to 1 as

$$e^{\pm\beta i T} = e^{\pm 2\pi i T^{-1} T} = e^{\pm 2\pi i} = 1. \tag{2.13}$$

This means that two Floquet multipliers equal to one marks the beginning of the periodic regime.

Definition 2.1.1 *A bifurcation from a branch of equilibria to a branch of periodic oscillations is called a Hopf bifurcation.*

This happens in our system at $Re \approx 46$. At $Re \approx 189$ there is a bifurcation to a three dimensional state. Unfortunately we can not find this instability, because we use the DNS code in a two dimensional mode.

2.2 Continuation

Now we know how to calculate the stability of solutions y^* and are able to locate bifurcations by computing the dominant eigenvalue of Floquet multiplier. We have to determine the rightmost eigenvalues of A or the largest in modulus eigenvalues of M to see whether a solution is stable or not. Of course we are interested in the whole branch of stable solutions and the values of our branching parameter γ where stability is lost.
With the technique of continuation we can calculate a branch of stable and unstable solutions. The continuation process contains two steps. First a predictor step, where a new point on the branch is predicted and second the corrector step where the predicted point is refined to a new point on the branch.

2.2.1 The predictor

The predictor predicts the next point on the branch based on already known points. PDEcont predicts new points by means of polynomial extrapolation using already known points (y, γ), where a point consists of a state y at a corresponding value of γ. In PDEcont a linear extrapolation method is used, called the secant predictor. In Fig. 2.2 the main idea is shown. First the search direction is computed based on known points. Then a parameter step is taken in that direction.
However, a good continuation code does not parameterize the state vector y with the branching parameter γ. When the state vector is parameterized by γ, the continuation code get problems when the tangent to the curve $y(\gamma)$ becomes vertical, i.e. a turning point occurs.
So, good continuation codes use a new parameter η such that $(y(\eta), \gamma(\eta))$. The secant method needs the current point (y_i^*, γ_i^*) and the previous point $(y_{i-1}^*, \gamma_{i-1}^*)$ to compute a search direction an takes a parameter step $\Delta\eta$. After the prediction, the estimate $(y_{\hat{i+1}}, \gamma_{\hat{i+1}})$ of the next solution must be refined to $(y_{i+1}^*, \gamma_{i+1}^*)$. (see Fig. 2.2) This is done by the corrector.

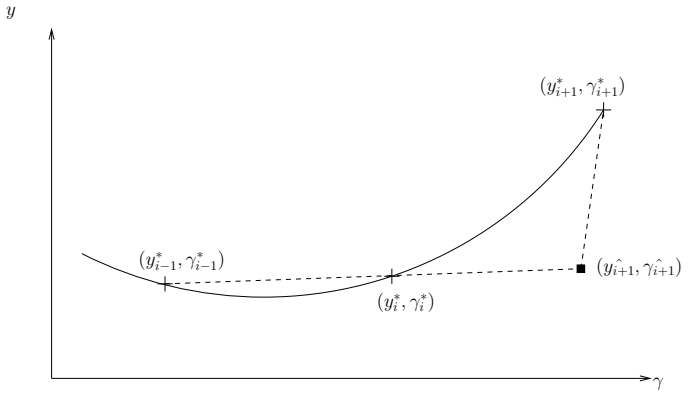


Figure 2.2: *The secant method.*

2.2.2 The corrector

The corrector refines the estimate to the solution y^* on the branch. To calculate a new point on the branch of steady states we have to solve the system

$$\begin{cases} f(y,\gamma) &= 0 \\ n(y,\gamma;\eta) &= 0 \end{cases} \tag{2.14}$$

where $n = 0$ is the parameterizing equation since we want to be able to pass a turning point. This system determines the two unknowns (y, γ).
In the case of periodic solutions $y^*(t)$ has to satisfy to the system

$$\begin{cases} r(y,T,\gamma) &= \varphi(y,T,\gamma) - y &= 0 \\ s(y,T,\gamma) &= 0 \\ n(y,T,\gamma;\eta) &= 0 \end{cases} \tag{2.15}$$

where equation s is the phase condition and n is again the parameterizing equation. The equation s is needed to fix an initial point on the orbit. The equation n must be chosen such that no singularities occur in the parameter η. Examples of possible parameterizing equations are

$$\begin{aligned} n_1(y,T,\gamma;\eta) = \theta_x||y(t_0;\eta) - y^*(t_0;\eta_0)||_2^2 + \theta_T(T(\eta) - T^*(\eta_0))^2 + \\ \theta_\gamma(\gamma(\eta) - \gamma^*(\eta_0))^2 - (\eta - \eta_0)^2 = 0, \end{aligned} \tag{2.16}$$

where $(y^*(t_0;\eta_0), T^*(\eta_0), \gamma^*(\eta_0))$ is a known periodic solution on the branch. An other example is

$$n_2(y,T,\gamma;\eta) = \theta_y(y - y_p^*(t_0))^T y_s^*(t_0) + \theta_T(T - T_p^*)T_s^* + \theta_\gamma(\gamma - \gamma_p^*)\gamma_s^* = 0, \tag{2.17}$$

where $(y_p^*(t_0), T_p^*, \gamma_p^*)$ is the point generated by the predictor and $(y_s^*(t_0), T_s^*, \gamma_s^*)$ is the normalized predictor direction. PDEcont uses n_2 as the parameterizing equation.
This system determines the three unknowns (y, T, γ). Both systems can be solved with the Newton-Raphson method. In the periodic case we solve

$$J \begin{bmatrix} \Delta y \\ \Delta T \\ \Delta\gamma \end{bmatrix} = - \begin{bmatrix} r(y^{(\nu)}, T^{(\nu)}, \gamma^{(\nu)}) \\ s(y^{(\nu)}, T^{(\nu)}, \gamma^{(\nu)}) \\ n(y^{(\nu)}, T^{(\nu)}, \gamma^{(\nu)}; \eta) \end{bmatrix} \tag{2.18}$$

where J is the Jacobian matrix

$$J = \begin{bmatrix} \frac{\partial r(y^{(\nu)},T^{(\nu)},\gamma^{(\nu)})}{\partial y} & \frac{\partial r(y^{(\nu)},T^{(\nu)},\gamma^{(\nu)})}{\partial T} & \frac{\partial r(y^{(\nu)},T^{(\nu)},\gamma^{(\nu)})}{\partial \gamma} \\ \frac{\partial s(y^{(\nu)},T^{(\nu)},\gamma^{(\nu)})}{\partial y} & \frac{\partial s(y^{(\nu)},T^{(\nu)},\gamma^{(\nu)})}{\partial T} & \frac{\partial s(y^{(\nu)},T^{(\nu)},\gamma^{(\nu)})}{\partial \gamma} \\ \frac{\partial n(y^{(\nu)},T^{(\nu)},\gamma^{(\nu)})}{\partial y} & \frac{\partial n(y^{(\nu)},T^{(\nu)},\gamma^{(\nu)})}{\partial T} & \frac{\partial n(y^{(\nu)},T^{(\nu)},\gamma^{(\nu)})}{\partial \gamma} \end{bmatrix}, \tag{2.19}$$

followed by the update

$$y^{(\nu+1)} = y^{(\nu)} + \Delta y, \tag{2.20}$$
$$T^{(\nu+1)} = T^{(\nu)} + \Delta T, \tag{2.21}$$
$$\gamma^{(\nu+1)} = \gamma^{(\nu)} + \Delta \gamma, \tag{2.22}$$

where we use $(y^{(0)}, T^{(0)}, \gamma^{(0)})$ from the predictor as starting point. In the Jacobian matrix (2.19) we see derivatives of $r(y, T, \gamma) = \varphi(y, T, \gamma) - y$ with respect to y, T and γ. In the first derivative,

$$\frac{\partial r(y,T,\gamma)}{\partial y} = \frac{\partial \varphi(y,T,\gamma)}{\partial y} - I = M - I \tag{2.23}$$

we recognize the monodromy matrix M (2.11) explained in the previous section. This matrix is also needed for the determination of the stability.

2.3 The stability results from literature

There is already a lot of literature about this particular stability problem. Therefore two articles, which are referenced in many studies about this flow problem, are used for the main idea. The first instability at $Re \approx 46$ was studied by Jackson in [3]. This is a bifurcation from a two dimensional to a two dimensional state and hence it can be analyzed in a two dimensional domain. The second instability, at $Re \approx 189$, is a bifurcation to a three dimensional state. This bifurcation was studied by Barkley [9]. We will not calculate this instability in this project, but we can see how Barkley used a flow solver to find this instability.

2.3.1 First instability by Jackson

The first instability is found by Jackson [3]. At $Re < 46$ the solution is a stable steady-state. At $Re \approx 46$ a Hopf bifurcation occurs and the flow becomes periodic in time. The frequency of the periodic solution is quantified by the Strouhal number (1.8).
After space discretisation of the Navier-Stokes equations (1.1) the system can be rewritten as

$$B\frac{dy}{dt} = f(y, Re), \tag{2.24}$$

where $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$. B is a singular matrix, because the continuity equation is time independent. Jackson was interested in the value of Reynolds where the Hopf bifurcation occurs. He calculated the solutions y^* and the the corresponding eigenvalues for the stability analysis. He was interested in the equilibria y^* which are solutions of

$$f(y; Re) = 0. \tag{2.25}$$

At the Hopf bifurcation point a pair of complex conjugate eigenvalues crosses the imaginary axis. Jackson used *extended-system techniques* which give accurate results for the critical Reynolds and Strouhal number. He used the following extended system:

$$\begin{cases} f(y;Re) &= 0, \\ f_y\xi_r + \omega B\xi_i &= 0, \\ f_y\xi_i - \omega B\xi_r &= 0, \\ (e^k)^T \cdot \xi_r &= 0, \\ (e^k)^T \cdot \xi_i &= 1, \end{cases}$$

where the subscripts r and i denote the *real* and *imaginary* part of the eigenvector ξ. This extended system determines a Hopf bifurcation point, the real and imaginary parts of the bifurcating eigenvector and the corresponding angular frequency ω and Reynolds number Re. The complex eigenvalues can be traced with the continuation technique.
This approach is much cheaper than time-dependent calculations for determining these quantities. However, this technique does not give directly the time-dependent behavior at values greater than the critical Reynolds number $Re \approx 46$.

2.3.2 Second instability by Barkley

The second instability at $Re \approx 189$, which corresponds with three-dimensional flow, has been found by Barkley and Henderson [9]. Because three-dimensional simulations are very expensive, a two dimensional flow is calculated. With an artificial periodic flow in the spanwise direction z the second instability can be found. A two dimensional periodic solution $y^*(t)$ is obtained with direct numerical simulation from (1.1). The techniques Barkley used to calculate the periodic solutions are discussed in detail in Henderson and Karniadakis [10]. For the continuation of the solution at different Reynolds numbers Barkley used a shooting technique [1]. For the stability analysis a three-dimensional perturbation $\boldsymbol{u'}(x,y,z,t)$ is added to the periodic solution $y^*(t)$. The evolution of the perturbation $u'(t)$ is described by a linearized Navier-Stokes equation.
The system is homogeneous in the spanwise direction z and the perturbation can be decomposed in space by a Fourier transform

$$\boldsymbol{u'}(x,y,z,t) = \int_{-\infty}^{\infty} \boldsymbol{u'}_\beta(x,y,t)e^{i\beta z}d\beta. \tag{2.26}$$

The decomposition of the perturbed pressure p' is similar. In the linearized equation the modes with different wave numbers β do not couple. Now take for example a perturbation

$$\begin{aligned} \boldsymbol{u'}(x,y,z,t) &= (u'_\beta(x,y,t)\cos\beta z, v'_\beta(x,y,t)\cos\beta z, w'_\beta(x,y,t)\sin\beta z), \\ p'(x,y,z,t) &= p'_\beta(x,y,t)\cos\beta z. \end{aligned} \tag{2.27}$$

This perturbation will remain in this form under $M(y^*;Re)$ and hence every wave number β can be considered separately. After substituting the chosen perturbations in equation (2.2) the unknown components $(u'_\beta, v'_\beta, w'_\beta)$ and pressure p'_β from (2.27) can be computed. These unknowns are functions of x, y and t. What follows is an extra parameter β, the wave number. The main objective of Barkley is to determine the precise range of Re and β for which a perturbation of the periodic solution grows. Via Floquet analysis the stability of the time-dependent periodic solution $y^*(t)$ is determined. The method used to find the Floquet

Re_c	St	Author	Comment
45.403	0.13626	Jackson	Coarse grid
46.136	0.13793	Jackson	Fine grid
46.184	0.13804	Jackson	Richardson extrapolation

Table 2.1: *Results of the critical Reynolds number depending on the grid size. Source:[3]*

Re	St	Author	Comment
50	0.14	Gresho	Time dependent calculations
100	0.18	Gresho	Time dependent calculations
100	0.16	Braza	Time dependent calculations

Table 2.2: *Results of Strouhal number depending on the Reynolds number. Source:[3]*

modes corresponding to the multipliers near the unit circle is described by Schatz, Barkley and Swinney in [11]. In [12], Barkley and Henderson discuss the second instability in more detail.

2.3.3 Numerical results

There are many ways to discretise the equations (1.1). Finite difference, finite element or finite volume methods have all been used. The most popular method used for this bifurcation problem is the *spectral element method*. This method is used in [9, 12, 13] and [10] to obtain time-periodic solutions.
Barkley used for the integration a high-order time-splitting scheme. A more detailed discussion about how Barkley solves the Navier-Stokes equations is given in Henderson and Karniadakis [10].
Other methods have also been used to study the instabilities. Zhang *et al.* [14] used a finite-difference method for solving the Navier-Stokes equations. Jackson [3] and Noack *et al.* [15] used a two-dimensional finite-element Galerkin method.
Using the methods described above, the bifurcation points can be found. First, some results from Jackson are summarized in Table 2.1. Table 2.2 shows some other results. Table 2.2 shows that the results from Braza are in good agreement with the results from Fig. 2.4 and the results from Gresho are far above these results. We present our results in Chapter 5.

Next we discuss the numerical results of the second instability found by Barkley. Fig. 2.3 shows graphs of the dependence of the non-trivial dominant eigenvalue μ on the additional parameter β for different Reynolds numbers. At $Re = 140$ the graph is monotonically decreasing, which means that for any β the flow will return to its stable periodic state. At $Re = 170$ there turns up a local maximum, but the dominant eigenvalue stays in the unit circle and the periodic flow is still stable. At $Re \approx 188.5$ the maximum of the dominant eigenvalue becomes 1 and a bifurcation occurs. At $Re = 220$ there is a range of values of β for which the periodic flow is unstable. Note that for $\beta = 0$ there is always an eigenvalue $\mu = 1$ independent of the branching parameter Re. For detailed information, see [16]. For finding the critical values Re_2

Re_c	β_c	Author	Comment
188.5	1.585	Barkley	Spectral element
170	1.75	König, Noack and Eckelmann	Galerkin

Table 2.3: *Results with different methods.*

and β, stability computations are performed at $Re = 187$ and $Re = 190$ for $\beta = 1.4, 1.5, 1.6$ and 1.7. With two-dimensional fit the critical values are $Re = 188.6$ and $\beta = 1.584$. At this value the dominant eigenvalue or Floquet multiplier leaves the unit circle and the flow becomes fully three-dimensional. Table 2.3 shows the two result from Barkley, Noack, König

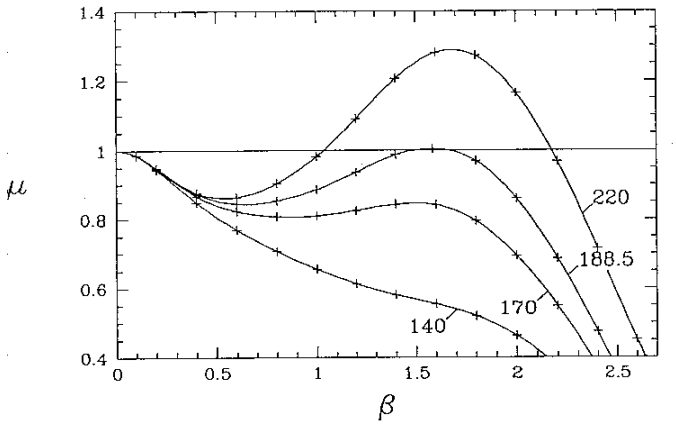


Figure 2.3: *Parameter dependence of the dominant Floquet multiplier on the spanwise wave number β* Source:[9]

and Eckelmann [15]. König, Noack and Eckelmann used in their Galerkin projection method a much lower number of grid elements than Barkley. The experimental results for the second instability were very inaccurate, so they were satisfied with their numerical results at that time.

To find an accurate result for the second instability in the third dimension, the mesh size of the grid has influence on the critical Reynolds number. The mesh dependence of the periodic solutions and the stability calculation, is discussed in [9]. In [4] different sub-domains of the solution are used to analyze the Floquet multipliers.

In Fig.2.4 the main results are shown. The two important critical values $Re_1 \approx 46$ and $Re_2 \approx 188.5$ and the branch of stable periodic solutions are depicted.
To verify the numerical results, experimental results are needed. It is very difficult to find the two important instabilities in experiments. What can be found is the branch of periodic solutions, or equivalently, the relation between the Strouhal number and the Reynolds number. A lot of experiments have been done by Mathis, Provensal and Boyer [17]. They did experiments with Reynolds numbers ranging from 40 to 300. The first instability is found

between Reynolds numbers ranging from 47 to 123. The difference are due to the distance from the cylinder to the walls and the diameter of the cylinder. Cylinder vibrations, wall conditions and end effects also play a roll in this inaccuracy. With the experimental results and the Stuart-Landau Law [17], they showed that the first instability is at $Re \approx 47$. Also resulting from the experiments is a continuous dependence of the Reynolds number and the Strouhal number starting from approximately $Re \approx 47$. The experimental curve is pretty close to the value computed by Jackson [3].
In the experiments at values $Re > 47$ the vortex shedding becomes oblique. Hammache and Gharib [18] produced very good approximations of the flow past an infinitely long cylinder in the laboratory with Re ranging from 47 to 160. They manipulated the end conditions in their experiments so that the shedding becomes parallel. These results are in very good agreement with the universal Strouhal curve for the low Reynolds number regime of Williamson [19]. This universal curve is obtained from measurements for parallel shedding in different facilities. Williamson [20] investigated why in experiments an oblique shedding turns up and also transformed the Strouhal data by a transformation using the oblique angles. He showed that the Strouhal curve has a discontinuity at $Re \approx 180$. His experimental results are a little bit below the numerical results. The Reynolds Strouhal relation is shown in Fig. 2.4.

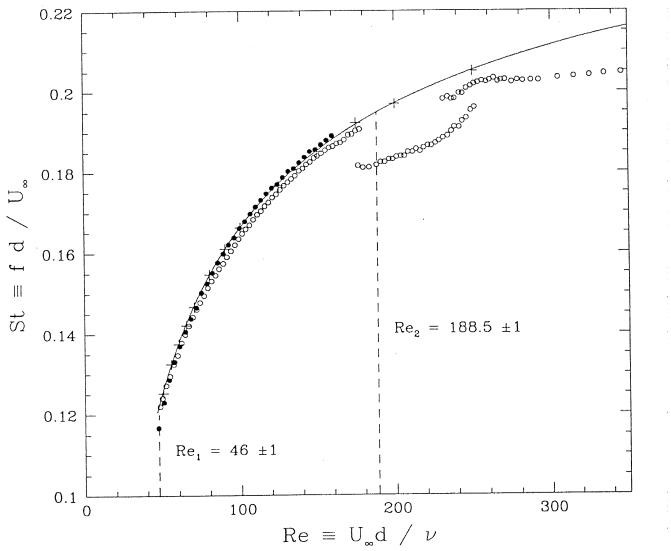


Figure 2.4: *Branch of stable periodic solutions with the two instabilities Re_1 and Re_2 . The variation of the Strouhal number with Reynolds number from Williamson [19], called 'universal curve' (solid line), experiments of oblique shedding from Williamson(1989) ∘, Hammache and Gharib (1991)•* Source:[9]

Despite the fact that at Re_2 and β_2 the wake becomes unstable in the third-dimension, the flow remains stable to two dimensional perturbations. This follows from the Floquet multipliers for $\beta = 0$, meaning no perturbation in the spanwise direction. Except for the neutral eigenvalue due to the invariance under time-translation, the largest eigenvalue is $\mu = 0.52$

which means that the perturbation is decreasing by factor of about two per period. This eigenvalue stays in the unit-circle until at least $Re = 250$, the end of Barkleys computations which means that the two-dimensional flow is stable under two dimensional perturbations for $Re \leq 250$. Noack, König and Eckelmann [15] confirmed that all the eigenvalues which are associated with the two-dimensional Floquet modes stay in the unit circle for $Re \approx 50$ up to 500.

Chapter 3

Numerical methods

We already introduced the codes used in this project in Section 1.3. Now we explain how we reached our first goal, the numerical bifurcation analysis of our flow problem. We use DNS for time integration and PDEcont for calculating the branches of solutions y^* depending of Re. In this chapter we describe both codes in some more detail. We want to know how these codes work, because these codes have to work together. Therefore we need information about the in- and output of both programs. There are different ways to couple both codes. We decided to couple the codes by exchanging data by files. This is probably the easiest way to do this. We come back to this topic later in Chapter 4.
We also describe some numerical tools the codes use and how the computations are done. We know that there are two methods for doing a numerical bifurcation analysis: repeated simulations and continuation. Both ways are done in this project. We will see that the first method is not very efficient. For the first method only the DNS code is used to simulate the flow from a certain initial condition $y_0 = y(t_0)$ in time. In this way we can look how the behavior of the flow changes when the physical parameter Re is changing. The results are presented in Section 5.1.
For the second method we use PDEcont in combination with DNS. In Chapter 4 we show how this is realized. The results are presented in Chapter 5.

3.1 DNS

3.1.1 What does DNS do?

On a predefined domain DNS gives us $\varphi(y_0, T; Re)$, the time integration over the time interval T starting from the initial state y_0 at a given Reynolds number Re. From the introduction we already know that we have two regimes in our flow problem: A steady state regime for $Re \lesssim 46$ and a periodic regime for $Re \gtrsim 46$ (see Fig. 2.1). In Fig. 3.1 we show some results from DNS started with values of Re in a different regime.

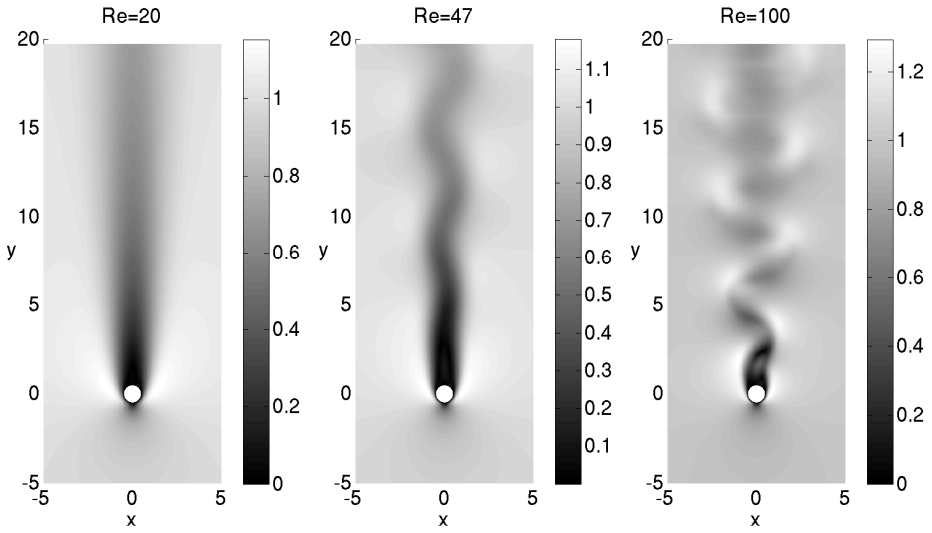


Figure 3.1: *Velocity fields of* $Re = 20$*,* $Re = 47$ *and* $Re = 100$*. Plots of the Euclidean norm* $\sqrt{u(x_i, y_j)^2 + v(x_i, y_j)^2}$

If we chose y_0 close to y^*, the state converges to y^* in time. We can clearly see the difference in the type of the solution in the two regimes in Fig. 3.1. These pictures are made from the output files from the DNS code. With the DNS code we can also perform a numerical bifurcation analysis. We can just choose a Reynolds number and simulate the system. In this way we can study the behavior of the cylinder wake.

3.1.2 Numerical methods

Computational grid

The grid is defined on the domain shown in Fig. 1.1. The domain sizes are $-a < x < a$ and $-a < y < b$. The cylinder is centered in the origin and has a radius of $r = 0.5$. Although the computations are three dimensional with 2 cells in the z direction, we skip the the z direction in the figures. The behavior of the flow is two dimensional at low Reynolds numbers.
On the cylinder surface a no-slip condition is applied. The boundary conditions on the walls of our domain are such that there exists no boundary layer which influences the behavior of the cylinder wake. On the left $x = -a$, right $x = a$ and upper $y = b$ boundary a homogeneous Neumann condition is applied. On the inflow boundary $y = -a$ a Dirichlet condition is applied as $(u, v, w) = (0, 1, 0)$. In the $z-$direction periodic boundary conditions are applied.
The grid is stretched in the x and y direction and uniform in the z direction. On the grid the unknown velocities u, v, w and pressure p are defined by the marker and cell method (MAC) (see Fig. 3.2). The velocities are defined on the faces of the computational cells and the pressure at the centers of the cells.

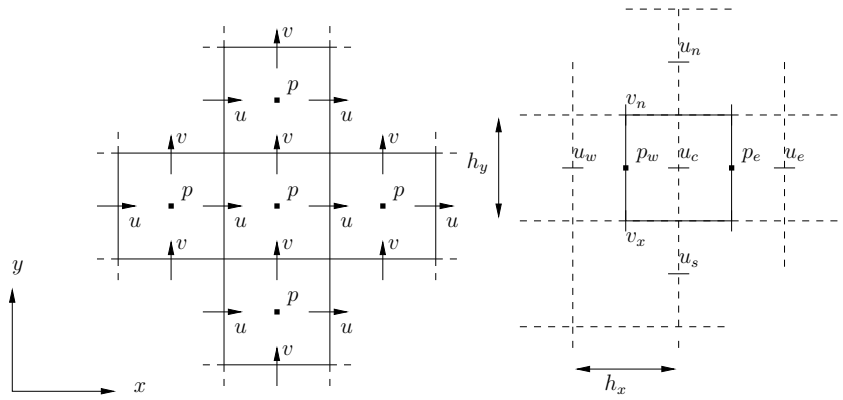


Figure 3.2: *Left: Locations of the pressure p and velocity components u, v. Note that the cells are not necessarily squares. They may be stretched in one or more directions. Right: The solid square is the control cell around u_c.*

Spatial discretization

For the spatial discretization a finite-volume method is applied. The discretization is based on the conservation of the physical quantities u, v and uses control cells around the unknowns u, v; see right picture in Fig. 3.2. In the control cells both conservation of mass and impulse is required. The convective flux $-\boldsymbol{u}(\boldsymbol{u} \cdot \boldsymbol{n})$ and diffusive flux through the control cells are discretised by a second order symmetry-preserving method [2]. In the right figure in Fig. 3.2 the control cell K for velocity u_c is shown.

Time integration

The time integration uses a second-order Adams-Bashforth method. Herefore, first the solution at time levels n and $n-1$ are extrapolated to

$$\boldsymbol{u}^{n+\frac{1}{2}} = \frac{3}{2}\boldsymbol{u}^{n} - \frac{1}{2}\boldsymbol{u}^{n-1}. \tag{3.1}$$

The extrapolated value is used to calculate the convective and diffusive fluxes in the control cells of u, v and w. Next, the velocity field $\boldsymbol{u}^n$ is updated by integration over one time step

$$\boldsymbol{u}^{*} = \boldsymbol{u}^{n} + \delta t(convective + diffusive)^{n+\frac{1}{2}}, \tag{3.2}$$

indicating that the convective and diffusive term are evaluated with the velocity from (3.1). After the time integration, the updated velocity field $\boldsymbol{u}^*$ does not satisfy $\nabla \cdot \boldsymbol{u}^* = 0$. We use the pressure equation to fix this.

Pressure equation

To compute the velocity field $\boldsymbol{u}^{n+1}$ we still have to add the pressure gradient

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^{*} - \delta t \nabla p^{n+1}, \tag{3.3}$$

where $\boldsymbol{u}^*$ has been computed by (3.2). Since we want $\nabla \cdot \boldsymbol{n}^{n+1} = 0$, the pressure p^{n+1} can be solved from the equation

$$\begin{aligned} \nabla \cdot (\boldsymbol{u}^n - dt \nabla p^{n+1}) = 0 \Rightarrow \\ \nabla^2 p^{n+1} = \frac{1}{dt} \nabla \cdot \boldsymbol{u}^n. \end{aligned} \tag{3.4}$$

The pressure equation (3.4) is solved with ICCG in the spectral space using fast Fourier transformation (FFT). The ICCG method is an advanced Conjugate Gradient method (CG) with an incomplete Cholesky factorization as pre-conditioner [21]. With the pressure from (3.4) we can calculate the new state $\boldsymbol{u}^{n+1}$ with (3.3).

3.1.3 DNS interface

The DNS code needs a lot of information. First of all, to work properly, it needs some information to configure the numerical methods discussed in previous section. The properties of the time integration, convergence criteria for the Poisson solver and order of discretization are needed. Furthermore it needs settings about the size of the calculation domain, the number of cells and the boundary conditions. There are also some problem related settings: the Reynolds number Re, time step δt and the number of time steps nt. The product of the latter two specifies the time integration interval. And finally some output settings determining the storage of the state vector at every time interval.
When starting the DNS code, the user can choose to start with an initial state vector. When there is no initial state vector defined, DNS starts computing the flow with a zero state vector. In our project, the DNS code always has to start with a given state vector.

3.2 PDEcont

3.2.1 What does PDEcont do?

PDEcont computes branches of solutions y of a physical system and the corresponding stability information with the theory described in chapter 2. It uses a Newton-based single shooting technique combined with a Picard iteration process. With PDecont we can get an overview of all the asymptotic solutions y, i.e., stable and unstable solutions in a certain range of the physical parameter γ. The critical values γ_c, where certain instabilities occurs, can be also found. Due to this continuation code we get a good understanding of the behavior of a physical system in certain conditions. With this kind of research, we hope to avoid events like the collapse of the Tacoma Narrows suspension bridge.
To understand how PDEcont works we continue with the numerical methods used by PDEcont.

3.2.2 Numerical methods

The Floquet multipliers μ of the monodromy matrix M characterize the stability of a periodic solution $y^*(t)$. It is too expensive to compute all the Floquet multipliers and moreover, most of these multipliers aren't even interesting.
A property of the physical system we investigate is the fact that most multipliers are close to zero and thus are not able to cause any instability of $y^*(t)$. The instability is caused by

those multipliers which are close to the unit circle. PDEcont computes only these dominant multipliers of M. To define the most dominant Floquet multipliers of the system, PDEcont uses the following assumption.

Assumption 3.2.1 *Let $y^* = (y^*(t_0), T^*)$ denote an isolated solution to (2.15), and let $\mathcal{B}$ be a small neighborhood of $y^*(t)$. Let $M(y) = \frac{\partial \varphi}{\partial y(t_0)}(y)$ for $y \in \mathcal{B}$ and denote its eigenvalues by $\mu_i, i = 1, ..., N$. Assume that for all $y \in \mathcal{B}$ precisely p eigenvalues lie outside the disk*

$$C_\rho = \{|z| < \rho\}, \quad 0 < \rho < 1 \tag{3.5}$$

and no eigenvalues has modulus ρ; i.e., for all $y \in \mathcal{B}$

$$|\mu_1| \geq |\mu_2| \geq \cdots \geq |\mu_p| > \rho > |\mu_{p+1}|, \cdots, |\mu_N|. \tag{3.6}$$

The dominant Floquet multipliers defined by this assumption and the corresponding eigenvectors are computed using an orthogonal subspace iteration technique. PDEcont computes the basis for the p-dimensional subspace $\mathcal{U} \subset \mathbb{R}^N$ of the generalized eigenvectors corresponding to $\mu_1, \cdots, \mu_p$. Let the column vectors of $V_p \in \mathbb{R}^{N \times p}$ define an orthonormal basis for the subspace $\mathcal{U}$ of $\mathbb{R}^N$ and the column vectors of $V_q \in \mathbb{R}^{N \times (N-p)} = \mathbb{R}^{N \times q}$ define an orthonormal basis for $\mathcal{U}^\perp$, the orthogonal complement of $\mathcal{U}$ in $\mathbb{R}^N$. These matrices V_p and V_q can be computed by the real Schur factorization of M. Note that V_q is a huge matrix and we do not want to compute this matrix in the final algorithm.
Now we can construct orthogonal projectors P and Q of $\mathbb{R}^N$ onto the subspaces $\mathcal{U}$ and $\mathcal{U}^\perp$ respectively as

$$\begin{aligned} P &:= V_p V_p^T, \\ Q &:= V_q V_q^T = I_N - V_p V_p^T. \end{aligned} \tag{3.7}$$

Now for any $y(t_0) \in \mathbb{R}^N$ there exists a unique decomposition

$$y(t_0) = V_p \bar{p} + V_q \bar{q} = P y(t_0) + Q y(t_0). \tag{3.8}$$

Consider the system (2.18),

$$\begin{bmatrix} M - I & b_T & b_\gamma \\ c_s^T & d_{s,T} & d_{s,\gamma} \\ c_n^T & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta T \\ \Delta \gamma \end{bmatrix} = - \begin{bmatrix} r \\ s \\ n \end{bmatrix}, \tag{3.9}$$

where we abbreviated the derivatives in the iteration matrix. By multiplying the first N equations with $[V_q \ V_p]^T$ and substituting (3.8) into (3.9) we can write the system in terms of these matrices V_p and V_q. This gives the possibility to split the system into a $q-$ and a $p-$ dimensional subsystem, where $p << q$.
We also know $V_p^T V_q = 0_{p \times q}, V_q^T V_p = 0_{q \times p}$ and $V_q^T M V_p = 0$, where the latter equation holds since $\mathcal{U}$ is an invariant subspace of $M(y(t_0), T, \gamma))$. We get the following system:

$$\begin{bmatrix} V_q^T M V_q - I_q & 0 & 0 & V_q^T b_\gamma \\ V_p^T M V_q & V_p^T M V_p - I_p & V_p^T b_T & V_p^T b_\gamma \\ c_s^T V_q & c_s^T V_p & d_{s,T} & d_{s,\gamma} \\ c_n^T V_q & c_n^T V_p & d_{n,T} & d_{n,\gamma} \end{bmatrix} \begin{bmatrix} \Delta \bar{q} \\ \Delta \bar{p} \\ \Delta T \\ \Delta \gamma \end{bmatrix} = - \begin{bmatrix} V_q^T r \\ V_p^T r \\ s \\ n \end{bmatrix}. \tag{3.10}$$

From the system (3.10) we can write the q system as

$$(V_q^T M V_q - I_q)\Delta\bar{q} = -(V_q^T r + V q^T b_\gamma \Delta\gamma). \tag{3.11}$$

Since $(V_q^T M V_q - I_q)$ is nonsingular by construction we can transform (3.11) into

$$\Delta\bar{q} = -(V_q^T M V_q - I_q)^{-1}(V_q^T r + V q^T b_\gamma \Delta\gamma). \tag{3.12}$$

We split the $\Delta\bar{q}$ into two components $\Delta\bar{q_r}$ and $\Delta\bar{q_\gamma}$ as

$$\Delta\bar{q} = \Delta\bar{q_r} + \Delta\gamma\Delta\bar{q_\gamma}, \tag{3.13}$$

and obtain two separate subsystems

$$(V_q^T M V_q - I_q)\Delta\bar{q_r} = -V_q^T r \text{ and } (V_q^T M V_q - I_q)\Delta\bar{q_\gamma} = -V_q^T b_\gamma. \tag{3.14}$$

Each of these subsystems is solved approximately by a Picard iteration scheme. To avoid computing the large matrix V_q, we use the projectors P, Q (3.7) by multiplying every iteration step with V_q. Let b denote either r or b_γ. The Picard iteration is

$$\left\{ \begin{array}{rcl} \Delta q^{[0]} & = & 0, \\ \Delta q^{[i]} & = & V_q V_q^T M \Delta q_b^{[i-1]} + V_q V_q^T b \\ & = & Q M \Delta q_b^{[i-1]} + Q b, \quad i = 1, \cdots, l, \\ \Delta q_b & = & \Delta q_b^{[l]} \end{array} \right. \tag{3.15}$$

where $\Delta q_b = \Delta q_r$ if $b = r$ and $\Delta q_b = \Delta q_\gamma$ if $b = b_\gamma$. Substituting $\Delta\bar{q}$ into (3.10) and rewriting in terms of the components $\Delta\bar{q_r}$ and $\Delta\bar{q_\gamma}$ we get

$$\left[\begin{array}{ccc} V_p^T M V_p - I_p & V_p^T b_T & V_p^T(b_\gamma + M V_q^T \Delta\bar{q_\gamma}) \\ c_s^T V_p & d_{s,T} & d_{s,\gamma} + c_c^T V_q \Delta\bar{q_\gamma} \\ c_n^T V_p & d_{n,T} & d_{n,\gamma} + c_n^T V_q \Delta\bar{q_\gamma} \end{array} \right] \left[\begin{array}{c} \Delta\bar{p} \\ \Delta T \\ \Delta\gamma \end{array} \right] = - \left[\begin{array}{c} V_p^T(r + M V_q \Delta\bar{q_r}) \\ s + c_s^T V_q \Delta\bar{q_r} \\ n + c_n^T V_q \Delta\bar{q_r} \end{array} \right]. \tag{3.16}$$

First the two systems (3.14) are solved approximately using Picard iteration (3.15). Then we compute $\Delta\bar{p}, \Delta T$ and $\Delta\gamma$ from (3.16) and finally $\Delta\bar{q}$ is computed with (3.13)
The nice thing of this approach is that PDEcont computes branches of solutions and also returns stability information at little or no extra cost.

Variable step size

In PDEcont we use a variable step size. It is not safe to fix the step size $\Delta\eta$, because when the step size is too large there is the possibility to jump to another branch or jump over a bifurcation point. PDEcont decides how large the step is, the convergence of the Newton-Picard iterations. When the parameter step $\Delta\eta$ in the computed direction is too large, the Newton-Picard step will fail. As a result, half the steplength $\Delta\eta$ will be used. When the Newton-Picard step succeeds, the steplength will grow by a factor. The behavior of the steplength can be configured by the user. For detailed information see the manual [22].

3.2.3 PDEcont interface

First of all, to run the PDEcont code a lot of work is required. We can only start PDEcont when the system to investigate, is implemented. A lot of settings PDEcont needs are dependent on how the user has implemented the system. However PDEcont still needs a lot of settings for configuring the solver and the continuation method. Also some settings of the solution type and branch type are required.
The solver has to deal with three iteration processes, namely the Newton-Raphson iteration, orthogonal subspace iteration and the Picard iteration. Every iteration process needs a lot of setting as thresholds, convergence criteria, maximum number of iteration steps, etc.
Also the continuation method requires a lot of settings. PDEcont uses a variable stepsize method. Here settings are needed for minimum, maximum and starting stepsize, some constants which have to do with the behavior of the stepsize, the range of the branching parameter γ and the number of branch points to compute in that range. Finally PDEcont needs of course a starting point (y, T, γ) close to a branch. All the details are in the PDEcont manual [22].

3.3 What to do?

For this project we want to compute the branch of steady-states and periodic solutions and give an accurate result of the location of the Hopf bifurcation. We have to implement our flow problem into PDEcont and compute the branches in combination with the DNS code. PDEcont is the main program. PDEcont starts the continuation process with a point $x_i = (y_i, T_i, Re_i)$. As a result PDEcont gives the point on the branch $x_i^* = (y_i^*, T_i^*, Re_i^*)$. To find this point x_i^*, the solver from PDEcont needs to compute some directional derivatives and the derivatives with respect to Re and T of $\varphi(y, T, Re)$ at every iteration step. This is explained in section 2.2. The φ is computed by the DNS code. This means that the DNS code needs a y, T and Re. While running, PDEcont gives DNS the y, T and Re. DNS sends the results back and PDEcont uses these results to continue the continuation process. This exchange of information is very important to couple the codes. In the Fig. 3.3 we give a schematic view of the information exchange.

In the next chapter we describe in detail how this is done.

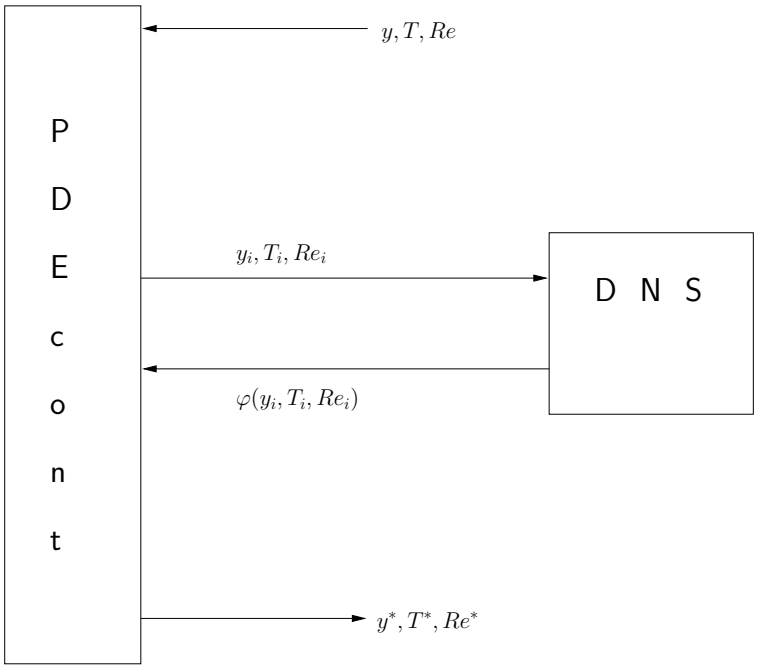


Figure 3.3: *Schematic view of the data exchange between PDEcont and DNS.*

Chapter 4

Implementation

From section 3.3 we know that both programs have to work together and that some information must be exchanged between the codes. Therefore we have to create some communication routines and implement them in PDEcont. When PDEcont needs a solution of a time integration it gives the proper information to the DNS code. The DNS code gets the calculation job from PDEcont and when finished it gives the results back to PDEcont.
The communication routines are created with the knowledge we have of the format of the files containing Re, T and y. These routines are created specific for this DNS code and cannot be used for other purposes.

4.1 Implementation methods

There are different methods of implementing a system into PDEcont. The implementation method is very dependent on the simulation code and the creative powers of the programmer. In this project, we have to find out to what extent the DNS code can be used for timestepper-based bifurcation analysis. We can interpret this again into different implementation methods. We discuss two of them.

4.1.1 Possible methods

Our first idea was to use the DNS code for what it is and do all the necessary data exchange shown in Fig. 3.3 by ASCII files. The communication routines read and write the data in the proper format. Nothing of the concerning content of the DNS code is skipped or changed. The second possibility is to split up the DNS code in a subroutine that does the initializations and a subroutine for the time integration and link both routines into PDEcont. The codes are written in different programming languages, namely DNS in Fortran and PDEcont in C. This method requires thus more programming in both languages Fortran and C, but when finished, it is probably easier to run and easier to debug. We used the first method and will not discuss the second method any further.

4.1.2 Our method

We have chosen to exchange the data Re, T and y by files and leave the DNS code for what it is. The method we used makes the implementation and the connection easy, because we don't have to strip down the DNS code and all the programming work will be in C. This means

the program work is a lot easier compared with the other method we mentioned. Although some small changes in the DNS code are required. These are discussed in Chapter 6. We think that the communication between the codes slows down the performance. More of this method in our final discussion in Chapter 6.

4.2 Implementation with files

4.2.1 Files for data exchange between PDEcont and DNS

In Section 2.2.2 we showed that the solver from PDEcont needs at every iteration step a time integration φ of state vector y over a time interval t at a certain Reynolds number. In DNS the state vector y contains four components, namely the velocities u, v, w and pressure p. Since we perform a numerical bifurcation analysis on a two dimensional flow problem we neglect the velocity component w. We also neglect the pressure p, because we want to keep the state vector y as small as possible to save computation time. Besides pressure p is obtained from the velocity field (u, v) and thus is the pressure not important in the state vector y. Thus the state vector PDEcont contains two components $y = (u, v)$. In Fig. 3.3 from Section 3.3 a schematic view is shown. The state vector y, Re and time interval T are exchanged by files. DNS uses two types of files, namely a file with parameter settings and a file containing the state vector. In Fig. 4.1 we give a schematic view of the file exchange and we can see that the files `parameters` and `fld##.dat` are important in the data exchange between PDEcont and DNS.

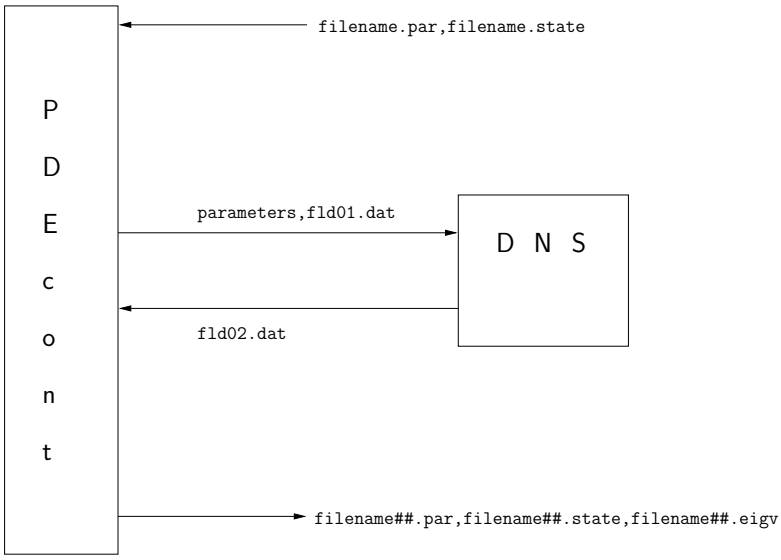


Figure 4.1: *Data exchange data with files.*

The parameter file

The value of Re , the time step δt and the number of time steps nt are specified in the DNS file `parameters`. The latter two are derived from the time interval when generating the

parameter file. The file `parameters` is divided into sections and every section contains some settings of parameters. An important section is for example

```
----- Time integration  -------------------------

         nt = 1398
         dt =  4.9956631298147492e-03
       beta =  5.0000000000000003e-02
```

The important parameters from the `parameter` file are `Re, nt, dt, nsf`. `nsf` is easily missed and very important. This value plays an important role in the output of DNS. This parameter is an integer and every `nsf` time steps the state is stored in the state file. Since we are only interested in the state at time step `nt`, `nsf` must be a divisor of `nt` to store the final state. We take `nsf` equal to `nt`, because storing the state every `nsf` time steps will slow down the computations.

The state vector file

Another important file can be found in the `dat/` directory of DNS, namely the state file `fld##.dat` containing the state vector y. The state file `fld##.dat` contains 3 data elements, namely the time t, the state vector in a matrix with 4 columns (u, v, w, p) at time t and finally the state in a matrix with 3 columns (u, v, w) at time $t - \delta t$, where δt is the time step defined in the file `parameters`. The run number `##` in the state filename is defined in the `Makefile` of DNS. This number specifies the file containing the initial state vector $y(t_0)$. Every time integration has a run number and the output files containing the result has the run number `##+1` in the file name. The setting in this file is very important to start with the file containing the initial state vector. For example, `fld01.dat` contains the state vector $y(t_0)$. As a result DNS generates the file `fld02.dat` containing the state vector $\varphi(y(t_0), t; Re)$.

Somehow PDEcont has to read from and write information to these files. The state vector y and parameters Re, dt, nt are written to the standard input filenames of DNS, `fld01.dat` and `parameters` respectively. The resulting state of DNS is always written to the file `fld02.dat`. We use the same run number for every run, so we don't need to recompile the code.

4.2.2 Files for initialization and results of PDEcont

PDEcont also uses some files. As a result, PDEcont gives a solution (y^*, T^*, Re^*) on the branch with the corresponding stability information of this branch point. To analyze all the branch points we save every point in files. The names of the files are defined as `filename##` where the `filename` is defined by the user through the command line options of PDEcont. The `##` is the number of the point on the branch assigned by PDEcont. PDEcont restarts the numbering from one at every run.
To make clear the contents of the files, we defined three types of files: files with the extensions `.par`, `.state` and `.eigv`, containing the parameter settings, the state vector y and the corresponding eigenvalues respectively. These files are defined only for the output of PDEcont.(See Fig. 4.1) It is also possible to restart the continuation process with the output files `.par` and

`.state`.

The `.state` files contains two data elements, namely the time t and the state $y^*(t) = (u^*, v^*, w = 0, p = 0)$ on the branch. This file is not the same as the state file `.dat` from DNS. Remember that the state file from DNS contains 3 data elements, namely the time t, the state $y(T) = \varphi(y(t_0); Re)$ containing the four components (u, v, w, p) and the state at $y(T - \delta t)$ containing three components (u, v, w). These state files are thus not directly useable for DNS. When a branch point is computed, PDEcont stores the state to `filename##.state`.

The format of the `.par` files are exactly the same as the format of the `parameter` file DNS uses and contains exactly the same information. PDEcont starts with a solution $y = (y(t_0), T, Re)$ close to the branch, where T =`nt`$*$`dt`. The corresponding parameter settings of the new branch point are stored in `filename##.par`.

Finally, when the next point on the branch is found, the corresponding stability information is stored in the `.eigv` file.

We start PDEcont with a point (y, T, Re) close to the branch. PDEcont needs two starting files `filename.state` and `filename.par` containing y and T, Re respectively. The `filename` can be specified by the use in the configuration file of PDEcont. When the next branch point is found all the necessary information is stored in the files `filename##` and from the last calculated point PDEcont continuous to compute the next branch point.

4.2.3 Implementation of the IO routines

Now we know which files are important and we have seen in Fig. 4.1 that two types of files are used for the data exchange between PDEcont and DNS. We need two routines which are able to read the files containing the parameter settings and the files containing the state vector. We have to keep in mind that the files `fld##.dat` and the `.state` files do not have the same format. Note that the format of the `.par` files and `parameters` is the same.
We also need two routines which are able to write the parameter files and the state files. With these four IO routines PDEcont must be able to read data and change data in these files. We created the routines `ReadPar, WritePar, ReadState` and `WriteState` based on the known format of the `parameters` and `fld##.dat` from DNS. The routines are written in C such that they can be implemented in PDEcont. The technical details of these IO routines can be found in the appendix.

ReadPar

This routine is programmed to read the `.par` files and the file `parameters` from DNS. These files have a fixed format. This routine scans the files for the parameter names `Re, nt, dt, nsf` and stores the values of these parameters in memory, such that PDEcont can use these settings.

WritePar

When this routine is called, the parameter values of `Re, nt, dt, nsf` stored in memory are written to a `.par` file or `parameters`. Because the format is fixed the initial `filename.par`

will be copied, but only the changed parameter values of `Re, nt, dt, nsf` are overwritten to the new target file. This target file could be a new `filename##.par` or `parameters`.

ReadState

This routine has two modes, namely reading a DNS state file `fld##.dat` or a PDEcont database file `filename##.state`. There is a small difference between both files as already explained. With an argument from this routine we can decide to read the third data element, $y(T - \delta t)$, from `fld##.dat`. We can skip this element when reading a `.state` file. This is also a very important option when computing points on the periodic branch. When PDEcont is computing points on a steady-state branch we don't need the $y(T - \delta t)$. When computing points on the periodic branch we need the derivative of the state $y(t)$ to time T which we compute from $y(T)$ and $y(T - \delta t)$.

WriteState

When this routine is called, the current state $y = (u, v)$ in memory is written to a file. This routine has also two modes. It can write `fld##.dat` files and `.state` files. The difference is again the third data element $y(T - \delta t)$. When the state $y = (u, v)$ is needed by DNS, then we write the state to the format of `fld##.dat` with the proper arguments of this routine, where w and p are zero. The state $y(T - \delta t)$ needed for this format is unknown and instead we write the state $y = (u, v)$ with $w = 0$. The effect of this on the calculations from DNS is discussed in chapter 6. If the state is a branch point then $y^* = (u^*, v^*)$ is written to a `.state` file.

4.2.4 Testing the IO routines

After creating the IO routines we have to test if the routines work properly. We give now the description of our test. The test gives an idea how the PDEcont is connected to DNS. The following situation is executed with a small C program which is just a sequence of calls to the IO routines. It is a test of the communication between a C program and the DNS code with files.

Imagine we start with $y^*(t_0)$ on the branch of periodic solutions and call it `point1`. We have two data files of this `point1`, namely `point1.par` with the parameter settings and `point1.state` containing the state $y^*(t_0) = (u^*(t_0), v^*(t_0))$. The routine `ReadPar` reads the initial parameters from `point1.par`. `ReadState` reads the state from `point1.state` and `WriteState` copies it to `fld01.dat`. This file `fld01.dat` contains the initial vector field $(u^*(t_0), v^*(t_0), w = 0, p = 0)$ for the DNS code. Assume that after the first call of `ReadPar` the parameters are changed by PDEcont. The new parameter values are copied with `WritePar` to the standard parameter file `parameters`.
Now DNS has the necessary files to integrate $y^*(t_0)$ over a time interval `nt*dt` at `Re`. As a result we have `fld02.dat` with the corresponding parameter settings `parameters`. These files are again read and written to the resulting files `point2.par` and `point2.state`. Notice that the format of these resulting files is in such a way that the process can be repeated.
After the test it is very important that no data is missing. Furthermore we have to make sure that no digits are lost. After calling `ReadPar` and `Writepar` we compare a small part

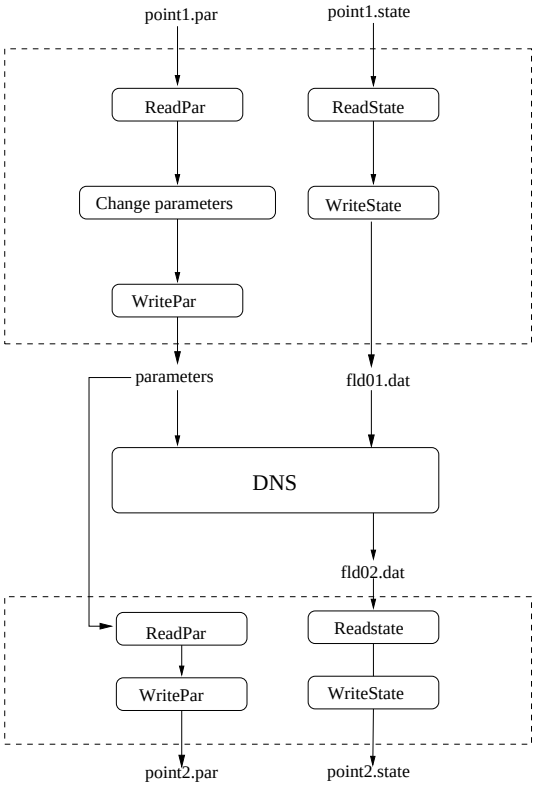


Figure 4.2: *Schematic view of the communication between DNS and the C program which uses the IO routines.*

of `point1.par` with `parameters`. Between the read and write routines there is a parameter change. We present a part of the files below to show that `ReadPar` and `Writepar` works fine.

`point1.par`

```
----- Physical parameters ------------------------

      Re =  8.0000000000000000e+01


----- Time integration  --------------------------


        nt =  60
        dt =  2.0000000000000000e-03
      beta =  8.0000000000000002e-02
```

`parameters`

```
----- Physical parameters ------------------------

         Re =  9.0000000000000000e+01


----- Time integration  --------------------------

         nt =  70
         dt =  2.0000000000000000e-03
       beta =  8.0000000000000002e-02
```

Furthermore we compare the first few lines of the state files `fld02.dat` resulting from DNS with `point2.state` after calling `ReadState` and `WriteState`. The first line of both files contains the time t at which this state occurs. Then follows the state $y(t)$ at time t. Note the difference expained between the `.dat` file and the `.state` file as explained.

`fld02.dat`

```
 0.3002000000000010E+03
 0.0000000000000000E+00  0.1000000000000000E+01 ...  ...
 0.6377042586634170E-05  0.1000000000000000E+01 ...  ...
 0.1256226285825501E-04  0.1000000000000000E+01 ...  ...
 0.1855667991431518E-04  0.1000000000000000E+01 ...  ...
 0.2438159870673748E-04  0.1000000000000000E+01 ...  ...
```

`point2.state`

```
 3.0020000000000101e+02
 0.0000000000000000e+00  1.0000000000000000e+00 ...  ...
 6.3770425866341704e-06  1.0000000000000000e+00 ...  ...
 1.2562262858255010e-05  1.0000000000000000e+00 ...  ...
 1.8556679914315180e-05  1.0000000000000000e+00 ...  ...
 2.4381598706737479e-05  1.0000000000000000e+00 ...  ...
```

The test program is executed successfully. From this test we can conclude that with the four IO routines it is possible to let a C program, like PDEcont, and a Fortran program exchange data with each other.

4.2.5 Implementation of the system definition

PDEcont is a continuation code, but the code doesn't know anything about our flow problem and its governing equations. We now continue with the implementation of our flow problem. In PDEcont a system definition API is defined [22], Application Program Interface. This is a set of routines which are needed for the implementation of a system. This is possible, because

for a stability analysis PDEcont always needs the same ingredients. These ingredients are shown in Chapter 2. A programmer can use these routines to implement a system and let PDEcont perform a bifurcation analysis on it. The input and output of the routines are already defined in the code [22]. The programmer has to program the routines.

System API

The following routines are used to implement a system :

- `SysInit()`: This is the initialization phase of the system. The routine reads settings from the configuration file, not related with the calculations. For example the filename `filename` containing the initial state y and parameter settings or some time integration settings.
- `SysHelp()` : This routine gives information about the system and its parameters.
- `SysStart()`: Read starting state y, period T expressed in `dt` and `nt`, value of branching parameter Re. These values are found in the file defined in the routine `SysInit`.
- `SysPrintStateR()`: Prints certain data to screen.
- `SysStoreState()`: This routine can be used to create the files `filename##.par`, `filename##.state` and `filename##.eigv`. The files get a unique number for every new point on the branch. The `filename` is specified by the user with a PDEcont command line option.
- `SysRHS()`: In this routine the user can define the $f(y, \gamma)$. We do not need this routine in our experiments.
- `SysIntegrate()`: This routine integrates over a time interval T. Due to the choice of the implementation method this routine contains the connection to DNS code by means of the IO routines. It also returns the derivative of the state y with respect to T when needed.
- `SysMV()`: Matrix vector multiplication with a matrix like the monodromy matrix. This multiplication is the directional derivative of φ in the direction of vector v. This routine computes MV_p in the iteration scheme of (3.16) and is also used in the Picard iterations.
- `SysDDpar()`:This routine computes the derivative of φ with respect to the continuation parameter Re.
- `SysSysDim()`: This routines returns the dimension of our system.

For the implementation of our flow problem in PDEcont we have to program the API routines. Together with our IO routines PDEcont must be able to do a numerical bifurcation analysis. We know from Chapter 2, PDEcont solves the new point on the branch by iterating to the exact solution y^*. For this, PDEcont needs the derivatives of φ with respect to T and γ and matrix-vector products with $M = \partial\varphi/\partial y$. For this the routines `SysIntegrate, SysDDpar` and `SysMV` are used. We explain these most important routines in detail.

SysInit

This routine reads some values of variables. The values of these variables do not change during the calculation of a branch of solutions. All these values are stored in a configuration (`Cfg`) file. In our case the following values are read from the `Cfg` file:
`-fname`: The name of the file containing the starting point on a branch
`-diff_order`: The order of the finite difference method for computing a derivative
`-h_difference`: Determines the finite difference stepsize.
`-integration_info`: Value for printing the used finite difference method to screen.

Also some memory for workspaces is reserved.

SysStart

This routine reads from the file, specified in the `SysInit` routine, the initial branching parameter value Re, initial value of the period T and the initial state vector y. This routine is used only in the start of PDEcont. Here comes the IO routines in use, namely `ReadPar` and `ReadState`. The length of the time interval T must be calculated from `dt` and `nt`. This routine also creates the grid for the DNS code.

SysIntegrate

In our experiment this routine is in fact the connection to the DNS code. It looks very similar to our IO routine test. This routine sends the state y, T and Re to DNS. DNS returns the result $\varphi(y, T; Re)$. In Chapter 4.2.1, we saw that in result file `fld##.dat` the final state and the state at a previous time step can be found. In combination with the time step `dt` we make a first order approximation of

$$\frac{\partial \varphi(y(t_0), T, Re)}{\partial T} \approx \frac{\varphi(y(t_0), T, Re) - \varphi(y(t_0), T - dt, Re)}{dt}. \tag{4.1}$$

This derivative is only computed when computing a periodic solution.

SysMV

In the iteration process (3.16) we can see that at every iteration step we need the matrix vector multiplication MV where V contains p vectors. It is also needed in the Picard iteration process. The matrix-vector multiplication MV is computed by a finite difference method. For the directional derivative a first, second and fourth order finite difference method is implemented. The user can choose the order in the configuration file. The value of h is also defined in the this file. The settings for the finite differences are initialized in the `SysInit` routine. We used a first order approximation of the matrix vector multiplication, namely

$$M(y(t_0), T, Re)v \approx \frac{\varphi(y(t_0) + hv, T, Re) - \varphi(y(t_0), T, Re)}{h}, \tag{4.2}$$

in all our experiments. Here

$$h = h_{difference}\frac{||u||_2}{||v||_2}. \tag{4.3}$$

SysDDpar

The derivative of φ with respect to the branching parameter Re is calculated with this routine. For this derivative we also use a finite difference technique. Here it is also possible to use a first, second or fourth order method. The settings of this finite difference method are equal to the settings from `SysMV`, hence in all our experiments we used a first order approximation of this derivative,

$$\frac{\partial\varphi(y(t_0), T, Re)}{\partial Re} \approx \frac{\varphi(y(t_0), T, Re + h) - \varphi(y(t_0), T, Re)}{h}. \tag{4.4}$$

SysStoreState

When a branch point (y^*, T^*, Re^*) is found, this routine creates the files `.state, .par` and `.eigv` containing the state vector y^*, parameters T, Re and the corresponding eigenvalues respectively. The latter file gives us the stability information. With Matlab we can visualize the results. (See Section 4.3).

SysPrintStateR

This routine prints the following data to screen: the 2 norm of the u and v component, the current value of the branching parameter Re and the current value of the period T.

All these routines together with the IO routines must be programmed is a `C` file called the system implementation file. After the proper configuration of the `Makefile` of PDEcont the routines from system implementation file will be implemented in PDEcont.

4.3 Matlab

As a result of defining the special files `.par, .state` and `.eigv` we are forced to create a Matlab routine to read all the data from these specific files for visualizing the results. We are interested in the stability of the branch points calculated in a specific range of the physical parameter Re. Every branch point, which represents the state vector y^* in the `.state` file has a corresponding `.par` file which gives the location of Re and the stability information in the `.eigv` file. To get a good overview of the stability of the branch points y^* in a certain range of Re we have to collect for every Re value from the `.par` files the corresponding stability information from `.eigv` files. Since we have two regimes in our flow problem, namely the steady-state and the periodic regime we have two different procedures in our Matlab routine. In the steady-state regime, the routine collects at every branch point y_i^* the corresponding Re_i number and the right most eigenvalue λ_i and plot this λ_i as function of Re_i.
In the periodic regime, the routine collects at every branch point y_i^* the corresponding Re_i

number and the largest in modulus non trivial Floquet multiplier μ_i and plots this μ_i as function of Re_i.

Chapter 5

Results

In this project we have two goals, namely the numerical bifurcation analysis of the flow problem and finding out if the Newton-Picard method in PDEcont works with the DNS code for this problem. For the second goal of this project the discretization errors in DNS are not very important. However, for our first goal it is nice to have results which are in good agreement with results from literature. Therefore we did calculations on different domains. To study the combination of PDEcont and DNS we used a small domain with relatively few cells. To find the Hopf bifurcation we used a larger domain with more cells. The flow problem is situated in a space with no boundaries. Probably our results are better when we choose our domain very large, such that the boundary conditions have no influence on the cylinder wake.

5.1 Results with simulation

5.1.1 Estimates of the eigenvalues

First we choose a suitable observation point in the calculation domain, not far from the cylinder, where a lot is happening with the flow. This point corresponds to the i^{th} component of the state vector $y \in \mathbb{R}^n$. To investigate the stability of the system, we follow the trajectory of the $y_i = (u_i, v_i)$ components in this fixed point. When we are in the steady state regime $Re < 46$, the trajectory of the solution $y_i(t)$ will converge to y_i^* for $t \to \infty$. In time, $y_i(t)$ will approximately evolve as

$$y_i(t) - y_i^* \approx e^{\lambda_d t}(y_i(t_0) - y_i^*) \tag{5.1}$$

where λ_d is the dominant eigenvalue of A. With these trajectories of u_i or v_i we can make estimates of this dominant eigenvalue λ_d which determines the rate of convergence to y_i^*.

For the periodic solution at $Re > 46$, we can do something similar. Any trajectory starting close enough to the periodic solution will converge to it, and the asymptotic speed at which this happens is governed by the dominant non-trivial Floquet multiplier μ_d.

Estimate the rightmost eigenvalues of the steady state

If we follow the rightmost eigenvalues λ_d when Re goes to the critical Reynolds number Re_c, we shall see that there is a pair of complex conjugate eigenvalues leaving the complex left

half plane. At $Re_c \approx 46$, $\Re(\lambda_d) = 0$ and $\Im(\lambda_d) = \beta \neq 0$. We will now estimate the complex part of λ_d from the trajectory $y_i(t)$. For this estimate we used the grid with domain size $a = 30, b = 20$ and number of grid points $nx = 200, ny = 240$, because the results from our experiments using this grid are in good agreement with the results from literature.
We calculated the trajectory $y_i(t)$ at $Re = 45$ with initial state $y_i(t_0)$ at $Re = 47$. From this trajectory we can estimate β close to the Hopf bifurcation. The trajectory $y_i(t)$ is shown in the left panel of Fig. 5.1. We can make an estimate of β from the frequency $f = T^{-1}$ of the velocity component $u_i(t)$ of $y_i(t)$. From the $y_i(t)$ follows $f \approx 0.117$ and thus $\beta = 2\pi f \approx 0.74$. Near the Hopf bifurcation, at $Re = 45$, the dominant eigenvalue has $\Im(\lambda_d) \approx 0.74$.
In the left panel of Fig. 5.1, we see that the maximum of the amplitude of the oscillations decreases over time which means that $\Re(\lambda_d) < 0$. To estimate the rate of convergence we use the amplitude of $y_i(t)$. We denote $E^{[j]}$ as the j^{th} maximum of $y_i(t) - y_i^*$. Then from (5.1) $E^{[j]}$ approximately evolves as

$$E^{[j+1]} \approx e^{\Re(\lambda_d)T} E^{[j]}. \tag{5.2}$$

Taking the logarithm of (5.2) we obtain

$$\ln E^{[j+1]} \approx \Re(\lambda_d)T + \ln E^{[j]}, \tag{5.3}$$

and finally

$$\Re(\lambda_d) \approx \frac{\ln E^{[j+1]} - \ln E^{[j]}}{T}. \tag{5.4}$$

From the right panel of Fig. 5.1 we can estimate the real part. As a result we estimated $\Re(\lambda_d) \approx -0.0082$, where $T = f^{-1} \approx 8.547$.
With this result we can validate the stability analysis from PDEcont in the neighbourhood when following the branch of steady states.

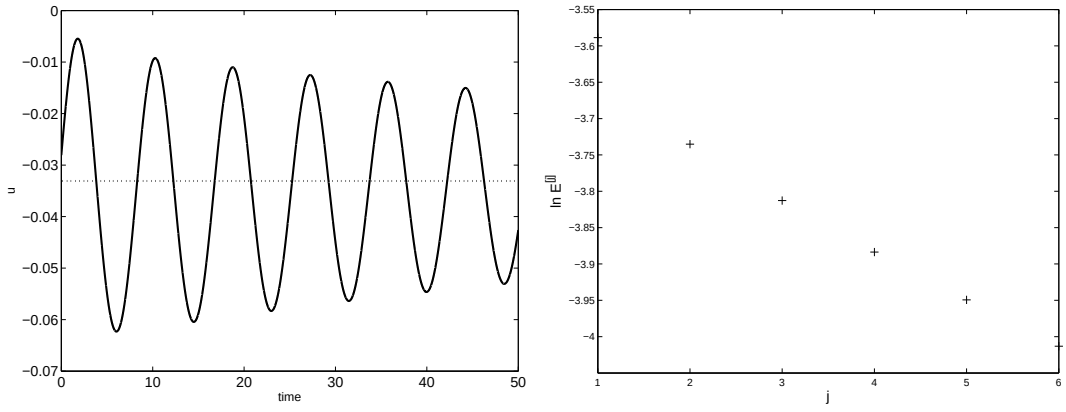


Figure 5.1: *Left : Velocity $u_i(t)$ (solid curve) in a fixed point against time at $Re = 45$. The dashed line is the stable steady state. Right : The $\ln E^{[j]}$ as function j.*

Estimate dominant Floquet multiplier $Re > 46$

To estimate this multiplier at $Re = 47$, close to the Hopf bifurcation point, we first determined the maximum of the velocity component u_i at our observation point. Then we started a

simulation from a nearby point (a point on the periodic solution at $Re = 50$) and we studied how the maximum of u_i for this trajectory evolved to the maximum for the periodic orbit. For estimating the dominant multiplier we use again the grid with domain size $a = 30, b = 20$ with $nx = 200$ and $ny = 240$ grid points since we want a good estimate. Let $E^{[j]}$ denote the difference between these maxima at the j^{th} maximum, then for large enough j,

$$E^{[j+1]} \approx \mu_d E^{[j]} = e^{\sigma_d T} E^{[j]} \tag{5.5}$$

with μ_d the dominant non-trivial Floquet multiplier and σ_d the corresponding Floquet exponent. Note that as the Hopf bifurcation point is approached, a real Floquet multiplier will approach one. So we are sure that close enough to the Hopf bifurcation point on a branch of stable orbits, the dominant non-trivial Floquet multiplier will be real, something which is needed for the above formula. Taking the logarithm of (5.5), we obtain

$$\ln E^{[j+1]} \approx \sigma_d T + \ln E^{[j]}$$

or

$$\sigma_d \approx \frac{\ln E^{[j+1]} - \ln E^{[j]}}{T}.$$

In the left panel of Fig. 5.2, we show the trajectory that we used and in the right panel the subsequent values of $E^{[j]}$ on a logarithmic scale in function of the time at which the maximum is obtained. From this figure we can easily estimate the period $T \approx 8.445$ and the dominant Floquet exponent $\sigma_d \approx -0.0161$, so $\mu_d \approx 0.87$.

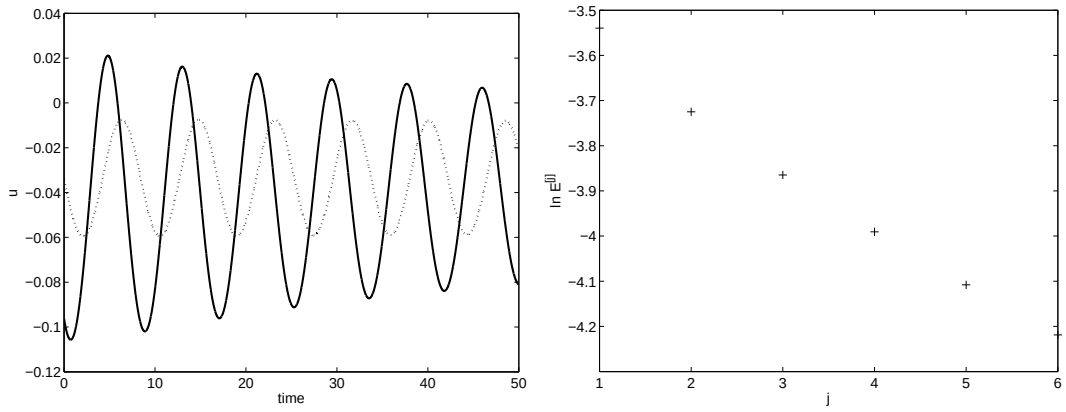


Figure 5.2: *Left: Velocity $u_i(t)$ in a fixed point against time at $Re = 47$ with $y_i(t_0)$ at $Re = 50$. The dashed line is the stable periodic solution $y_i^*(t)$ at $Re = 47$. The solid line is the perturbed solution. Right: The $\ln E^{[j]}$ as function of j.*

5.1.2 Frequencies of the vortex shedding at different Reynolds numbers

We can also use this fixed observation point in the domain to calculate the orbit $y_i^*(t)$ at different values of the Reynolds numbers. The frequency of $y_i^*(t)$ represents the frequency of the global behavior of the cylinder wake. As a result from the calculations we have a relation between the frequency and the Reynolds number. We can compare these results with the

domain			grid		Frequency f				
nr.	a	b	nx	ny	$Re = 47$	$Re = 50$	$Re = 70$	$Re = 80$	$Re = 100$
1	4	10	100	120	0.146	0.149	0.168	0.175	0.186
2	8	20	150	180	0.129	0.133	0.154	0.161	0.173
3	20	20	200	240	0.121	0.125	0.147	0.154	0.166
4	30	20	200	240	0.119	0.124	0.145	0.153	0.164

Table 5.1: *Frequency of periodic regime on different domains*

'universal' Strouhal-Reynolds curve from literature [19]. In our scaled model the frequency is equal to the Strouhal number. We did calculations on four different domains and plotted the results. On the largest domain, DNS gives us results which are in good agreement with the universal curve, Fig. 5.3. In Table 5.1 we give the domain settings and the corresponding results.
When changing the size of the domain, we have adapted the number of grid points in each direction in such a way that the mesh size near the cylinder hardly changes. We note that on the smaller domain, the results differ significantly from those in literature. On the two largest domains though, the results are very good. This shows that one needs to put the boundaries far enough from the cylinder.

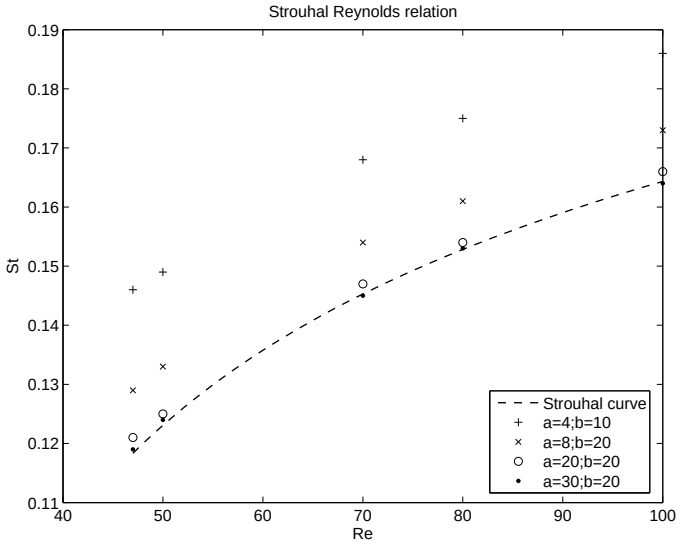


Figure 5.3: *The dashed line is the universal Strouhal curve. The numerical results on different domain sizes and grid sizes are plotted.*

5.2 Results with PDEcont

5.2.1 First results

To keep the calculation time low, we use domain nr. 1 from Table 5.1. We started from the steady state y^* at $Re = 20$ and computed the rightmost eigenvalue λ as function of Re, where Re is ranging from 20 up to 300. The Hopf bifurcation is found at $Re_c \approx 43,5$. We also followed the branch of periodic solutions. First we continued the branch from $Re = 100$ back to the Hopf bifurcation point. Later we calculated from $Re = 100$ up to $Re = 300$.

For the steady state branch for Reynolds 20 up to 300 we computed 58 points and for the

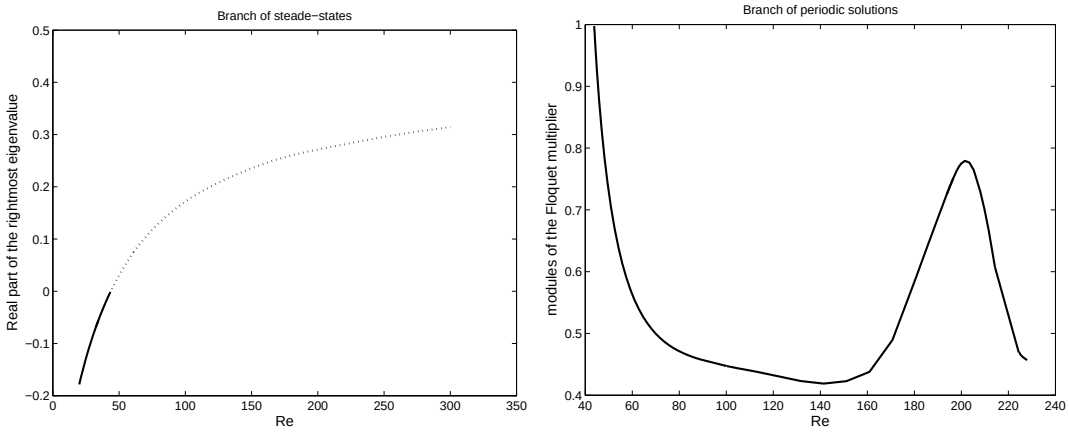


Figure 5.4: *Left: The branch of steady states in terms the corresponding rightmost eigenvalues as function of Re. The solid curve denotes a stable steady state and the dashed an unstable steady state. The transition takes place around* $Re_c \approx 43$. *Right: The branch of periodic solutions in terms of the Floquet multiplier with longest modulus as function of Re. At the Hopf bifurcation around* $Re_c \approx 43$ *the periodic regime starts. The dominant Floquet multiplier stays in the unit circle, i.e. the modulus stays smaller than one.*

periodic branch from Reynolds 43 up to 227 we computed 82 points. In Fig. 5.4, we show the real part of the rightmost eigenvalue at the steady state (left panel) and the modulus of the dominant non-trivial multiplier for the periodic solutions (right panel). From these figures, we can see that domain and grid nr. 1 is not sufficient to get a good result compared with the results from literature. The critical Reynolds number Re_c should be near ≈ 46, while our computations show the Hopf bifurcation at $Re \approx 43$.

We have already seen from the DNS results in Section 5.1 that the frequency is also not close to the results from literature. This small domain with a rather coarse grid is not good enough to obtain accurate results. This inaccuracy is due to the small domain where the upper and lower boundary have to much influence on the cylinder wake. However, we can see that the combination of PDEcont and DNS works fine.

domain			grid		Hopf bifurcation R_c
nr.	a	b	nx	ny	Re_c
1	4	10	100	120	$43.75 < Re_c < 43.85$
2	8	20	150	180	$45.46 < Re_c < 45.56$
3	20	20	200	240	$45.96 < Re_c < 46.06$
4	30	20	200	240	$46.34 < Re_c < 46.43$

Table 5.2: *The location of Hopf on different domains*

5.2.2 The Hopf bifurcation

We can locate the Hopf bifurcation at Re_c by starting the continuation process on both sides of the bifurcation point $Re \approx 46$. We start on the steady state branch and continue the branch until the real part of the dominant eigenvalue becomes positive. The point where $\Re(\lambda) = 0$ marks the Hopf bifurcation.
We can also start the continuation process on the periodic branch and follow the branch until the non-trivial dominant Floquet multiplier becomes one, i.e., there are two Floquet multipliers equal to one which marks the beginning of the periodic regime. This should be close to the the initial Reynolds number found during the continuation of the steady state branch. In the previous section we showed that on a small domain with a coarse grid, the results are not accurate enough.

Starting in steady state regime

Now we present the results of more accurate computations for locating the Hopf bifurcation Re_c. These computations are performed on different domains and grids, also used in Section 5.1. In Table 5.2 we present the boundaries of the interval in which the Hopf bifurcation can be found. Fig. 5.5 shows the real part of the rightmost eigenvalue pair for the four grids. The location of the Hopf bifurcation Re_c is marked where these curves crosses the value zero.

We can see from Fig. 5.5 that the location of the Hopf bifurcation Re_c gets closer to the value from literature $Re_c \approx 46$ [3] when our domain gets larger. The walls at $x = \pm a$ have much influence on the behavior of the cylinder wake. For good results these must be far away from the cylinder, i.e., a should be very large.

In Section 5.1 we estimated the complex part of the dominant eigenvalue λ_d at $Re = 45$, near the Hopf bifurcation. The estimate is done from computations on the largest domain with the most accurate grid we used. With this estimate we can validate the results from PDEcont. At $Re \approx 45$ the imaginary part of the rightmost eigenvalue computed by PDEcont is approximately 0.73 which is close to our estimate from the simulation. It is not really possible to compare the real parts of the rightmost eigenvalue from PDEcont and the estimate, because these values are too small. But since the imaginary part is already close to our estimate, the combination of PDEcont and DNS gives good results for the steady state regime.

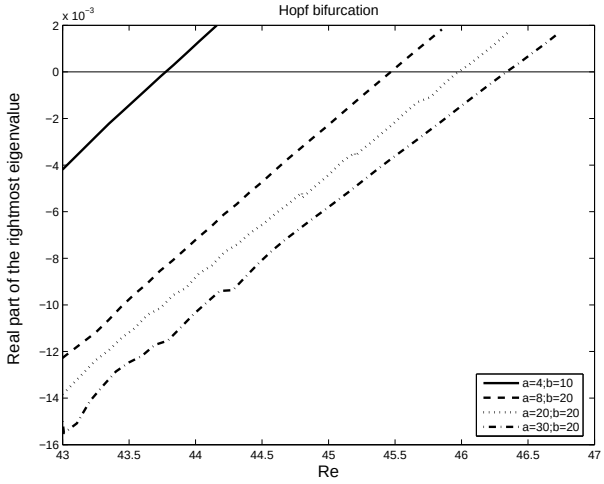


Figure 5.5: *Curves of the real part of the rightmost eigenvalues as function of the Reynolds number on different domains and grid sizes.*

Starting in the periodic regime

In Fig. 5.6 the results of the dominant Floquet multiplier μ_d as function of Reynolds at different domain sizes and grids are presented. We can see that from the other side of the bifurcation point, the Re_c marks the beginning of the periodic branch and the dominant non-trivial Floquet multiplier is 1.

We also estimated the dominant Floquet multiplier μ_d close to the Hopf bifurcation in Section 5.1. The non trivial dominant Floquet multiplier computed by PDEcont is approximately 0.96 which does not at all match with our estimate 0.88 from simulation. The estimated period $T \approx 8.445$ is close to the period computed by PDEcont $T = 8.463$. It is not clear what causes the discrepancy between the estimate of the Floquet multiplier and the computed value.

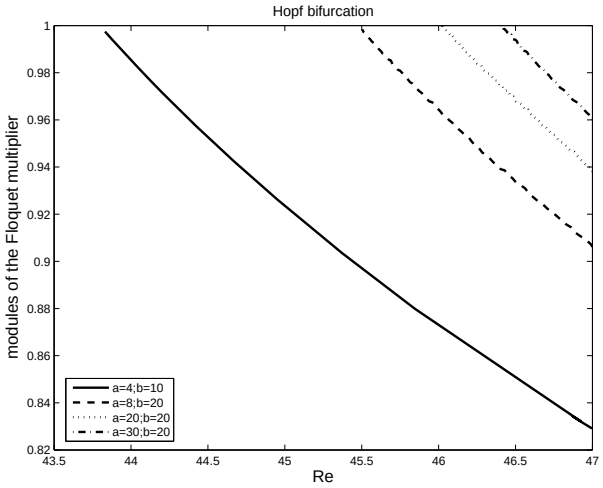


Figure 5.6: *A plot of the dominant Floquet multiplier μ as function of Re. The periodic regime starts where the dominant Floquet multiplier becomes equal to one.*

5.3 Performance

Clustering of the Floquet multipliers

In the Newton-Picard method of PDEcont, we have to choose a radius ρ which defines which Floquet multiplier are considered to be "large" and are included in the dominant subspace. If the number of such vectors is too large, the Newton-Picard method becomes inefficient. The optimal value of ρ is problem dependent. In our flow problem, we varied the ρ at different parts on the branches, since we sometimes had convergence problems. In our flow problem most Floquet multipliers are clustered around zero. This is a good condition for PDEcont. In Fig. 5.7 we see the dominant Floquet multipliers. Since we used $\rho \approx 0.5$, we typically had only about 2 basis vectors.

Computation time

To give an idea of the computation times, we approximated the time needed to compute a steady state or a periodic branch point with different domains. The choice of radius ρ in the Newton-Picard process has a lot of influence on the computation times. These are shown in Tables 5.3 and 5.4. Computing the full branch of steady state solutions on grid nr. 1 (58 points) took about 3.5 days, computing the whole branch of periodic solutions (82 points) about 5 days. So the computations are very time-consuming. A speedup should come from a parallel implementation of the CFD code.

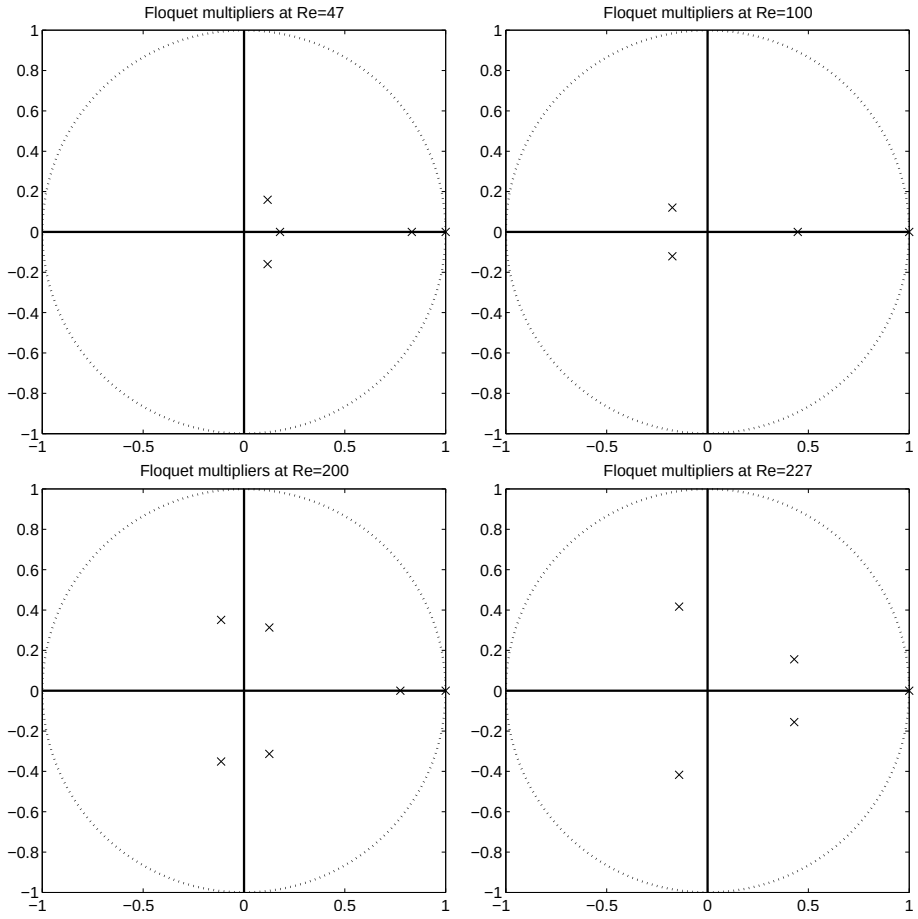


Figure 5.7: *The Floquet multipliers at different values of Reynolds computed by PDEcont.*

domain			grid		Computation time	
nr.	a	b	nx	ny	Int. time (min)	time/point (hours)
1	4	10	100	120	2 min	1 h
2	8	20	150	180	5 min	3 h
3	20	20	200	240	10 min	6 h
4	30	20	200	240	10 min	6 h

Table 5.3: *Approximation of computation times for one time integration and one point on the steady state branch at different domains*

domain			grid		Computation time	
nr.	a	b	nx	ny	Int. time (min)	time/point (hours)
1	4	10	100	120	1.5 min	1.5 h
2	8	20	150	180	4.5 min	4.5 h
3	20	20	200	240	8.5 min	8.5 h
4	30	20	200	240	8.5 min	9.5 h

Table 5.4: *Approximation of computation times for one time integration and one point on the branch of periodic solutions at different domains*

Chapter 6

Evaluation

In this project we had two objectives, namely the numerical bifurcation analysis of our flow problem and finding out if PDEcont does work for our flow problem and with this DNS code. We showed in Chapter 5 that with the combination of PDEcont and DNS it is possible to compute the steady state and periodic branch in a whole range of Reynolds numbers. We will see that there is an interesting event on the branch of periodic solutions which can lead to further research.
We also showed that it is possible to locate the Hopf bifurcation very accurately. The location of the Hopf bifurcation is in good agreement with the result from literature.

6.1 Implementation difficulties and possible weaknesses in the combination

We are very satisfied with the results obtained with the combination of PDEcont and the DNS code. There were no real problems in the implementation of DNS in PDEcont with the method used. However, due to the implementation method we used, it was hard to find errors when something went wrong. The DNS code and PDEcont only communicate with each other by files. In this way it is hard to find possible errors when a time integration fails in the DNS code, because PDEcont has not enough control over the time integration. Probably with the second implementation method, mentioned in Chapter 4, it would have been easier to find such rare errors. This is an inconvenience in the combination of PDEcont and DNS, but it is a consequence of the chosen implementation method.
We mention also three small things which need some extra attention. First we have to notice that since PDEcont uses the output of DNS for further calculations, this output must be accurate enough. We changed the number of digits of the output of DNS to obtain the necessary accuracy for PDEcont. A second issue we encountered was the fact that for starting the DNS code, we also needed the state at the previous time step. The previous state was not computed by PDEcont and thus we copied the current state instead of the previous state. As a result the time integration method will behave like a forward Euler method instead of an Adams-Bashforth method in the first time step. Thirdly, in the current implementation, the simulation code runs in its source directory and all the paths are hard-coded in the system definition files. We need to switch to a more flexible setup to make it possible to run several instances of the code on a single machine, each in its own directory, without having to change the source for each test.

We can conclude that the combination works perfectly, but an other implementation method would probably be more proper.

6.2 Performance

This flow problem is very suitable for PDEcont since the spectrum of the Floquet multipliers is nicely clustered (see Fig. 5.7). For accurate results with the combination of PDEcont and DNS, we need domain nr. 4 in our experiments. This results in very long computation times as we saw in Table 5.3 and 5.4. For this reason we did not compute the steady state and periodic branch in the range of Reynolds 20 up to 300 on this grid.
We did compute these branches in this range on domain nr. 1 to show that we are able to compute branches in a whole range of Reynolds numbers. We computed the branch of steady states very successfully. The branch of periodic solutions has some convergence problems at $Re \approx 227$. At this branch point the Floquet multipliers are still nicely clustered. The problems already occur in the computation of the basis. It is very likely that the finite difference used for the matrix-vector products are not longer accurate enough and we suspect that this is caused by non-smooth behaviour of the DNS code. It is possible that the DNS code is not sufficiently 'smooth'. To continue the branch from $Re \approx 227$ we could try the following:
-Increase the accuracy of the Poisson solver in the DNS code,
-Fix the number of iteration steps in the Poisson solver,
-Use a higher order finite difference method in PDEcont to compute the derivatives of φ.
However, if the failure is caused by too non-smooth behaviour of the DNS code, this will be of little help.
For computing the matrix-vector products and the derivatives of φ with respect to T and Re, a first order finite difference method was sufficient for our flow problem for smaller Reynolds numbers.

6.3 Further research

Notice that in the periodic regime the Floquet multiplier with largest modulus is increasing very fast around $Re \approx 170$. After a local maximum at $Re \approx 200$ the Floquet multiplier decreases in modulus and does not leave the unit circle (see right Fig. 5.4). This event might have something to do with the second instability around $Re \approx 189$.
We used the DNS code in a two dimensional mode to find the steady state and the periodic branch. Unfortunately, due to the two dimensional mode of DNS, we were not able to find the second instability at $Re \approx 189$. There is a possibility to use the DNS code in a three dimensional mode and try to find the second instability as Barkley did in his research [4].

Appendix A

Description of subroutines for data transfer

ReadPar

Synopsis
int **ReadPar**(par_fname_in, sdir)

Description of Parameters
char *par_fname_in: Read the parameters from this file.
char *sdir: The directory of the source file.

Return values
`IO_SUCCESS`, 0 : Successful
`IO_FAILURE`, 1 : Failed

Action
The routine ReadPar() reads the parameter settings from the DNS code such that PDEcont can use the necessary parameter values. The following parameters and their values can be found in the input file:

branching parameter, the Reynolds number	:	`Re`,
time step	:	`dt`,
number of time steps	:	`nt`,
time integration parameter	:	`nsf`.

WritePar

Synopsis
int **WritePar**(par_fname_in, par_fname_out, sdir, ddir)

Description of Parameters
char *par_fname_in: File name to copy the unchanged parameters from.
char *par_fname_out: The file name with the new parameters.
char *sdir: The directory of the source file.

char *ddir: The destination directory of the new file.

Return values

`IO_SUCCESS`, 0 : Successful
`IO_FAILURE`, 1 : Failed

Action

The routine WritePar() writes the new parameter values to a given file name in the destination directory. This new can be read by the DNS code. This routine reads the parameters from the input file. The unchanged parameter values are copied from the input file to the new file. The following parameter values can be updated in a new file:

branching parameter, Reynolds number	:	`Re`,
time step	:	`dt`,
number of time steps	:	`nt`,
time integration parameter	:	`nsf`.

ReadState

Synopsis

int **ReadState**(state_fname_in, state, state_o, sdir)

Description of Parameters

char *state_fname_in : File to read the state vector from.
double *state: Pointer to state vector.
double *state_o: Pointer to old state, one time step before state.
char *sdir: The directory of the sourse file.

Return values

`IO_SUCCESS`, 0 : Successful
`IO_FAILURE`, 1 : Failed

Action

The routine ReadState() reads the state (u, v, w, p) from fldxx.dat. It converts the vectors u and v into one vector $(u(1), v(1), u(2), v(2), \cdots, u(total), v(total))$, where `total` is the total number of grid points. The column vector is stored in the variable `state`. The w and p components in the `fld##.dat` file are not stored in memory.The previous time step is stored in the variable `state_o`. The routine has to know the total number of discritization points. The total length of a column vector in the standard DNS state input file `fld##.dat` is 3 times `total`. It contains 3 identical 2D vector fields (u, v, w, p). Of course only one is read.

WriteState

Synopsis

int **WriteState**(state_fname_out, state, state_o, ddir)

Description of Parameters

char *state_fname_out : Write the state vector to this file.
double *state: Pointer to state vector.
double *state_o: Pointer to old state, one time step before state.
char *ddir: The destination directory of the file.

Return values

`IO_SUCCESS`, 0 : Successful
`IO_FAILURE`, 1 : Failed

Action

The routine WriteState() writes the known state in memory $[u(1), v(1), u(2), v(2), \cdots]$ to a new file. The format will be the standard DNS input format or to an output file of PDEcont. The two columns w and p are added as the third and fourth column in the new file. The w and p components are fixed to zero. In the case for writing a DNS input file, the current state will be written two times to the file, since there is no old state.

Bibliography

[1] K. Lust. *Numerical bifurcation analysis of periodic solutions of partial differential equations*. PhD thesis, Katholieke Universiteit Leuven, 1997.

[2] R. W. C. P. Verstappen and A. E. P. Veldman. Symmetry-preserving discretization of turbulent flow. *Journal of Computational Physics*, 187:343–368, 2003.

[3] C. P. Jackson. A finite-element study of the onset of vortex shedding in flow past variously shapes bodies. *J. Fluid Mech.*, 182:23–45, 1987.

[4] D. Barkley. Confined three-dimensional stability analysis of the cylinder wake. *Physical Review*, E 71:1–3, 2005.

[5] E. J. Doedel, A. R. Champneys, T. J. Fairgrieve, Y. A. Kuznetsov, B. Sandstede, and X. J. Wang. *AUTO97: Continuation and bifurcation software for ordinary differential equations.*, 1997. Technical report.

[6] A. I Khibnik, Yu. A Kuznetsov, V. V. Levitin, and E. N. Nikolaev. Locbif, version 2: Interactive local bifurcation analyzer. *CAN Expertise Centre*, 1992. Amsterdam.

[7] Yu. A. Kuznetsov and V. V. Levitin. *CONTENT, a multiplatform continuation environment.*, 1996. Technical report.

[8] R. E. Bank. *PLTMG, a software package for solving elliptic partial differential equations: users guide 7.0*. Frontiers in Applied Mathematics. SIAM, 15 edition, 1994.

[9] D. Barkley and R.D. Henderson. Three-dimensional floquet stability analysis of the wake of a circular cylinder. *J. Fluid Mech.*, 322:215–241, 1996.

[10] R. D. Henderson and G. E. Karniadakis. Unstructured spectral element methods for simulation of turbulent flows. *J. Comp. Phys.*, 122:191–217, 1995.

[11] M. F. Schatz, D. Barkley, and H. L. Swinney. Instability in spatially periodic open flow. *Physics of Fluids*, 7:344–358, 1996.

[12] R. D. Henderson and D. Barkley. Secondary instability in the wake of a circular cylinder. *Physics of Fluids*, 8:1683–1685, 1996.

[13] O. Posdziech and R. Grundmann. Numerical simulation of the flow around an infinite long circular cylinder in the transition regime. *Springer-Verslag*, 2001.

[14] H. Zhang, U. Fey, and B. R. Noack. On the transition of the cylinder wake. *Physics of Fluids*, 7:779–794, 1995.

[15] B. R. Noack, M. König, and H. Eckelmann. Three-dimensional stability analysis of the periodic flow around a circular cylinder. *Physics of Fluids*, A5:1279–1281, 1993.

[16] R. Seydel. *Practical Bifurcation and Stability Analyses, From Equilibrium to Chaos.* Springer-Verlag, 1994.

[17] M. Provensal, C. Mathis, and L. Boyer. Bénard-von Kármán instability: transient and forced regimes. *J. Fluid Mech.*, 182:1–22, 1987.

[18] M. Hammache and M. Gharib. An experimental study of the parallel and oblique vortex shedding from circular cylinders. *J. Fluid Mech.*, 232:567–590, 1991.

[19] C. H. K. Williamson. Defining a universal and continuous Strouhal-Reynolds number relationship for the laminar vortex shedding of a circular cylinder. *Physics of Fluids*, 31:2742–2744, October 1988.

[20] C. H. K Williamson. Oblique and parallel modes of vortex shedding in the wake of a circular cylinder at low reynolds numbers. *J. Fluid Mech.*, 206:579–627, 1989.

[21] F. W. Wubs. Computational methods of science. Rijksuniversiteit Groningen, 2003.

[22] K. Lust. *PDEcont, a timestepper-based continuation code for large-scale systems.*, 1997. Technical report.