A three-dimensional numerical model of a horizontal axis, energy extracting turbine

An implementation on a parallel computing system

Angus C.W. Creech

Submitted for the degree of Doctor of Philosophy

Heriot-Watt University, Scotland

School of Engineering and Physical Sciences

March 2009

Abstract

In the last decade, there has been a resurgence of interest in tidal power as a renewable, and environmentally friendly source of electricity. Scotland is well placed in this regard, as the currents in the surrounding seas are primarily tidal; that is to say, driven by lunar and solar tides.

Investigations into tidal streams as an energy source, their viability in particular locales, the efficient organisation of marine turbine farms, and most importantly, the effect of such farms on the environment, demand the use of computational fluid dynamics for effective modelling. They also require a turbine model sophisticated enough to generate realistic power output and wakes for a variety of flow conditions, yet simple enough to simulate a number of turbines on modest computing resources.

What is presented here then, is the justification for such a model, the development and deployment of it during my PhD, and my validation of the model in a variety of environments.

Acknowledgements

I would like to thank my supervisor, Dr Wolf-Gerrit Früh, for his assistance during my studies, and the Applied Modelling Computional Group of Imperial College, London, for supplying Fluidity, as well as helping me to use it. My office colleagues also deserve a mention, for being good company and sources of valuable discussion. Lastly but not least, I must mention the three most important people – my parents, who indulged my curiosity in how things work from an early age; and Vanessa, without whose support none of this would have been possible.

Author's Declaration

I declare that the work in this dissertation was carried out in accordance with the Regulations of Heriot-Watt University

Signed: . . . .

Date: . . . .

Contents

Chapter 1

Introduction

In this chapter, after introducing the main aims of the thesis and an outline of the structure, we shall briefly cover the origins of tidal currents, before moving on to describing tidal power generation. Following on from this is a discussion of various computational fluid dynamics (CFD) techniques.

1.1. Thesis overview

This thesis covers the design, development and validation of a computational model for axial-flow wind and marine (tidal) turbines. Its purpose is not only to show how the performance of such turbines can be assessed in realistic flow conditions using modest computing resources, but also to provide a means for studying the wake structures behind them. Wake recovery is known to be an issue in wind farms, but – as this thesis will show – may be even more of a problem in tidal turbine farms.

A background to marine energy concerns and computational fluid dynamics (CFD) is given in chapter 1; this is then followed by the validation of the CFD program chosen in chapter 2. Briefly, alternative pre-existing axial-flow models are covered in chapter 3, which leads into the full mathematical presentation of the turbine model developed in chapter 4. Chapter 5 deals with the software development, particularly parallelism issues; chapters 6 & 7 then describe the experiments and results in detail, which chapter 8 discusses. Chapter 9 forms conclusions from these results, and suggests future work.

1.2. Tidal currents

Marine energy schemes have requirements regarding the types of oceanic currents can be exploited. Tidal power schemes demand a flow strength of around $2 - 3\,\mathrm{m/s}$ to generate a worthwhile amount of power, and these must be in waters of tens of metres deep. This rules out wind-driven Langmuir cells, and consequently convergence/divergence zones. Instead, we must restrict ourselves to tidal currents generated by the sun and the moon.

1.2.1. Newton's theory of lunar tides

Newton was the first to arrive at a satisfactory explanation of tide generation: this is known as the Equilibrium Theory of Tides. Below is a brief account, adapted from Bryden [14].

In a simplified model, the Earth-moon system rotates around a common centre of mass CM_{system}, as shown in fig. 1.1. This is off-centre from the Earth's centre of mass, CM_{Earth}, by a distance r in the same orbital plane; similarly, the Moon's centre of mass CM_{Moon} orbits around CM_{system} in a circular path at a radius of R_{Moon} (see fig. 1.1). Since the moon's gravity at point B will be weaker than the centrifugal acceleration due to the rotation of the Earth about CM_{system}, this will result in a tidal bulge at B. Newton's equilibrium theory dictates that there will be two tidal bulges, with the second one at A.

These two tidal bulges follow the moon as it orbits around CM_{system} with a periodicity of 27.3 days; the Earth rotates about CM_{system} in the same direction as the moon orbits, which gives a tidal cycle of 12 hours 25 minutes; the moon also has an elliptical orbit around the Earth, causing tidal forces to vary by 40%. Furthermore, additional monthly variations occur due to the inclination of the Moon's orbit, which is 28^o with respect to the Earth's orbital plane, shown in fig. 1.2.

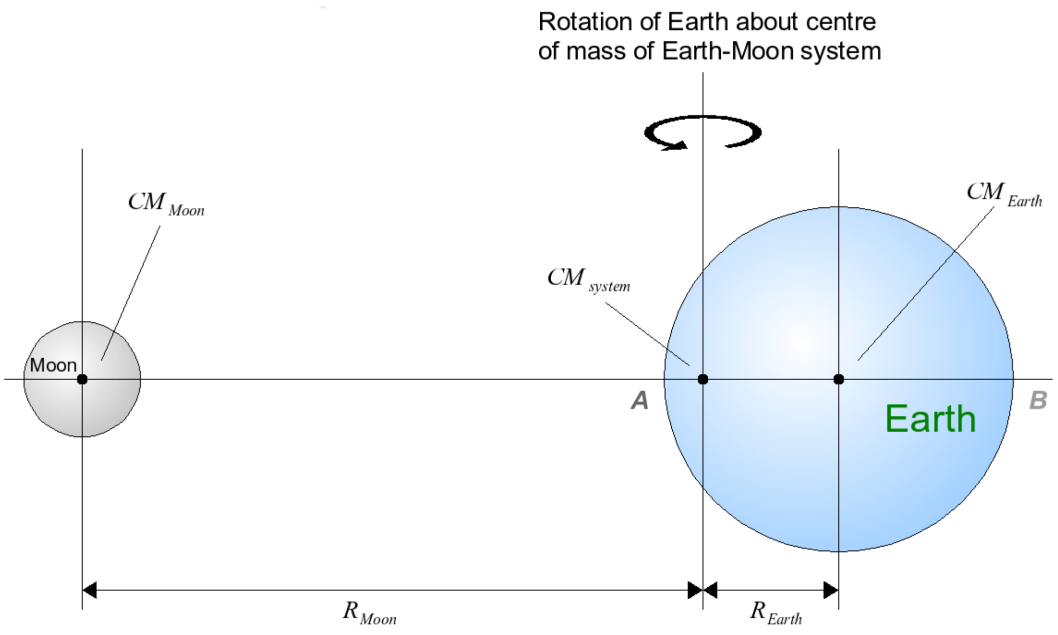


Figure 1.1: Rotation of Earth-Moon system

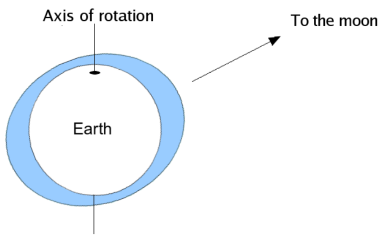


Figure 1.2: Consequences of Moon's elliptical orbit

1.2.2. Solar tides

Despite the Sun's mass, the solar influence on tides is roughly half that of the Moon's – however, the combination of the two gives rise to monthly fluctuations: the sun's tide is semi-diurnal, with a periodicity of precisely 12 hours. The Earth's orbit around the Sun is almost circular in comparison with Moon's orbit around the Earth; the distance varies by about 4%. The highest tides, ie. the spring tides occur when both the Sun and the Moon are exactly in phase, and the neap (lowest) tides occur when they are out of phase by $\frac{\pi}{2}$ or $\frac{3\pi}{2}$.

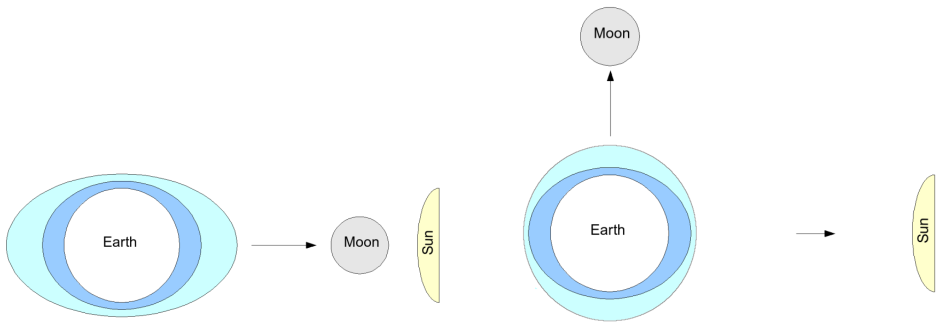


Figure 1.3: In (a) we can see the spring tidal bulge, occuring when the Moon and Sun are in alignment; the dark blue ring represents solar tide, and the lighter blue the tide with the lunar component added. The neap tides are shown in (b), occurring when the Sun and Moon are out of phase by $\frac{\pi}{2}$ or $\frac{3\pi}{2}$

1.2.3. The Coriolis effect

The Coriolis Effect is a consequence of the conservation of angular momentum, and the Earth's rotation. In the Northern Hemisphere, as a parcel of water travels northwards or southwards it will veer to the right; in the Southern Hemisphere this is mirrored, to the left.

The Coriolis force is, along with bathymetry, a key component in the formation of gyres – large scale semi-closed currents which inhabit ocean basins such as the North Atlantic's. These will explained further in section 1.2.4.

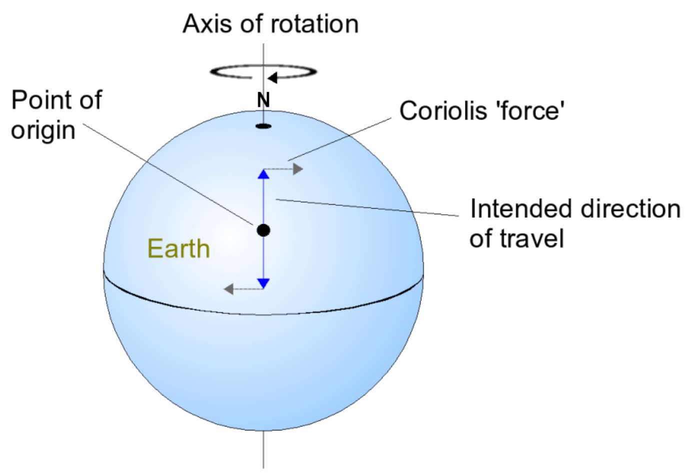


Figure 1.4: How the Coriolis effect deviates the path of bodies in the Northern hemisphere

1.2.4. Factors affecting tidal streams

Tidal currents, or streams, are a consequence of several influences, which will be explained below.

Gyres and tidal ranges

Gyres are produced through a combination of tidal forces, the Coriolis effect, topography and bathymetry. Ocean basins, such the North Atlantic basin, are semi-closed circulatory systems, which underpin the gyre. Due to Coriolis ‘forces’, any water travelling either northward or southward is diverted to the right, and this generates a rotating tidal wave which sweeps round the basin. This can give rise to large tidal ranges, such as those in the Bay of Fundy [31] and the Lofoten Islands in Norway [32]. A more local example would that of the semi-diurnal M_2 tide that passes from the North Atlantic, around the Scottish coast and into the Artic-Barents basin [33].

According to Bryden [14] and Lemonis [44], areas that exhibit large tidal ranges are:

- **England** - Bristol Channel; Severn Estuary; Alderney Straits, Channel Islands

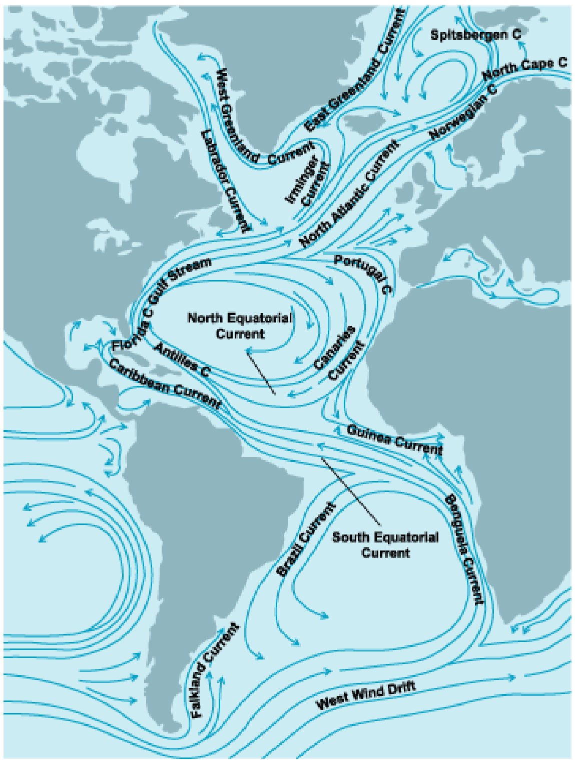


Figure 1.5: Gyres in the Atlantic and Artic oceans, courtesy of Advanced Atlas of Modern Geography [12]

- **Scotland** – Pentland Firth; Shetland Isles; Solway Firth
- **France** – La Rance, Brittany
- **Italy** – Straits of Messina
- **Southeast Asia** – Singapore
- **China** – Jangxia Creeck, East China Sea
- **Japan** – Naruto Strait
- **Russia** – Bay of Kislaya
- **Canada** - Bay of Fundy
- **Western Australia** – West Kimberley

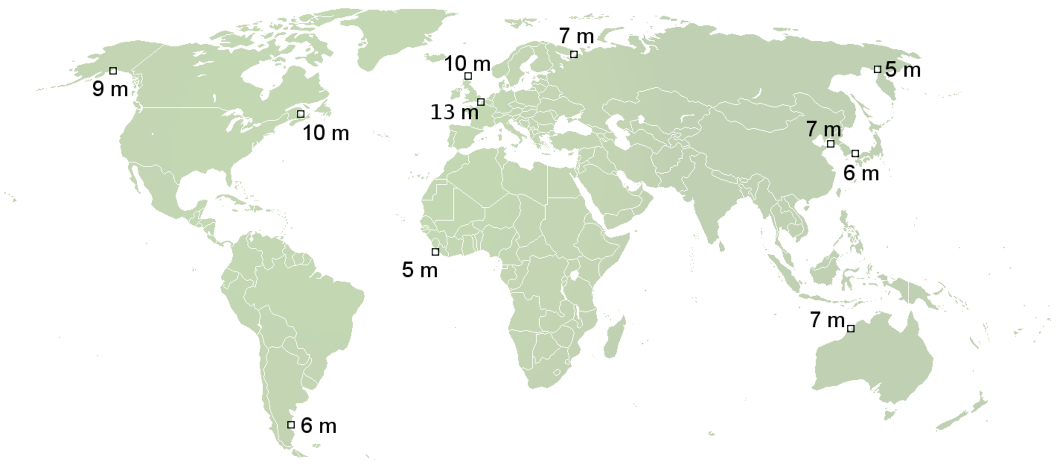


Figure 1.6: Tidal ranges around the World in metres.

These large tidal ranges raise the sea level above equilibrium and generate gravitational potential energy; this potential energy is then converted into kinetic energy in the form of tidal streams, as the sea attempts to return to its equilibrium state.

Bathymetry and topography

Topographic features such as islands, headlands etc. can have a drastic effect on tidal current flow. A famous example of this in Scotland would be the Gulf of the Corryvreckan, a hazardous strait with a strong tidal stream which runs between the Islands of Jura and Scarba, off the West Coast of Scotland: the channelling of the tidal currents through a relatively narrow strait can cause flow speeds in excess of 5 m/s. The bathymetry of the local ocean floor also has a profound effect on flow too; the turbulent whirlpools of the Corryvreckan are thought to be magnified in their effect by a ridge of submerged rock that runs out from Scarba to the north. Other examples of topographically-enhanced flow would be the Pentland Firth, between the northern tip of Scotland and the Orkney Islands; the tidal resonance that occurs in the Bay of Fundy, North America; and the Moskstraumen whirlpools - more famously known as The Maelstrom. A more detailed, qualitative analysis of the role topography plays in creating the Moskstraumen is given by Gjevik, Moe and Ommundsen [32].

1.3. Tidal power

The kinetic energy associated with tidal currents is vast – current estimates [44] are that globally 26 PWh of energy could feasibly be extracted from the world's oceans each year, with a third of that in shallow-water areas. This suggested value represents approximately 10% of the World's energy demands, a significant amount, whose importance should increase in light of the Kyoto Protocol.

In Europe, two companies, IT Power Ltd and Tecnomare SpA examined 106 sites in around Europe, which could supply 105 TWh/year. Scotland, England and Wales have been estimated to have a potential energy output of 33.5 TWh/year according to [25], but recent research suggests that this figure is too low [14], with 270 TWh/year being more realistic, and that even the Pentland Firth alone, with currents in excess of 7 m/s, may have a potential for 140 TWh/year.

One of the reasons for these large figures is that even at mild flow speeds (2 m/s) compared to those that wind turbines experience, the density of seawater (approximately 1027 kg/m^3), about 850 times that of air, implies that tidal currents can harbour a considerable amount of kinetic energy.

There can be little doubt of the importance of tidal energy as a supply of electricity; however the technology of tidal stream energy extraction, especially in open waters, is not yet fully formed or complete, and is still being researched vigorously. Consequently a variety of designs have flourished. The more successful designs are detailed below.

1.3.1. The power equation

The power associated with a moving fluid can be stated as

$$PW = \frac{1}{2}\rho A\overline{u}^3 \tag{1.1}$$

Where is ρ is the density of the fluid (in this case seawater), A is the cross-sectional area through which the fluid passes, and $\overline{u}$ is the mean speed of the flow.

As a crude estimate, a tidal channel 1000 m wide and 40 m deep with a sustained current of 3 m/s would give a power of roughly 530 MW; however no tidal power devices operate at 100% efficiency, so the amount of extractable power is

$$PW = c_P \frac{1}{2} \rho A_D \overline{u}^3 \tag{1.2}$$

Where c_P is the power extraction coefficient, and A_D is the cross-sectional area of the device (or devices) exposed to the flow. However, detailed assessment of tidal sites is difficult, since c_P varies with both the device design and local flow conditions. This interdependent relationship is likely to be highly non-linear due to its origin in fluid dynamics, and so perhaps not suited to analyses such as [15].

1.3.2. Power extraction devices

Horizontal axis turbines

Horizontal axis turbines have the propeller perpendicular to the flow, which allows the tidal current to drive the rotor, and thus an electric generator. One of the most successful of these is SeaFlow from Marine Current Turbines Ltd. (see fig. 1.7), who have a prototype operating near Lynmouth, England [48]. It is built on a monopile embedded in the sea floor, with a maintenance tower above the surface; the turbine mount slides up towards the surface for repairs or in heavy weather. Marine Current Turbines estimate that the SeaFlow device generates around 300 kW at a peak flow of 2.7 m/s.

A limitation of this design is that it is restricted to relatively shallow waters, approximately 40 metres deep. Alternatives have been suggested for deeper waters, such as semi-buoyant devices secured to the seabed via moorings. However there are concerns about safety particularly about the stress, mainly through buoyancy thrust and wave motion, that these moorings would be subjected to [30].

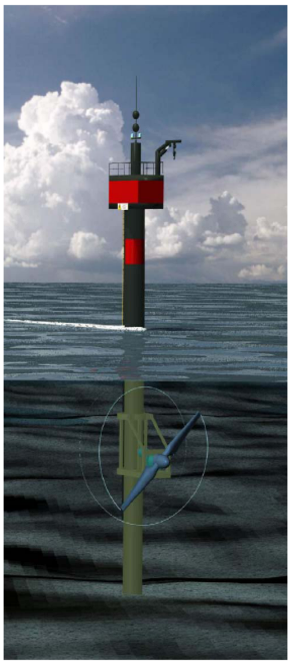


Figure 1.7: The SeaFlow horizontal axis turbine (courtesy of Marine Current Turbines Ltd.)

Vertical axis turbines

The propeller blades in vertical axis turbines rotate in a manner as shown in fig. 1.8. There is only one turbine known of this type – the ENERMAR device developed by Ponte di Archimede nello Stretto di messina SpA in Italy. This turbine consists of a specially designed vertical axis rotor named Kobold which is connected to an electrical generator, housed in a floating platform 10 meters in diameter. One key feature of this device is that the direction of rotation is independent of the direction of the current, which is unique in marine energy to the Kobold rotor.

A prototype was deployed 150m offshore in the Messina Straits, Italy, in 2002, at a depth of 20m. With a current of 1.8m/s, 20 kW of electrical power was generated, giving an estimated energy conversion efficiency of 23%; at higher speeds approximately 3 m/s, 150 kW is expected according to Lemonis [44].

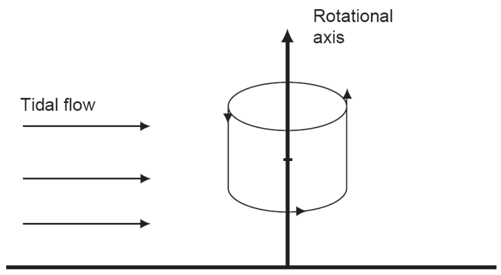


Figure 1.8: Current entering a vertical turbine

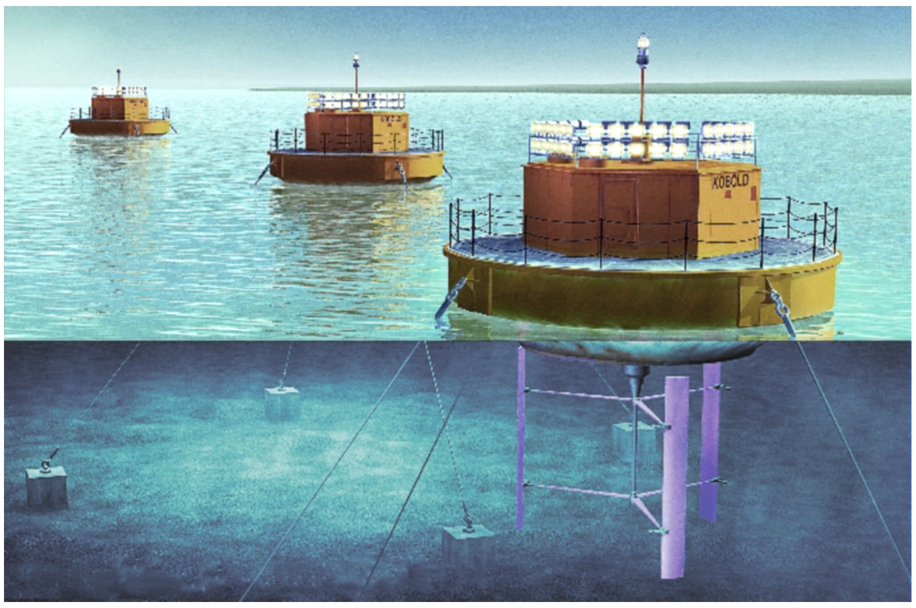


Figure 1.9: Artist's impression of an ENERMAR Kobold turbine farm (courtesy of ImageFactory/Quark). The turbine blades are shown in light purple.

Hydrofoils

The only hydrofoil currently being developed is the Stingray hydrofoil (see fig. 1.10) manufactured by Engineering Business Ltd. The broad yellow strip is a hydrodynamic surface, which generates lift. The hydroplanes oscillate up and down, as the angle is altered to generate lift or down-force. The hydroplane arm pumps high-pressure oil to drive a hydraulic motor, which then drives an electric generator.

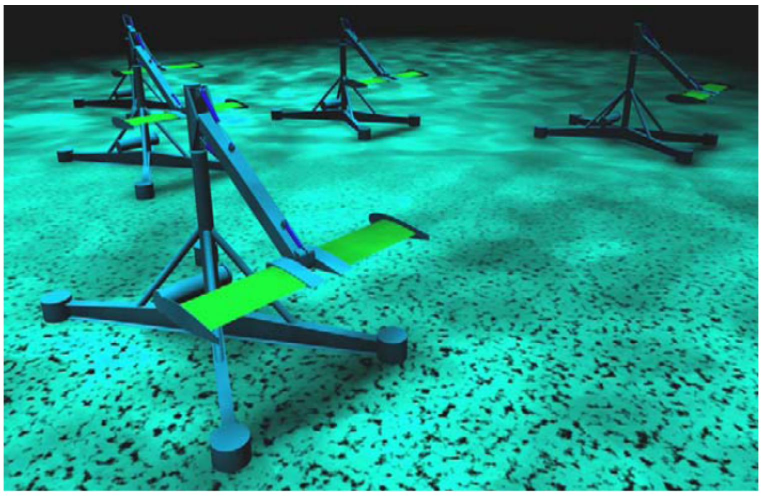


Figure 1.10: Stingray hydrofoil (courtesy of Lemonis).

In 2002, a full-size prototype with a rated power output of 150 kW was deployed in Scotland at a depth of 35 metres in Yell Sound, Shetland.

1.4. General fluid dynamics equations

The basic equations that govern the fluid are described here, limited to incompressible Navier-Stokes flow with the Boussinesq approximation.

1.4.1. The momentum equation

If we assume a three-dimensional Cartesian co-ordinate system in the rotating reference frame of the Earth, such that the axis of x_1 points eastward, the axis x_2 points northward, and the x_3 axis points upwards towards the zenith, and that the corresponding velocity components are $u_{1,}$, u_2 and u_3 respectively, then the momentum equations (from [42]) can be written in tensor notation as

$$\rho\frac{\partial u_i}{\partial t} + \rho u_j\frac{\partial u_i}{\partial x_j} + 2\rho\epsilon_{ijk}\Omega_j u_k = -\frac{\partial p}{\partial x_i} - g\rho\delta_{3i} + \frac{\partial \sigma_{ij}}{\partial x_j} \tag{1.3}$$

Where:

- ϵ_{ijk} is equal to

  1 if i, j and k are in cyclic order

  -1 if they are anticyclic

  0 if two or more axis are of identical value

- t is time
- p is pressure
- ρ is the density of the fluid
- Ω_j is the Earth's angular velocity
- g is the Earth's gravitational acceleration,
- δ_{3i} is a Kronecker delta
- σ_{ij} is the stress tensor

If the centrifugal/centripetal term is absorbed into $\underline{g}$, and assuming Newtonian fluid stress, this can be rewritten as:

$$\rho\frac{\partial \underline{u}}{\partial t} + \rho(\underline{u}.\nabla) + 2\underline{\Omega} \times \underline{u} = -\nabla p - \underline{g}p + \mu\nabla^2\underline{u} \tag{1.4}$$

Which should be a more familiar form of the Navier-Stokes momentum equation in a rotating frame.

1.4.2. The continuity equation

The first continuity equation guarantees the conservation of mass

$\frac{\partial\rho}{\partial t} + \frac{\partial(\rho u_i)}{\partial x_i} = 0$ or more commonly $\frac{\partial\rho}{\partial t} + \nabla.\underline{u} = 0$

Generally speaking, for properties of the fluid such as temperature and salinity, denoted by Φ:

$$\frac{\partial(\rho\Phi)}{\partial t} + \frac{\partial(\rho\Phi u_i)}{\partial x_i} = -\frac{\partial F_i}{\partial x_i} - q \tag{1.5}$$

Where F_i represents the flux, and q the internal generation.

This can be expressed as:

$$\frac{\partial(\rho\Phi)}{\partial t} + \nabla.(\rho\Phi\underline{u}) = \nabla.\underline{F} - q \tag{1.6}$$

This is a more general form of mass continuity equation.

1.4.3. Boundary conditions

Boundary conditions define the conditions at the edge of the solution space, and are usually defined prior to any simulation or model being run; they come in a variety of flavours.

Solid boundaries

Solid boundaries have, usually, one condition: the no-slip boundary condition. That is to say, the fluid velocity $\underline{u} = 0$ at every point on this boundary. This is not always so, especially in two-dimensional vertically-integrated models which may employ free-slip sea floor boundaries. As no-slip proscribes a fixed value at the boundary, it is also a Dirchlet boundary condition.

Open boundaries

Open, or tidal, boundaries can take several forms. They could be Dirchlet – that is to say, have fixed flux values. They could also be Von Neumann boundaries, where the velocity gradient is set to a constant, ie.

$$\frac{\partial u}{\partial x_n} = f \tag{1.7}$$

where

- u is the magnitude of $\underline{u}$
- x_n is a physical coordinate whose axis is normal to the boundary
- f is a constant (not the Coriolis parameter, as it is denoted later in this chapter)

Open boundaries can also be a combination of Dirchlet and Von Neumann: these are called mixed or Robin conditions. In this case, the boundary would be described as:

$$\frac{\partial u}{\partial x_n} + ku = f \tag{1.8}$$

where additionally, k is another constant. This is often described as a 'forcing' boundary condition.

1.5. Computational fluid dynamics modelling

In physical oceanographic modelling, the problem posed is often analytically intractable and requires numerical simulation to provide any meaningful solution; indeed, situations describing unsteady flow are not uncommon.

Two of the main numerical approaches were evaluated for suitability: finite difference and finite element. The basics of these techniques will be explained in general, discipline non-specific terms, laying the foundations for the following chapter which will cover each type of model in more detail.

A third set of techniques, finite volume methods, are quite similar to finite difference methods. They work by evaluating the fluxes between small, discrete volumes within the fluid, and so are by definition conserve mass; however, unlike finite difference methods they can also support unstructured mesh geometries. As will be shown later, finite element methods have similar advantages.

1.5.1. Finite difference

With finite difference fluid modelling, a discrete grid is created over the solution space, and the continuous partial differential equations (eg. the momentum and continuity equations) are replaced by finite difference expressions. An algorithm is constructed which uses these finite difference expressions to solve variables such as the fluid velocity $\underline{u}$ and pressure p, although fluid properties such as salinity and temperature may also be included here. Von Neumann and Dirichlet boundary conditions are usually defined at the edges of this solution space, but they can also be defined within it.

Finite difference methods have a long history in computational fluid dynamics; one example shall be taken to demonstrate finite difference techniques.

The SIMPLE algorithms: formulation

Introduced by Patankar and Spalding in 1972 [62], these finite volume algorithms are based on a staggered grid of the momentum and continuity equations. In figure 1.11 we can see a two-dimensional example of this grid, with pressure defined in between points where the velocity is defined:

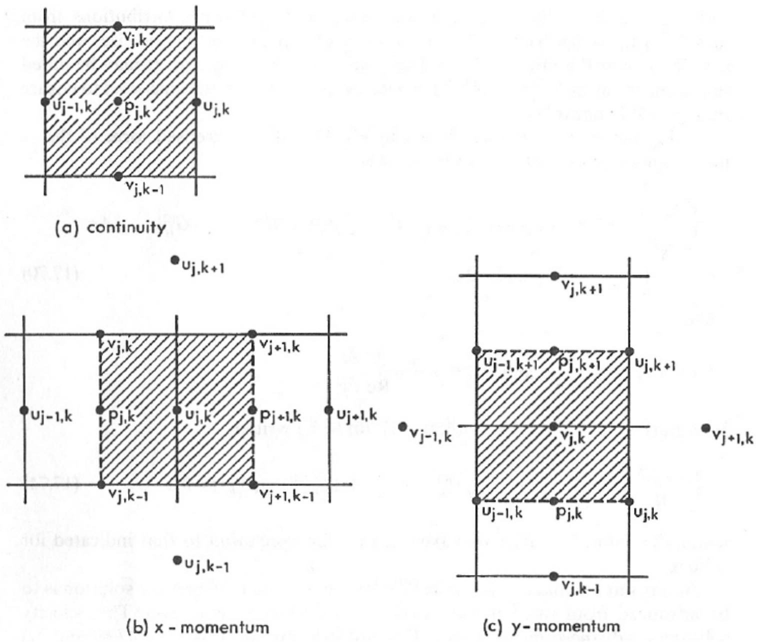


Figure 1.11: A two-dimensional finite difference grid (courtesy of Fletcher).

The SIMPLE algorithms semi-implicitly link the pressure and continuity terms to iteratively solve the equations, hence the name, SIMPLE – Semi-Implicit Method for Pressure-Linked Equations. They can solve for steady and unsteady flow.

We take the non-dimensional equations for two-dimensional incompressible laminar flow as described in Fletcher [26]:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{1.9}$$

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial y} + \frac{\partial u^2}{\partial x} + \frac{\partial}{\partial y}(uv) + \frac{\partial p}{\partial x} = \frac{1}{Re}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \tag{1.10}$$

$$\frac{\partial v}{\partial t} + \frac{\partial}{\partial x}(uv) + \frac{\partial v^2}{\partial y} + \frac{\partial p}{\partial y} = \frac{1}{Re}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) \tag{1.11}$$

Where Re is defined as $\frac{\overline{u}L}{\nu}$ (with $\overline{u}$ the mean speed of the flow, L the length of the problem, and ν the kinematic viscosity of the fluid), whilst u and v are the x and y components of the fluid's velocity.

If (j, k) represents a particular point on a grid representing the discretised solution space, and n denotes the n^{th} time step, then we can rewrite the discrete continuity equation as

$$(u^{n+1}_{j,k} - u^{n+1}_{j-1,k})\Delta y + (v^{n+1}_{j,k} + v^{n+1}_{j,k-1})\Delta x = 0 \tag{1.12}$$

where Δx and Δy represent the grid step size in the x and y directions respectively.

Fletcher shows how the momentum equations can be derived, however here we will simply state the x-momentum equation as

$$(\frac{\Delta x \Delta y}{\Delta t} + a^u_{j,k})u^{n+1}_{j,k} + \sum a^u_{nb}u^{n+1}_{nb} + b^u + \Delta y(p^{n+1}_{j+1,k} - p^{n+1}_{j,k}) = 0 \tag{1.13}$$

and the y-momentum equation as

$$(\frac{\Delta x \Delta y}{\Delta t} + a^v_{j,k})v^{n+1}_{j,k} + \sum a^v_{nb}v^{n+1}_{nb} + b^v + \Delta x(p^{n+1}_{j,k+1} - p^{n+1}_{j,k}) = 0 \tag{1.14}$$

where

- Δt is the iteration time-step size in the simulation
- $\sum a^u_{nb} u^{n+1}_{j,k}$ and $\sum a^v_{nb} v^{n+1}_{j,k}$ represent all the convection and diffusion contributions from adjacent grid nodes for $u_{j,k}$ and $v_{j,k}$ respectively
- coefficients $a^u_{j,k}$ and $a^v_{j,k}$ depend on the grid sizes and the solution of u and v at the n^{th} time step (see appendix A.1.6).
- $b^u = (\frac{-\Delta x \Delta y}{\Delta t}) u^n_{j,k}$; similar for b^v.

The main problem here is that the pressure terms and the velocity terms are coupled. The SIMPLE algorithm links the two together. Firstly, the pressure values on the grid are initialised to some value, usually to zero before the first first time-step; then, the values for the velocity and pressure are iteratively updated - see Fletcher [26] p362-365 for more details.

The steps of the iteration are:

1. Calculation of the guess for the correct value for the velocity, $\underline{u}^*$, from the following equations:

$$u^*_{j,k} = \frac{-1}{(\frac{\Delta x \Delta y}{\Delta t} + a^u_{j,k})} . [\sum a^u_{nb} u^*_{nb} + b^u + \Delta y (p^n_{j+1,k} - p^n_{j,k})] \tag{1.15}$$

$$v^*_{j,k} = \frac{-1}{(\frac{\Delta x \Delta y}{\Delta t} + a^v_{j,k})} . [\sum a^v_{nb} v^*_{nb} + b^u + \Delta y (p^n_{j,k+1} - p^n_{j,k})] \tag{1.16}$$

2. Obtain the correction to the pressure, δp, from the simultaneous set of equations

$$a^p_{j,k} \delta p_{j,k} = \sum a^p_{nb} \delta p_{nb} + b^p \tag{1.17}$$

where $b^p = -(u^*_{j,k} - u^*_{j-1,k})\Delta y - (v^*_{j,k} - v^*_{j,k-1})\Delta x$

3. Calculation of the velocity correction, $\underline{u}_c$, from the equations

$$u^c_{j,k} = d_{j,k}(\delta p_{j,k} - \delta p_{j+1,k}) \tag{1.18}$$

where $d_{j,k} = \frac{E \Delta y}{(1+E) a^u_{j,k}}$ and $E = \frac{\Delta t . a^u_{j,k}}{\Delta x \Delta y}$ with a similar definition for $v^c_{j,k}$.

4. Update pressure via

$$p^{n+1} = p^n + \alpha_p \delta p \tag{1.19}$$

where α_p is the relaxation parameter.

5. Set

$$\underline{u}^{n+1} = \underline{u}^* + \underline{u}^c \tag{1.20}$$

ready for the next iteration (where $\underline{u}^{n+1}$ effectively becomes $\underline{u}^n$.

6. Go to step 1, unless the flow (and thus the pressure) has converged, eg. steady flow has been achieved, or the maximum number of time-steps have elapsed. Regarding relaxation parameters, the SIMPLE algorithm has E (equivalent to Δt) and α_p. According to [26], empirical evidence from Patanker (1980) suggests that rapid convergence occurs when $E = 1$ and $\alpha_p = 0.8$.

For familiarisation purposes, a small finite difference solver based upon the SIMPLE algorithm was implemented in the C programming language. Details of its construction, along with source code, are in appendix A.1.

Adaptations of SIMPLE: SIMPLEC and SIMPLER

SIMPLEC The SIMPLE algorithms tend to be slow to converge; Fletcher [26] mentions Raithby and Schneider developing a more efficient algorithm. They suggested that if $E \approx 4$ and α_p is redefined as:

$$\alpha_p = \frac{1}{1+E} \tag{1.21}$$

Then faster convergence is achieved. This as known as a Consistent SIMPLE algorithm, or SIMPLEC.

SIMPLER Also from the same book is Patankar's revised SIMPLE algorithm, SIMPLER, to answer the criticism of SIMPLE that the δp term was often ineffective at converging the pressure field.

It can be shown that

$$a^p_{j,k}\delta p_{j,k} = \sum a^p_{nb}\delta p_{nb} + b^p \tag{1.22}$$

can be rewritten as a discretised Poisson equation, namely:

$$\nabla^2_d \delta p = \frac{1}{\Delta t}\nabla_d.\underline{u}^* \tag{1.23}$$

Other changes from the SIMPLE algorithms are:

1. A velocity $\hat{\underline{u}}$ is calculated in a similar manner to $\underline{u}^*$, in SIMPLE, but with the pressure terms removed.

2. The Poisson equation above is adapted to become a Poisson equation for p^{n+1}, and exchanging $\hat{\underline{u}}$ for $\underline{u}^*$ in b^p, p^{n+1} can be calculated.

3. The values for p^{n+1} are then used to calculate $\underline{u}^c$, which then gives us δp and thus $\underline{u}^{n+1} = \underline{u}^n + \underline{u}^c$.

Whilst this is more computationally expensive than the SIMPLE algorithm alone, it affords convergence in fewer iterations than its forebear.

1.5.2. Finite element

Finite element modelling adopts the line that finite difference and finite volume techniques take, ie. carving up the solution space into finite chunks to evaluate as discrete mesh, however its approach is altogether different.

With finite element techniques, the elements can be of arbitrary shape, such as triangles or quadrilaterals (2D) or tetrahedra (3D). Moreover, the meshes from which the elements are constructed do not have to be regularly spaced, as they are with finite difference: this allows data-dense clusters of points (and correspondingly small elements) at areas of interest, and larger elements with low data-density in less important or varying parts of the solution space. This, importantly, allows the concentration of computational resources where it is needed. Based upon Chung [18] S5.6.2, this section will serve as a rough overview of finite element analysis as applied to fluid dynamics. Index notation is used throughout this section.

Suppose that we can write an approximation of the velocity $\underline{u}$ and pressure p as a series of basis functions:

$$u_i(\underline{x},t) = \Phi_N(\underline{x}) u_{Ni}(t) \tag{1.24}$$

$$p(\underline{x},t) = \Psi_N(\underline{x}) p_N(t) \tag{1.25}$$

And the continuity equation as

$$\begin{aligned} u_{i,i} &= \epsilon^{(2)} \\ &= \frac{\partial u_i}{\partial x_i} \\ &= \nabla.\underline{u} \end{aligned} \tag{1.26}$$

where $\epsilon^{(2)}$ is the residual error of the continuity equation, and the momentum equation:

$$\begin{aligned} \rho\dot{u}_i + \rho u_{i,j}u_j + \rho F_i - \sigma_{ij,j} &= \epsilon_i^{(1)} \\ &= \rho\frac{\partial u_i}{\partial t} + \rho\frac{\partial u_i}{\partial x_j}u_j + \rho F_i - \sigma_{ij,j} \\ &= \rho\frac{\partial \underline{u}}{\partial t} + \rho(\underline{u}.\nabla)\underline{u} + \rho F_i - \sigma_{ij,j} \end{aligned} \tag{1.27}$$

where

- $\epsilon_i^{(1)}$ is the residual error of the momentum equation
- stress tensor is defined as $\sigma_{ij} = -P\delta_{ij} + 2\mu d_{ij}$(μ is the dynamic viscosity)
- $\sigma_{ij,j} = \frac{\partial \sigma_{ij}}{\partial x_j}$

The rate of deformation (tensor symmetric) is

$$\begin{aligned} d_{ij} &= \frac{1}{2}(u_{i,j} + u_{j,i}) \\ &= \frac{1}{2}(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}) \end{aligned} \tag{1.28}$$

and hence we can write

$$\begin{aligned}\sigma_{ij,j} &= \frac{\partial}{\partial x_j}[-p\delta_{ij} + \mu(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i})] \\ &= -\nabla p + \frac{\partial}{\partial x_j}[\mu(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i})] \qquad (1.29)\end{aligned}$$

Since μ is constant, then we can re-express this:

$$\begin{aligned}\sigma_{ij,j} &= -\nabla p + \mu\frac{\partial^2 u_i}{\partial x_j^2} + \mu\frac{\partial^2 u_j}{\partial x_i \partial x_j} \\ &= -\nabla p + \mu\frac{\partial^2 u_i}{\partial x_j^2} + \mu\frac{\partial}{\partial x_i}(\nabla.\underline{u}) \qquad (1.30)\end{aligned}$$

Given the continuity equation for an incompressible fluid $\nabla.\underline{u} = 0$ we can write the above as

$$\sigma_{ij,j} = -\nabla p + \mu\frac{\partial^2 u_i}{\partial x_j^2} \qquad (1.31)$$

and therefore the momentum equation becomes

$$\rho\frac{\partial \underline{u}}{\partial t} + \rho(\underline{u}.\nabla)\underline{u} + \rho\underline{F}_{body} + \nabla p - \mu\nabla^2\underline{u} = \underline{\epsilon}^{(1)} \qquad (1.32)$$

After employing the Galerkin method, Chung rewrites the momentum equation as

$$A_{NM}\dot{u}_{Mi} + B_{NiRM}u_{Mi}u_{Rj} + C_{NiM}p_M + D_{NMik}u_{Mk} = E^{(b)}_{Ni} + E^{(s)}_{Ni} \qquad (1.33)$$

where $N, M, R = 1, 2....r($ r is the number of elements)

and

For time-dependent solutions, we write:

$$A_{NM}\dot{u}_{Mi} + B_{NiMR}u_{Rj}u_{Mj} + D_{NM}u_{Ni} + C_{NiS}P_S = E_{Ni} \qquad (1.34)$$

Where node indices $N, M, R, S = 1, 2...r$

and

$A_{NM} = \int_\Omega \rho \Phi_N \Phi_M d\Omega$	Mass matrix
$B_{NiRM} = -\int_\Omega \frac{\rho}{2} \Phi_{N,i} \Phi_R \Phi_M d\Omega$	Convective matrix
$C_{NiM} = -\int_\Omega \Phi_{N,i} \Psi_M d\Omega$	Pressure matrix
$D_{NMik} = -\int_\Omega \mu(\Phi_{N,j}\Phi_{M,j}\delta_i^k + \Phi_{N,j}\Phi_{M,i}\delta_j^k) d\Omega$	Dissipation matrix
$E_{Ni}^{(b)} = \int_\Gamma \rho F_i \Phi_N d\Gamma$	Body force vector
$E_{Ni}^{(s)} = E_{Ni}^{(s_1)} + E_{Ni}^{(s_2)}$	Surface force vector
$E_{Ni}^{(s_1)} = \int_\Gamma \sigma_{ij} n_j {*} \Phi_N d\Gamma \;\; E_{Ni}^{(s_2)} = -\int_\Gamma \frac{\rho}{2} u_j u_j n_i * \Phi_N d\Gamma$	-

$\dot{u}_{Mi} = \frac{u_{Mi}^{(n+1)} - u_{Mi}^{(n)}}{\Delta t}$ or $\frac{\partial \underline{u}_M}{\partial t} = \frac{1}{\Delta t}(\underline{u}_M^{(n+1)} - \underline{u}_M^{(n)})$

and

$$[A_{NM} + \Delta t(1-\theta)(B_{NiMR}u_{Rj}^{(n+1)} + D_{NM}\delta_{ij})]u_{Mj}^{(n+1)} + \Delta t(1-\theta)C_{NiS}p_S^{(n+1)} = \Delta t E_{Ni}^{(n+1)} + [A_{NM}\delta_{ij} - \Delta t\theta(B_{NiMR}u_{Rj}^{(n)} + D_{NM}\delta_{ij})] - \Delta t\theta p_S^{(n+1)} \quad (1.35)$$

Where θ is a relaxation parameter.

These equations are detailed in Chung, pp211; the reader is suggested to pay particular attention to the definition of equation (5-92). For an explanation of the Galerkin Method, see pp41. As an example of implementation of FEM CFD, a small Fortran solver was written – see appendix A.2 for details.

1.6. Tidal modelling

This section surveys finite volume and finite element models that have been constructed as numerical models on a variety of scales, from the estuarine with dimensions of around 100 m, to global models with cells perhaps kilometres across. In particular, the different approaches taken to balance accuracy with computational complexity will be contrasted.

1.6.1. Finite difference methods

Finite difference methods in physical oceanography are formulated by carving up the solution space, ie. the oceans bounded by their adjacent bathymetry, into a regular grid. Whilst they may vary considerably from the SIMPLE algorithm and its descendants, the essential premise of each is the same.

Depth-averaged models

Gjevik and Straume, 1989

Formulation Gjevik and Straume [33] constructed a two-dimensional non-Boussinesq model to simulate tidal currents in the Arctic ocean and the neighbouring Nordic waters. It is grid-based, implemented on a stereographic map projection (see fig. 1.12).

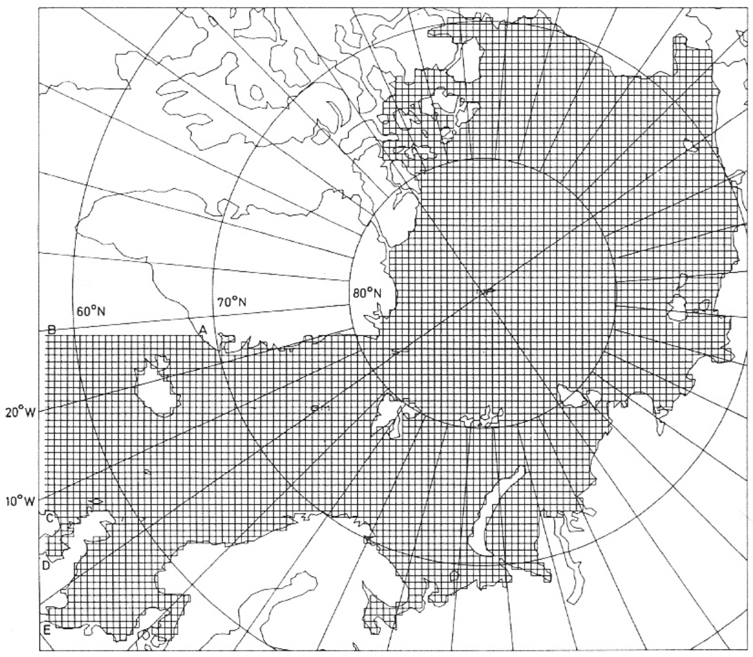


Figure 1.12: A stereographic projection of the Arctic ocean

Rather than assuming a flat ocean-bed, they employ a depth-averaged model, where the variation in ocean depth is taken into account through extra terms in the momentum and continuity equations. They write the momentum equations in Cartesian co-ordinates via the stereographic projection centred on latitude ϕ_s as

$$\frac{\partial U}{\partial t} - fV = -mgh\frac{\partial}{\partial x}(\eta - \tilde{\eta}) + A_x + B_x \tag{1.36}$$

$$\frac{\partial V}{\partial t} + fU = -mgh\frac{\partial}{\partial y}(\eta - \tilde{\eta}) + A_y + B_y \tag{1.37}$$

and the continuity equation as

$$\frac{\partial \eta}{\partial t} = -m^2\left[\frac{\partial}{\partial x}(\frac{U}{m}) + \frac{\partial}{\partial y}(\frac{V}{M})\right] \tag{1.38}$$

Where:

- $f = 2\Omega \sin\phi$ is the Coriolis parameter (ϕ being degrees latitude)
- $m = \frac{a}{a_s} = \frac{r^2+r_0^2}{r_s^2+r_0^2}$ is the map factor, $a = \frac{1+\sin\phi_0}{1+\sin\phi}$; $m = 1$ at $\phi_s = 60^o$. This scales the model to take into account variations due to stereographic projection.
- η is the deviation of the sea surface height from its undisturbed position, and $\tilde{\eta}$ the equilibrium tide
- h is the undisturbed depth of the ocean

(For a complete set of definitions, see the paper).

What is important here is how they copy with an ocean of varying depth yet highly anisotropic flow, which is a good approximation of large-scale currents.

To account for the increased drag in shallow waters, they introduce bottom stress via the components A_x and A_y; as in deeper waters ocean-bottom friction will have less of an impact, drag forces due to this must decrease as the depth of the ocean increases. They define this as

$$A_x = \frac{-kU}{h} \tag{1.39}$$

and

$$A_y = \frac{-kV}{h} \tag{1.40}$$

,

where k is the bottom friction coefficient, itself defined as $k = c_r\frac{\sqrt{(U^2+V^2)}}{h}$ and c_r is a drag coefficient.

If we rewrite the above as

$$\underline{A} = -\frac{c_r}{h^2}\sqrt{(U^2 + V^2)}\underline{U} \tag{1.41}$$

It can be seen that this is a quadratic friction law; used frequently in similar models according to Schwiderski [65]. Alternatively, a linear friction law such as the one below may by used:

$$k = c_f u_s \tag{1.42}$$

where c_f is the linear drag co-efficient and u_s is a typical speed for tidal currents.

Viscous terms are also modelled – these are represented by the B_x and B_y terms:

$$B_x = \nu m \nabla^2 U \tag{1.43}$$

and

$$B_y = \nu m \nabla^2 V \tag{1.44}$$

where ν is the eddy viscosity, which varies with h as $\nu = qh$ and q is a constant, with a typical value of $100m/s$.

Gvejik and Straume's model has both coastal and open boundaries. For coastal boundaries, the water is stationary, ie.

$$\underline{U} = 0 \tag{1.45}$$

$$\eta = 0 \tag{1.46}$$

whereas along open (sea) boundaries, the velocity normal to the boundary can be specified as

$$U_n = U_{no}\cos(\omega t - \delta_o) \tag{1.47}$$

where U_{no} is the amplitude, ω is the angular velocity of the tidal component and δ_o is the phase lag. These constants are determined from empirical data.

Similarly, the sea-surface displacement can be written as

$$\eta = H_o \cos(\omega t - G_o) \tag{1.48}$$

Where H_o is the amplitude of η; H_o and G_o are again calculated in a similar manner to above.

Computational model Moving on to the numerical solution of these equations, the Arakawa C grid was used, with a grid spacing of 50 km. The grid was staggered, as the volume flux (U, V) was calculated at separate points to the sea-surface elevation η. For further details, consult the paper.

Measurements and empirical data For each point on the grid, the depth of the ocean had to be determined from bathymetric charts. As this data was sparse, and not be aligned with the grid points of the model, linear interpolation was used to calculate the values of h for each of the grid points. For the open boundaries, volume flux and sea-surface elevation values were gleaned from Schwiderski's (1981) tidal charts and Flathers' (1981) Atlantic tidal model at the Proudman Oceanographic Laboratory.

For comparison with the model, actual values for the elevation η and volume flux (U, V) were taken from 116 tidal stations in the Arctic Ocean and Barents Sea area, shown in figure 1.13.

Gjevik, Nøst and Straume In this paper, a virtually identical model is used to the above in section 1.6.1 to model the Barents Sea, except with greater grid resolution (25 km). This paper does go into greater detail about how data for the model is acquired, for both comparison and simulation. Boundary conditions for the open boundaries were taken from the same source as Gjevik and Straume [33]; with volume flux and elevation data taken from 46 tidal stations, both coastal and open water, plus measurements from 30 mooring

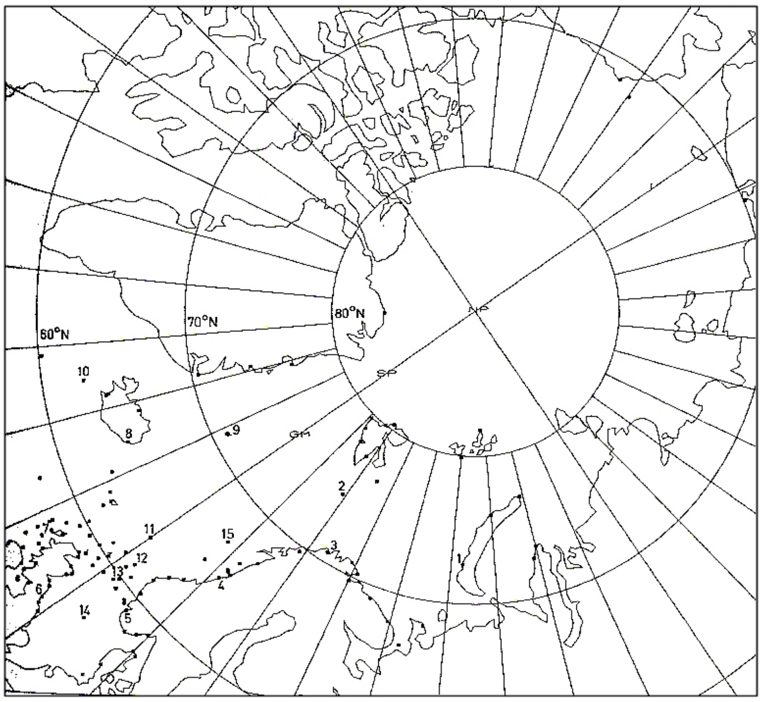


Figure 1.13: Locations of the Arctic tidal stations (courtesy of Gjevik and Straume)

stations – taken from various Norwegian institutions, namely the Norwegian Hydrographic Service, and the Institute of Marine Science. In fig. 1.14 we can see their locations plus depth contours.

In addition, they attempt to compute streaklines to see if there is a connection between the streaklines calculated from tidal currents and rifts in the sea ice, in order to validate the model.

Current profiling Up until this point, the vertical current profile has either assumed a linear or quadratic form; Nøst presents a technique for extracting two-dimensional depth-integrated models such as the two above, concentrating on the Barents Sea [58].

Nøst employs equations very similar to the previous papers, however in complex form. He explicitly writes the complex, depth-averaged velocity as:

$$\bar{w} = \bar{u} + i\bar{v} = \frac{1}{h}\int_{-h}^{0} w dz \tag{1.49}$$

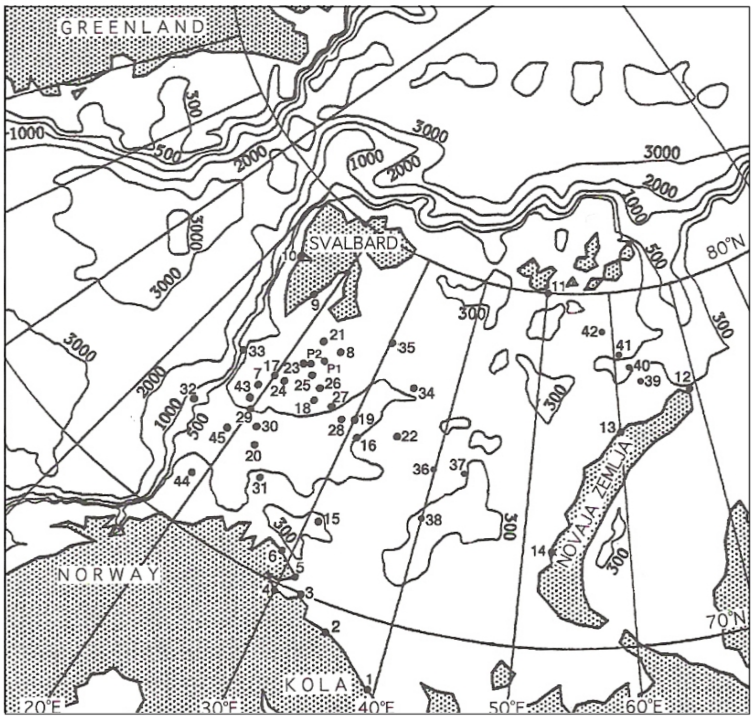


Figure 1.14: Mooring stations in the Barents Sea

and thus

$$\frac{\partial \bar{w}}{\partial t} + if\bar{w} = -g(S + \tilde{S}) + \frac{1}{\rho h}(T_o - T_h) + \bar{L} \tag{1.50}$$

where

- S is the complex surface gradient $\frac{\partial \eta}{\partial x} + i\frac{\partial \eta}{\partial y}$, and $\tilde{S}$ the gradient for the equilibrium tide
- T_0 and T_h represent the shear stresses in complex form, at the surface and at the ocean floor.
- $\bar{L}$ is the friction caused by the depth-average velocity.
- ρ the mean seawater density

At this point, the author goes on to describe the surface stress T_0 induced by pack ice, as this will surely have an impact on the vertical current profile in Arctic regions (see paper for more details). Following from that, he divides the current into the mean and the deviation from the mean, thus

$$w = \bar{w} + w' \tag{1.51}$$

This becomes, when inserted into his momentum equations from w and subtracting the depth-averaged part:

$$\frac{\partial w'}{\partial t} + ifw' = \frac{1}{h^2}\frac{\partial}{\partial s}\left(\nu\frac{\partial w'}{\partial s}\right) - \frac{1}{\rho h}(T_0 - T_h) \tag{1.52}$$

Nøst employs the Galerkin method to this deviation into basis functions, thus:

$$w' = w_n = w_o(s,t) + \sum_{r=1}^{N} c_r(t) w_r(s) \tag{1.53}$$

where t is time and s is a proprietary vertical co-ordinate, such that $s = -1$ at the seabed and 0 at the surface.

By assuming linear laws of friction, and zero surface stress he then forms a trial solution. He then applies the Galerkin method to the momentum equation for w', and after some involved mathematics combining the eddy viscosity, ends up with a description of the total current as:

$$w = (1 - a_b)\bar{w} + \sum_{r=1}^{N} C_r(t) F_r w_r(s) \tag{1.54}$$

Definitions of $C_r(t)$ and F_r can be found in the paper; a_b is a coefficient representing the effect of friction from the seabed.

Figure 1.15 is a comparison of computing profiles with actual data from a tidal station in shallow wate, with crosses indicating observed data. Figure 1.16 shows profiles for a sea covered in pack ice.

Clearly there is some correlation between actual and computed data; Nøst concludes as much in his paper, however he also acknowledges the key effect of bathymetry on current profiles. Realistic current profiles at smaller scales (approx. 10 m) are required for the models developed in this PhD, and such profiles are likely to vary considerably with irregularities on the sea floor. Couch [19] uses depth-integrated equations to model tidal currents on a smaller

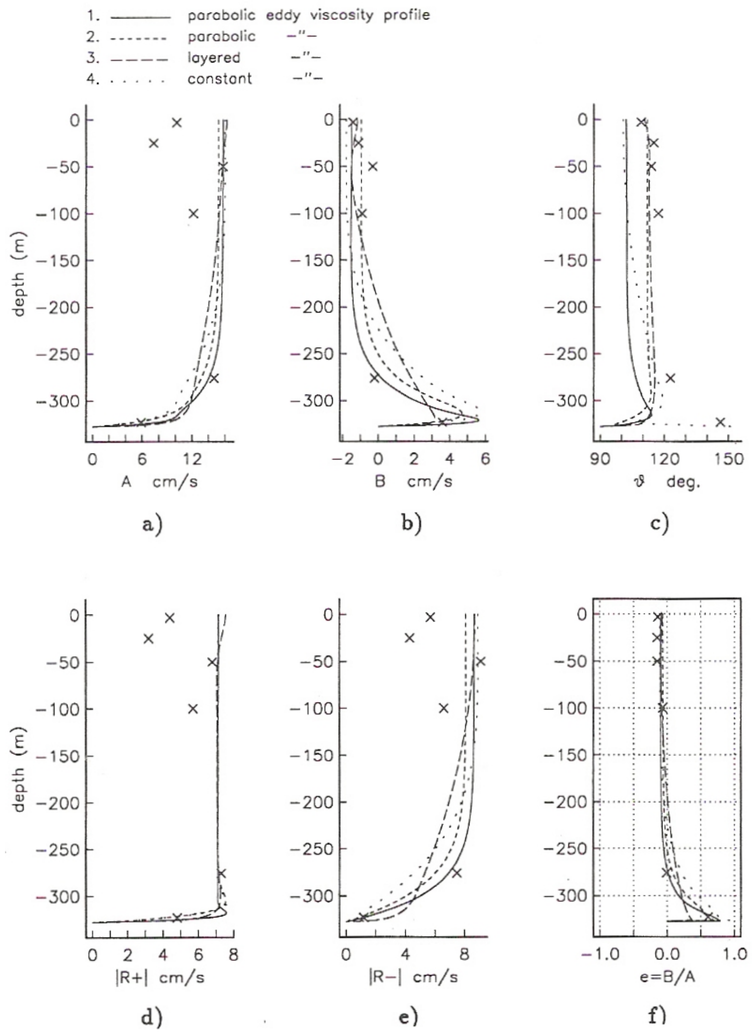


Figure 1.15: Computed current profiles from Nøst. The x-axis represents horizontal speed, and the y-axis depth.

scale, eg. headlands and so on, and goes into great depth about topographic and bathymetric eddy generation; he also deals with wind (ie. surface) stress.

General three-dimensional models

There are problems in introducing three dimensional Navier-Stokes equations; the additional computational complexity involved may require a level of processing power not readily available enough for such models to be pragmatic.

Marshal et al overcame this by separating their model into surface, non-hydrostatic and hydrostatic components [50]; they employ some “physically

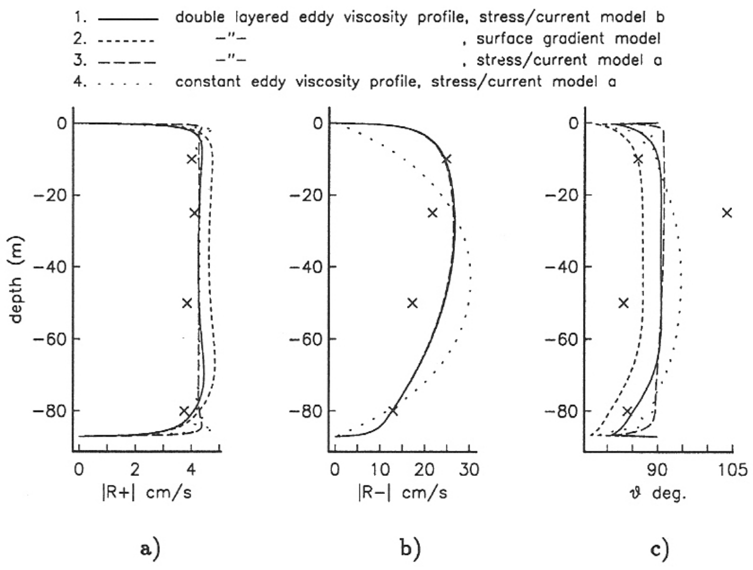


Figure 1.16: Current profiles for sea surface covered with pack ice, from Nøst. The x-axis represents horizontal speed, and the y-axis depth.

motivated preconditioners" to take advantage of the fact that, where the horizontal motion greatly exceeds vertical motion, common in oceanic currents, the flow can be treated as hydrostatic. This simplification greatly reduces the computing time required, allowing it to compete with existing hydrostatic models.

The application of this model to physical oceanography is covered by Marshal et al in their second paper [51].

Sigma coordinates

Sigma level models attempt to tackle the problems of resolving irregular bathymetric features and free surfaces by introducing the σ co-ordinate, which follows both the terrain and the ocean surface by replacing the z-components of position and velocity.

Taking Kowalik and Murty [42] and Vreugdenhil [74] as our guide, the transformation from z co-ordinate to sigma is defined as

$$\sigma = \frac{z - \eta}{D} \tag{1.55}$$

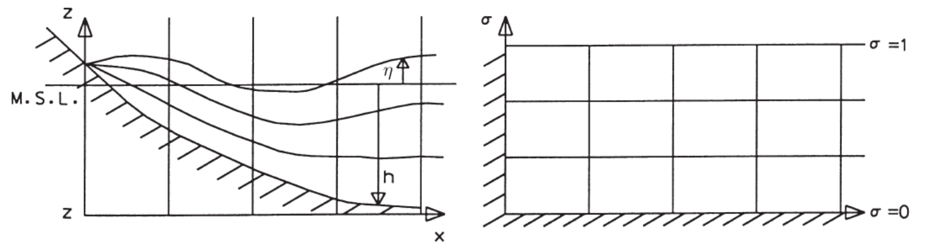


Figure 1.17: The transformation from Cartesian to σ co-ordinates

where h is the depth from the undisturbed sea surface, η is the displacement from equilibrium; $D = h + \eta$ represents the depth of the water column.

Now suppose that the transformation from normal coordinates to sigma coordinates is

$$(x, y, z) \Rightarrow (\bar{x}, \bar{y}, \sigma) \tag{1.56}$$

where

$\bar{x} = x$, $\bar{y} = y$, $\bar{t} = t$

and that ϕ represents some quality or property of the fluid. We can then write

$$\frac{\partial \phi}{\partial x} = \frac{\partial \phi}{\partial \bar{x}} - \frac{\partial \phi}{\partial \sigma}(\frac{\sigma}{D}\frac{\partial D}{\partial \bar{x}} + \frac{1}{D}\frac{\partial \eta}{\partial \bar{x}}) \tag{1.57}$$

$$\frac{\partial \phi}{\partial y} = \frac{\partial \phi}{\partial \bar{y}} - \frac{\partial \phi}{\partial \sigma}(\frac{\sigma}{D}\frac{\partial D}{\partial \bar{y}} + \frac{1}{D}\frac{\partial \eta}{\partial \bar{y}}) \tag{1.58}$$

$$\frac{\partial \phi}{\partial z} = \frac{1}{D}\frac{\partial \phi}{\partial \sigma} \tag{1.59}$$

$$\frac{\partial \phi}{\partial t} = \frac{\partial \phi}{\partial \bar{t}} - \frac{\partial \phi}{\partial \sigma}(\frac{\sigma}{D}\frac{\partial D}{\partial \bar{t}} + \frac{1}{D}\frac{\partial \eta}{\partial \bar{t}}) \tag{1.60}$$

Writing

$$Q_x = \sigma\frac{\partial D}{\partial x} + \frac{\partial \eta}{\partial x} \tag{1.61}$$

and

$$Q_y = \sigma \frac{\partial D}{\partial y} + \frac{\partial \eta}{\partial y} \tag{1.62}$$

Now the vertical velocity can be written as

$$w = D\frac{\partial \sigma}{\partial t} + (\sigma + 1)\frac{\partial \eta}{\partial t} + uQ_x + vQ_y \tag{1.63}$$

And by assuming incompressible fluid flow, the continuity equation can be written as

$$\frac{\partial Du}{\partial x} + \frac{\partial Dv}{\partial y} + D\frac{\partial}{\partial \sigma}\frac{\partial \sigma}{\partial t} + \frac{\partial \eta}{\partial t} = 0 \tag{1.64}$$

Similarly, the momentum equations can be rewritten in sigma coordinates. This type of model is popular mainly due to their similarity with more conventional finite difference models, ie. those on regular grids, and due to their abilities to deal with terrain and free surface. Yet, unlike the vertically-integrated, shallow-water equations, they also allow more accurate simulation of the vertical component of flow, especially boundary effects near the seabed, and also the surface with wind-driven currents.

Huang and Spaulding [39] developed a sigma model to predict the dispersion of pollutants in estuaries, modelling salinity, temperature and the pollutant via the continuity equations for fluid properties; turbulence was also factored via $k - \epsilon$ model – see 'turbulence' section for details. A more generic and flexible approach was taken by Drago et al [23], whose model featured variable boundary conditions, allowing parts to become flooded, or conversely empty as the sea-level changes. This would have particularly important consequences for mass conservation in areas of estuaries, etc. where such a change may affect the size of the computational domain, eg. tidal mud flats.

Sub-grid processes: turbulence

In finite difference models and indeed finite element models, there are effects which occur at a resolution finer than the grid used, which can effect the properties of the flow. This generally comes under the heading of turbulence

– sub-grid disturbances which have an associated viscosity of their own, called eddy viscosity. One turbulence model that has risen to prominence is the $k - \epsilon$ model, which although not without its limitations, is widely understood. Li and Zhu [46] deal with a 3D, sigma coordinate model for flow over submerged objects; which may be of particular interest, since such a model could describe turbulence generated from underwater marine turbine farms. Their model deals with flow over a cube, which rests on the solid 'floor'.

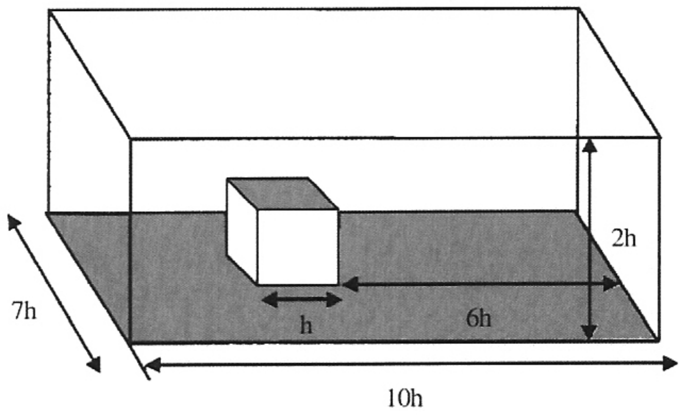


Figure 1.18: Li and Zhu: flow over a cube generating turbulence

Good agreement was found with practical experiments, suggesting this may be a viable approach to solving turbulent flow in such circumstances. Adaptive grids in FE models however, may be able to overcome some of the complexities in modelling sub-grid processes, since the size of the mesh cells can be varied according to need.

1.6.2. Finite element methods

Finite element models have only recently emerged as viable alternatives to finite difference methods in ocean modelling. It is suggested here that due to their ability to adapt elements to irregular boundaries such as coastlines and undulating sea beds, that they are perhaps better suited to the task; both two dimensional and three dimensional models are discussed below.

Depth-averaged models

One example of a finite element technique being used on a local-scale model is by Canu, Solidoro and Umgiesser [17] in their ecological model of the Venetian Lagoon. In fig. 1.19 we can see how the lagoon was divided up in triangular elements, the more densely packed areas representing the deeper channels:

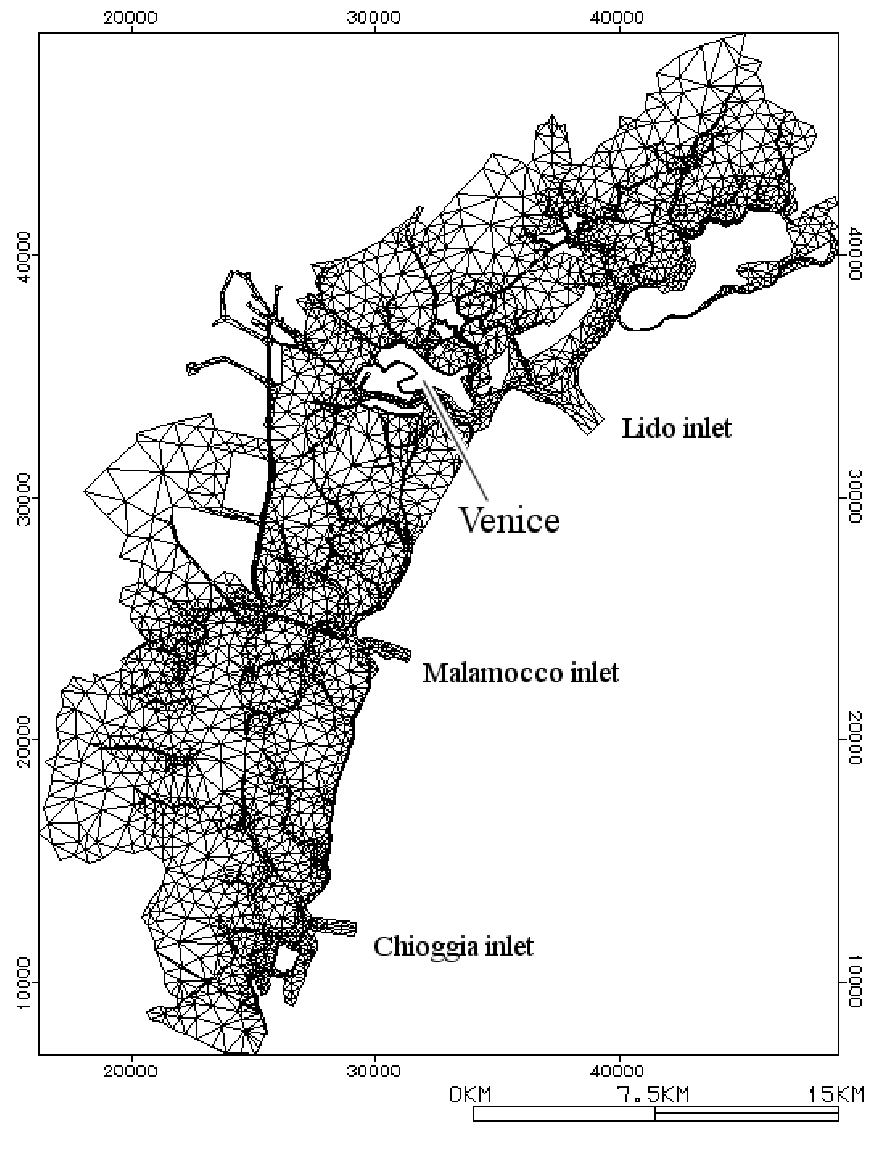


Figure 1.19: Finite element simulation of the Venetian lagoon

Their simulation was a coupling of two models: one ecological (WASP) and one fluid dynamic (SHYFEM) – we will concentrate on the fluid dynamic model [72]. Based on the shallow water equations, for momentum:

$$\frac{\partial U}{\partial t} + gD\frac{\partial \eta}{\partial x} + RU + X = 0 \tag{1.65}$$

$$\frac{\partial V}{\partial t} + gD\frac{\partial \eta}{\partial y} + RV + Y = 0 \tag{1.66}$$

and for continuity:

$$\frac{\partial \eta}{\partial t} + \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} = 0 \tag{1.67}$$

where

- U and V are the vertically-integrated horizontal velocity components.
- $R = r\frac{\sqrt{u^2+v^2}}{D}$ is the friction coefficient; r is a dimensionless parameter.
- X and Y represent additional terms

Notice there are no molecular viscosity terms, since the shallow-water equations consider only inviscid flow; also, due to the scale of the model the Coriolis effect has been ignored. Numerically, for time-integration a semi-implicit method is used. η, the water level, is solved implicitly via a non-linear second order PDE. ie. through a combination of Thomas and ADI (alternating direction implicit) algorithms:

$$\eta^{n+1} - \left(\frac{\Delta t}{2}\right)^2 \frac{g}{1+\Delta tR}\left(\frac{\partial}{\partial x}(D\frac{\partial \eta^{n+1}}{\partial x}) + \frac{\partial}{\partial y}(D\frac{\partial \eta^{n+1}}{\partial y})\right) = K_n \tag{1.68}$$

where

- Δt is the size of the discrete timestep
- g is the acceleration due to gravity
- K_n represents terms calculated from variables at n^{th} timestep

U^{n+1} and V^{n+1}, the velocity components at the next timestep, are calculated explicitly as a function of variables at the n^{th} iteration. Finite elements were used to discretise the model spatially; a staggered, triangular mesh was used – this was generated via an automatic mesh generator and then manually manipulated. Two different types of form function were used for η and velocity components; linear interpolation for η and 'stepped' constant form functions for velocity.

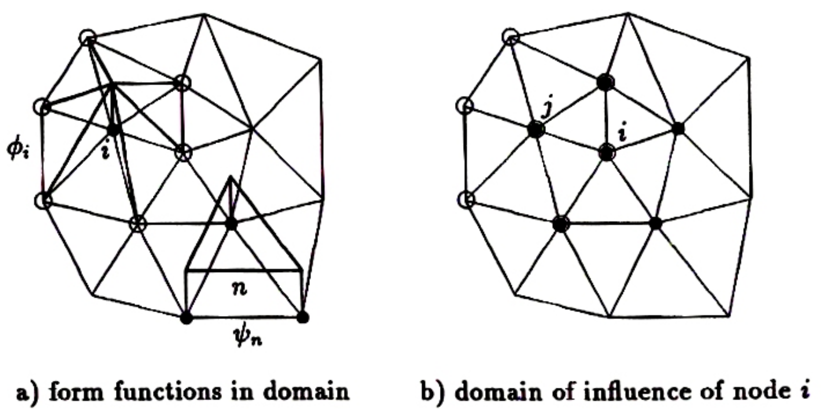

ϕ_i
i
n
ψ_n
j
i
a) form functions in domain
b) domain of influence of node i

Figure 1.20: Finite element form functions – ψ_i represents the form function for the i^{th} node in the finite element mesh, and ϕ_n the solution in element n. j is the index of nodes that neighbour node i.

Care was taken when applying the Galerkin method – if linear form functions were applied directly to the velocity and surface elevation equations, the model would not satisfy the continuity equation. By using staggered elements, only the surface elevations of elements around a particular node are needed to compute the velocities of the corresponding triangular elements; and by doing so the model conserves mass.

Furthermore, the lagoon is tidal and contains flats that are submerged in shallow water, but not at low tide. The authors employed a simple drying and wetting mechanism; if the water level of one of an element's nodes fell below a minimum value (5 cm is quoted), the element is considered 'dry' and removed from the calculations. Similarly, neighbouring nodes gain a water level higher than this minimum value, the element is is considered 'wet' and reintroduced into the simulation.

Three dimensional models

Topography and bathymetry, as has already been mentioned, have a strong effect on the nature of tidal flow [32]. Further to that, the sea bed and coastlines often have irregular, complex geometry (such as narrow straits, steep submarine slopes, islands, etc.), which cannot accurately be represented by regular meshes and finite difference models, nor the flow that results therein. Thirdly, especially on smaller scales, eg. headlands and islands, the hydrostatic approximation is no longer valid and depth-integrated or shallow-water equations cannot be used as the vertical component of velocity can no longer be ignored, and vertical profile of the horizontal velocity components may not be easily approximated as in previously mentioned papers, or even more recently in Umgiesser et al [72].

Finally, three-dimensional finite element models with irregular meshes, ie. variable-sized elements, can both horizontally and vertically concentrate finer spatial resolution – thus computing resources – on areas where it is most needed, such as areas of turbulent flow. This allows larger simulations to be tackled than may be permitted with Finite Difference models, even terrain-following σ ones [23] ; it also brings with it its own problems due to the inhomogeneous mesh-spacing, such as unphysical wave scattering. Despite previous work on three-dimensional FEM for fluid dynamics, especially for parallel domain decomposition [76], until very recently, most tidal FEM simulations were almost completely restricted to two-dimensional depth-integrated coastal models, although there have been attempts to lay the foundations for global models eg. Legrand [43]. Speculatively speaking, processing power and the previous lack thereof, may have played a part in the dearth of three-dimensional models.

Fixed meshes In their 2003 paper, Nechaev et al [57] present a diagnostic, ie. steady-state FEM; it uses a quasi-unstructured mesh based upon tetrahedral elements. Their mesh is generated in four steps:

1. The sea surface is tessellated with triangular facets.
2. The corners of these are then projected downwards, forming triangular vertical columns; triangulating the sea floor.
3. The columns are divided into horizontal layers.
4. Each layer of each column is then subdivided into one to three tetrahedra.

This is better exemplified with diagrams reproduced from the paper – figure 1.21 shows a plan view of the triangulation of the sea surface (the dark blue areas represent deeper waters), and in fig. 1.22 there is a cross-section of the vertical mesh structure below, with temperature colour map.

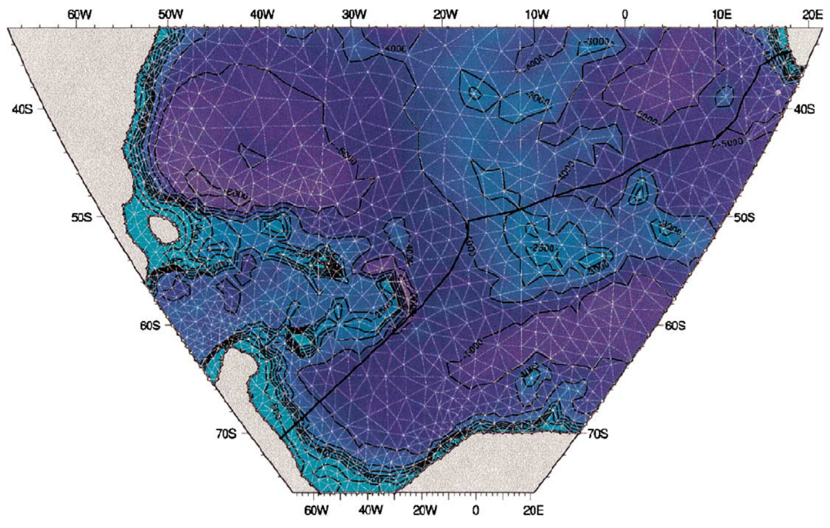


Figure 1.21: Plan view of Nachaev's finite element mesh. The darker areas represent deeper parts of the ocean; light blue represents the shallower regions.

As can be seen, it is not a truly irregular mesh. Their model uses Galerkin projection to achieve the discretisation of the governing equations. Similarly, Danilov, Kivman and Schröter [21] introduced a prognostic model which could either operate on a two-dimensional unstructured triangular mesh and vertically-integrated velocities, or a three-dimensional mesh based on tetrahedra, of similar structure to Nehchaev. Salinity, temperature and the horizontal velocity components are linear interpolated across the mesh, whereas the vertical component of velocity, w, is in their words 'elementwise constant': ie. a

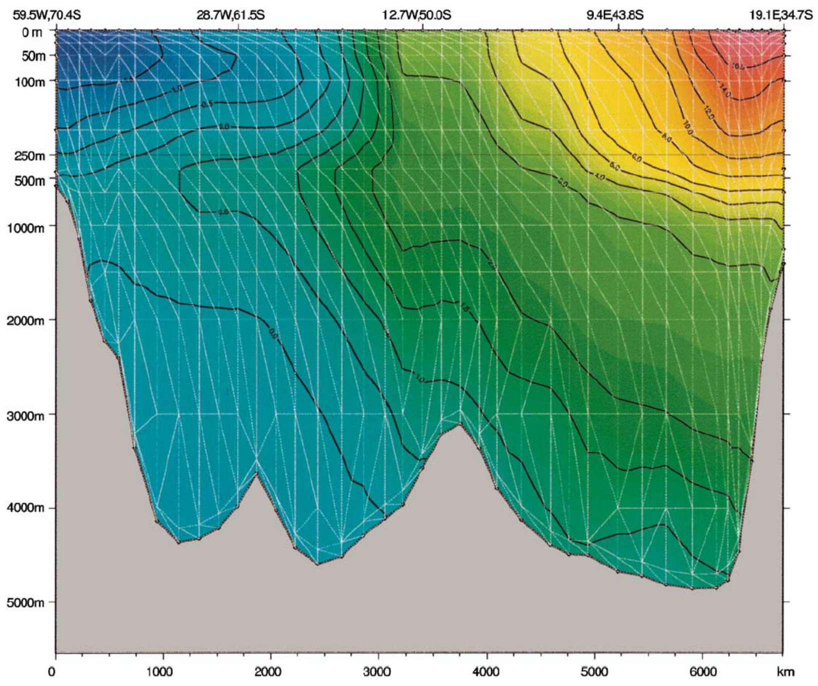


Figure 1.22: Vertical cross-section of Nachaev's finite element mesh, coloured by temperature – red is warmer than blue.

3D equivalent of the stepped constants in Umgiesser and Bergamasco's model. Rather than use the standard Galerkin method for discretising the Navier-Stokes and primitive equations, they use the Galerkin Least Squares method to combat instability (see paper for details, pp134-140).

Adaptive meshes One of the main problems with fixed meshes, irregular or not, is that while they have the ability to have varied resolution, with a finer mesh near points of interest, ie. large velocity gradients, they cannot change these locally concentrated areas. There are many features of the flow that may require greater spatial accuracy with the progression of time; turbulent wakes and eddies are but two. While one option may be to increase the resolution of the mesh in paths or areas where such phenomena are likely to occur prior to the simulation run, this also increases the wastage of processing power – once an eddy has passed downstream, or the flow has become less violent, fewer mesh nodes (and thus elements) may be sufficient for accurate modelling. In this case, what is needed is an adaptive mesh; a mesh where mesh points can be added, removed, or even moved to raise or lower the level of accuracy as required.

Huang and Russell introduced an adaptive method for a two-dimensional gradient flow problem in their 1999 paper [38]. Dynamically adaptive – while the number of mesh nodes remains fixed, the nodes are moved towards regions of the space-time domain where the solution gradient is steep; doing so will minimise errors. This is facilitated by what is known as a monitor function which gives some measure of potential error and the mesh points are moved accordingly.

Mesh adaptation, as applied to ocean modelling, is covered in detail by Piggot et al [64]. Specifically, they discuss h adaptivity (static mesh points, with varying number of mesh nodes), r adaptivity (moving mesh nodes) and hr adaptivity, which is a combination of the other two. For optimising h adaptive meshes, they introduce what they call an objective functional, as a measure of local mesh accuracy (see paper for justification):

$$F_e = \frac{1}{2}\sum_{l \in L_e}(r_l - 1)^2 + (\frac{\alpha}{\rho_e} - 1)^2 \tag{1.69}$$

where

- L_e is the set of edges for element e
- r_l is the length, with respect to the edge-centred metric tensor, of side l
- ρ_e is the radius of the largest sphere that can fit in element e.
- The second term $(\frac{\alpha}{\rho_e} - 1)^2$ takes the value of 0 for an element with an aspect ratio of 1, and from this α can be calculated.

Through minimising the function F_e, mesh optimisation is achieved – this is typically done by splitting elements, collapsing edges, etc. Such methods, however, cannot track moving features of flow, for instance, fronts and eddies. In such circumstances, an h adaptive method would remove points as the feature progressed and add them elsewhere to keep the mesh well-resolved; this is an inefficient process.

Moving mesh methods (or r adaptive methods) somewhat similar to Huang and Russell's are better suited to these conditions. They present three methods: mesh smoothing based node movement, variational based mesh movement, and free surface mesh movement. Mesh smoothing methods rely on an error measurement similar to the h method; the physical analogy drawn here is a series of springs of varying stiffness, and by minimising the local energy of the springs, you minimise error. In fig. 1.23 there are six triangular elements arranged in a hexagon. The left hand hexagon represents the intial node position, and the right hand one represents the local energy minimised state; variational based methods are using variational techniques to equally distribute the error.

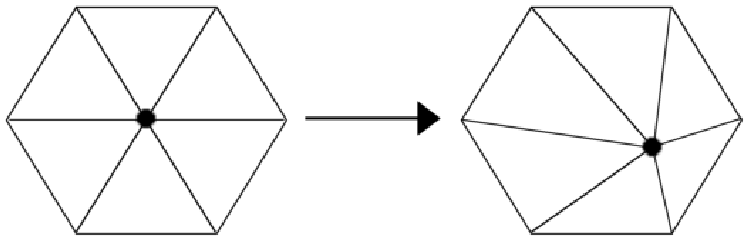


Figure 1.23: Finite element mesh r adaption

Thirdly, free surface mesh movement can be used to represent the free surface in ocean models. In unstructured meshes, a change in the surface elevation η is passed down through the mesh by interpolation – as shown in fig. 1.24, there are simularities with σ layers in finite difference models.

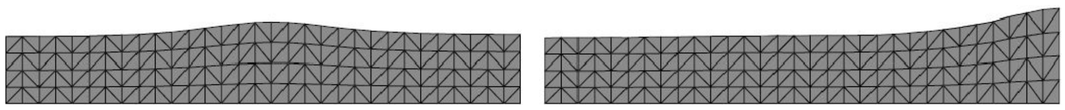


Figure 1.24: An hr-adaptive free surface at different times

However, h and r adaptive methods both have advantages and disadvantages. As stated above, h methods have the advantage of being able to concentrate computing power in particular instances, increasing the number of nodes where more are needed, and conversely removing nodes when less are needed to maintain an adequate accuracy of solution. While having disadvantages, this reduces the computational complexity of the problem. r adaptive methods on the other hand, have the ability to move mesh nodes towards areas with steep solution gradients, but cannot reduce the number of nodes used in the finite element mesh; thus it is feasible that at some stage a higher order of accuracy is maintained than is needed, wasting computing power on unnecessarily fine detail.

Clearly, a combination of the two methods – an hr adaptive method, would be desirable. Unfortunately there are few examples of this in literature according to Piggot; approaches vary from coupling a two dimensional moving mesh method to an h-adaptive method which locally alters the mesh to achieve the desired error tolerance, to splitting the solution domain into regions where h and r adaptive methods can be applied separately as appropriate. At the time of writing, Imperial college's ICOM/Fluidity [60] is the only known robust hr-adaptive CFD software tailored to three dimensional oceanographic models.

Chapter 2

CFD package validation

2.1. Introduction

Fluidity, a computational fluid dynamics software package from Imperial College, was decided upon as the foundations on which the turbine model would be built. It was originally developed for oceanographic applications, but is now emerging as a versatile model for a broader range of systems. To validate this software, the well-known problem of laminar flow over a flat plate was modelled and compared with analytic solutions, as well as those found with COMSOL, which has a CFD solver. This chapter represents parts of a paper submitted to the journal *Computers & Fluids*, by Dr Wolf Früh and Angus Creech, which is still in review at the time of writing.

2.2. About Fluidity

Fluidity [63] was originally developed for oceanographic applications [28] [29] [60], but is now emerging as a versatile model for a broader range of systems. One of the particular strengths of the Fluidity code is the adaptivity of the mesh whereby both the number of nodes and the position of each node can be adjusted to provide best resolution where it is required at minimum computational cost.

2.3. Boundary layer theory

The simplest model to describe the evolving boundary layer in the confined channel with a no-slip condition at the bottom of the channel and a stress-free condition at the upper boundary can be obtained from a minor modification of the standard boundary layer development over a flat plate. The starting point is the von Karman integral momentum equation which states that the growth of the boundary layer, as measured by the momentum thickness, θ, is proportional to the wall shear stress, τ_w normalised by ρu_F^2, u_F being the freestream velocity:

$$\frac{d\theta}{dx} = \frac{\tau_w}{\rho u_F^2} \tag{2.1}$$

Where x is the vertical distance from the plate. Rather than developing the theory in terms of the momentum thickness, we will use the equation above to derive one for the displacement thickness, $\delta*$. The reason is that this is the measure of the boundary layer which is most easily and reliably computed by the computational models used here. Following the assumption that the velocity profile in this modified situation is the Blasius profile, we can convert (2.1) to an equation for the displacement thickness using the standard laminar boundary layer equation and known conversion factors between the momentum thickness δ, δ^*, and θ (see [52]).

The conversion factor between the displacement and momentum thickness is

$$\gamma_1 = \frac{\delta^*}{\theta} = 2.586 \tag{2.2}$$

Variational based methods use variational techniques to equally distribute the error, in a similar manner to Huaug and Russell [38]. The conversion factor between the displacement thickness and the boundary layer thickness is

$$\gamma_2 = \frac{\delta^*}{\delta} = 0.344 \tag{2.3}$$

and the standard solution for the momentum thickness is

$$\frac{\theta}{x} = \gamma_3 Re^{-1/2} \tag{2.4}$$

with $\gamma_3 = 1.66$.

Scaling the definition of the wall shear stress by u_F and δ, respectively gives

$$\tau_w = \mu \left(\frac{du}{dy} \right)_{y=0} = \frac{\mu u_F}{\delta} \left(\frac{du^*}{d\eta} \right)_{\eta=0} = \frac{\gamma_2 \gamma_3 \mu u_F}{\delta^*} \tag{2.5}$$

where u^* and η are the nondimensional velocity and wall distance.

Inserting these factors into (2.1) yields

$$\frac{d\delta^*}{dx} = \frac{\gamma_1 \gamma_2 \gamma_3 \mu}{\rho u_F \delta^*} \equiv \frac{A}{\delta^*} \tag{2.6}$$

where $A = 1.477 \left(\frac{\mu}{\rho u_0} \right)$ is a constant, relating the constants describing the chosen boundary layer velocity profile and the flow conditions.

The modification to the standard boundary layer equation required to take into account the finite height of the domain is in the recognition that the free-stream velocity, u_F, is not constant over x but adjusts as the boundary layer grows to maintain the mass flux through the domain. As the available cross section of the channel is effectively reduced by the displacement thickness, the initial inlet velocity, u_0, over the full height, H, has to increase according to

$$u_0 H = u_F (H - \delta^*) \tag{2.7}$$

Replacing the velocity u_F in (2.6) by $u_F = \left(\frac{u_0 H}{H - \delta^*} \right)$ results in an additional dependence on the boundary layer thickness as

$$\frac{d\delta^*}{dx} = A \left(\frac{1}{\delta^*} - \frac{1}{H} \right) \tag{2.8}$$

Considering the two extreme situations, it can be seen that the above equation approaches (2.6) when the boundary layer is much thinner than the fluid depth but that the growth of the boundary layer approaches zero as the boundary layer thickness approaches the fluid depth.

Equation (2.8) can be solved analytically as

$$Ax = H^2 \ln \left(\frac{H}{H - \delta^*} \right) - H\delta^* \tag{2.9}$$

which is illustrated for a number of values of H in figure 2.1 as the displacement thickness against the scaled downstream distance from the leading edge, Ax. The solid curve labelled with ∞ is that of the standard developing Blasius layer. The limiting of the layer thickness by the upper boundary is immediately obvious, and it can also be seen that the displacement thickness is below the Blasius solution even in the early stages of the boundary layer development.

2.4. Computational models

Two computational models for laminar flow over the flat plate were used. Firstly, a three-dimensional model was run within Fluidity; a second, two-dimensional fixed-mesh approximation of the same model was run with the finite element package COMSOL (based on MatLab), for comparison and validation. First we shall introduce the Fluidity model in detail, and then describe the COMSOL model.

2.4.1. Adaptive-mesh, 3D finite element model in Fluidity

This was a cuboid volume of dimensions $250 \times 10 \times z_{top}$, containing an incompressible fluid of density 1.0 units. The volume was computationally divided into two parts as seen in figure 2.2:

1. The entry length section at $0 < x < 50$. Its existence shall be explained in section 2.4.1.

2. The flat plate section, at $50 \leq x < 250$.

In addition, the following boundary conditions were imposed:

1. **Inflow boundary:** at $x = 0$, the Dirichlet condition of $\underline{u} = \underline{u}_0 = (1, 0, 0)$ was imposed; this is the fluid influx.

2. **Outflow:** At $x = 250$, von Neumann condition $\frac{\partial u_n}{\partial x_n} = 0$ where $1 \leq n \leq 3$ (open).

3. **Side boundaries:** for $y = 0$ and $y = 10$, Dirichlet $v = 0$ and von Neumann $\frac{\partial u_n}{\partial x_n} = 0$ where $n \in (1, 3)$: a slip condition with frictionless sides.

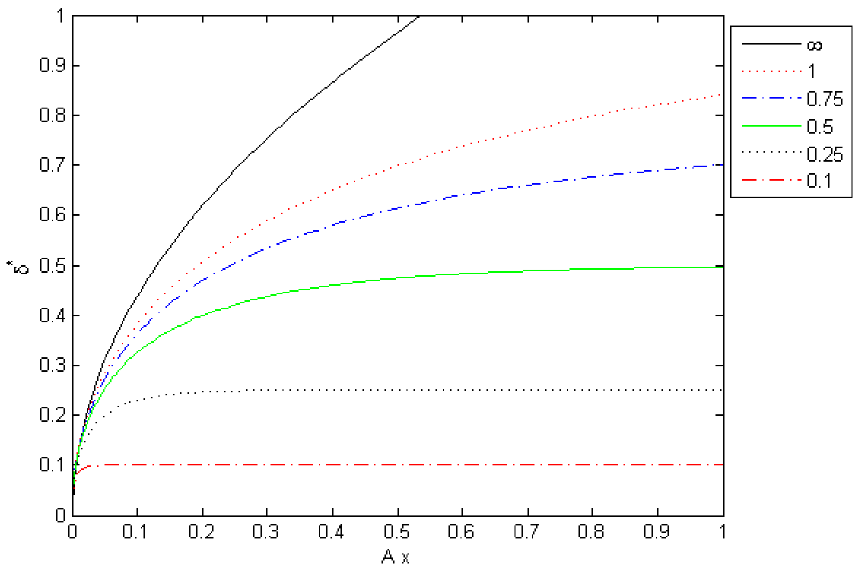


Figure 2.1: The displacement thickness against the scaled downstream distance from the leading edge, using $A = 1$ and the fluid depths as indicated by the legend. The infinity symbol represents the Blasius layer development in an unbounded fluid

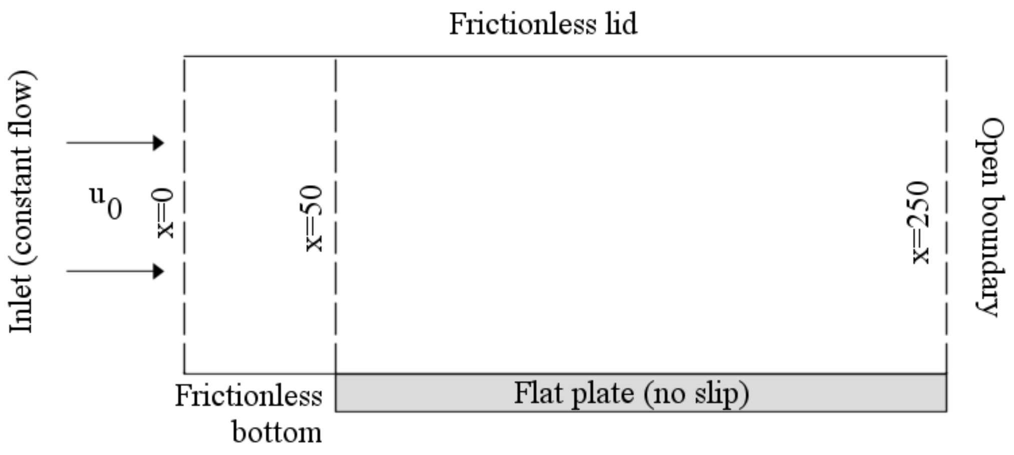


Figure 2.2: Side view of the flat-plate model.

4. **Top boundary:** for $z = z_{top}$, slip condition (frictionless lid).

5. **Bottom of entry length area:** for $z = 0$, slip condition as above.

6. **Plate boundary:** or $x \geq 50$ and $z = 0$, Dirichlet $\underline{u} = 0$; a no-slip boundary.

The entry length section

Most descriptions of boundary layer problems involving flat plates impose the Dirichlet condition for the inflow directly adjacent to the no-slip condition for the plate boundary. In Fluidity, however this creates a discontinuity of the prescribed velocities at the leading edge of the plate, resulting in a velocity gradient that grows as the mesh becomes fine. Therefore if the adaptive algorithm increases the resolution to reduce these gradients it is ultimately bound to fail, and can results in either unrealistic flow artifices or a solver error.

Consequently with an entry length introduced this instability disappeared, and – as will be seen later – the pressure gradient generated caused the boundary layer to effectively spread out, even at lower viscosities.

The computational volume

The ideal model for laminar flow over a flat plate assumes a semi-infinite expanse of fluid in the vertical and positive x directions, which has a free-stream velocity of u_0; this is also the Dirichlet condition at the leftmost boundary.

However, as Fluidity works in a finite computational domain, sensible dimensions had to be chosen. The criteria for these were:

1. **Large enough range for x:** The plate had to be sufficiently long so that the boundary layer developed fully beyond the initial leading edge. Furthermore, the entry length had to be large enough to accomodate the changes in flow upstream due to the onset of boundary forces at the start of the plate.

2. **Small enough range for y:** The theory assumes essentially two-dimensional flow, and so the width of the model had to be limited to suppress the pos-

sibility of three dimensional flow features evolving. However by keeping the Fluidity models three dimensional, extending the analysis to include spanwise motion would remain an option later.

3. **Large enough range for z:** By doing so, we reduce the effect of accelerate of the freestream velocity. The fluid had to be deep enough to ensure the fully-evolved boundary layer would only be a fraction of the total depth.

   By increasing z_{top} as far as is possible within memory and processor constraints, $z_{top} \gg \delta^*$(where δ^* is the displacement thickness, which for laminar flow $\delta^* \approx 0.33\delta$), and thus the acceleration of the fluid is kept to a minimum. This should allow direct comparisons with the analytic descriptions of vertical velocity profiles. Despite the decrease in resolution this will cause, Fluidity's adaptive meshing techniques should allow it to ultimately increase the resolution of the model where it is needed, and decrease it where it is not – and thus be able to accurately resolve flow near the flat plate.

Model dimensions

Three simulations were run, each with a different viscosity, safely with in the Reynolds numbers for laminar flow, ie. $Re \ll 10^5$. In each instance, care was taken that the timestep size Δt and the total simuation time t_{max} were set to ensure the smooth evolution to stable, time-independent flow. To ensure that the size of the model would not unduly affect the development of the boundary layer, for each value of the kinematic viscosity ν, the model's height H was calculated so that the maximum of theoretical boundary layer thickness δ^* was no more than 3% of H, ie.

$$H > \frac{\delta^*}{0.03} \tag{2.10}$$

For laminar flow of incompressible flow, δ^* can be taken [52] as 0.334δ where δ is the z-coordinate of the boundary layer, defined as:

$$\delta = 4.99 \left(\frac{x_{pl}}{\sqrt{Re_x}} \right) \tag{2.11}$$

where x_{pl} is the distance from the plate, and Re_x the Reynolds number of the flow.

It can be shown that max(δ^*) occurs at the outflow on the right of the model, where $x = 250$, and from this H was calculated to a rounded-up value.

Case	#	1	2	3
Kinematic viscosity	ν	0.01	0.1	1.0
Reynolds number	Re	20000	2000	200
Channel length	L	250	250	250
Channel width	W	10	10	10
Channel depth	H	100	500	800
Constant in (refvKSol)	A	0.01477	0.1477	1.477
Displacement thickness	δ^*	2.35	7.44	23.5

Table 2.1: Model parameters and theoretical momentum thickness for the Fluidity simulations. The Reynolds number is that at the trailing end of the plate of length 200, $Re = \frac{200\,u_0}{\nu}$, and the displacement thickness is that found from (2.9) (with the constant A as given here). All parameters are in unit lengths apart from ν, which is unit pressure · unit time.

Mesh adaptivity settings

For $\nu = 0.01$, the interpolation errors on the velocity components were set to $u = 0.025$, $v = 1.0 \times 10^{10}$, and $w = 0.0025$. More specifically, minimum and maximum element lengths for each dimension were geared to ensure greater adaptive sensitivity for the x and particularly z components, as there is negligible flow in the y direction due to the nature of the model. The overall maximum element size was set to 10 units; the minimum 0.01.

At higher viscosities, these settings were scaled to larger values, especially in the z direction. This would allow a similar number of nodes to be used despite the greater volume, the greater interpolation errors compensated by the lower velocity gradients theory predicts.

2.4.2. Fixed-mesh 2D finite element model in COMSOL

As mentioned before, this software was used to provide cross-validation of Fluidity's simulations. A two-dimensional model, effectively just the side view of the simulation seen in figure 2.2, was constructed within COMSOL; prior to directly comparing it with Fluidity, a parametric study was carried out by varying the Reynolds number whilst preserving the domain geometry.

Model dimensions

The model used variables that were scaled against a plate length of $L_p = 1$. The fluid depth was $H = 0.4$, and the entry length 0.25, giving the same aspect ratio as the Fluidity simulation. Also scaled were the inflow velocity and the density, with $u_0 = 1$ and $\rho = 1$. Therefore the Reynolds number of the system can be written as the direct inverse of ν, the kinematic viscosity.

Boundary conditions

The boundary conditions were as before, with a Dirichlet condition of $u = u_0$ at the inlet on the left, a frictionless rigid lid at the top, and a Von Neumann condition of $\frac{du}{dx} = 0$ (zero pressure) at the outflow on the right. The lower boundary had a no-slip condition at the plate section only.

Mesh settings

Three different meshes were used. Two of these concentrated elements near the lower boundary; the first with 3303 elements, the second with 16853. The third mesh also had 16853 elements, but these were evenly distributed. This was used for some cases of flow at low Reynolds numbers.

Variation of Reynolds number

The solution procedure for the parametric study was to initialise the model for a Reynolds number of $Re = 1$ with an initial guess for the solution for the steady-state solver as a uniform flow with horizontal velocity, $u = u_0$, and zero vertical velocity everywhere, and then to iterate to a steady-state solution of that flow. The integration for $Re = 2$ was then a continuation of the solution for $Re = 1$, and so on up to a final Reynolds number of $Re = 105$ in increasing steps of Reynolds number.

2.5. Results

2.5.1. Fluidity, adaptive mesh

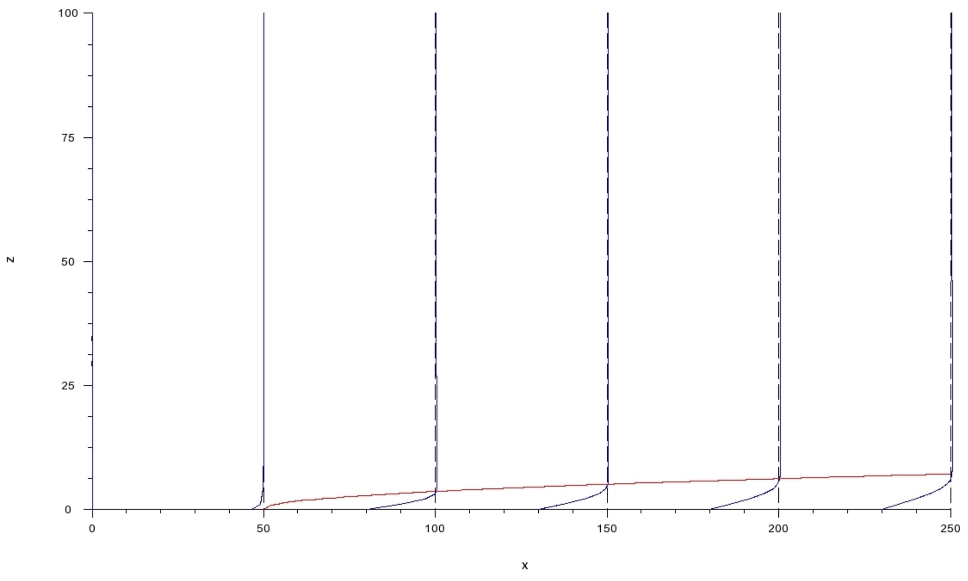
For each set of results, 6 columns of velocity data were extracted from the final time step at $x = (0, 50, 100, 150, 200, 250)$ using barycentric coordinates (see appendix). For each set of data, the data columns were plotted in figure 2.3 as (x', z), where x' is a transformation of x

$$x'_n = x_n\, 20(u - u_0) \tag{2.12}$$

for the n^{th} column with $n = 1, 2, ...6$.

This was designed to demonstrate the progression of the velocity profile downstream from the edge of the plate. Additionally, by placing vertical lines at $x = 0, 50, ..., 250$, the approach to a freestream velocity $u_F \lim u_0$ can be seen. Lastly, the line originating at the leading edge and growing towards the end of the domain represents the theoretical edge of the boundary layer, defined as where the velocity profile reaches 99% of the freestream velocity, given by (2.4).

All results conform largely to the theoretical laminar boundary layer, in particular, the calculated boundary layer thickness as indicated by the crossing of the velocity profiles with the line showing the boundary layer thickness, δ. Some features, however, are at variance to the simple theory. The velocity profile calculated at the leading edge shows a finite boundary layer thickness for all three cases. Another effect which is especially noticable at larger vicosity is a slight bulge in the velocity profiles just above the boundary layer at $x = 100$ and further downstream.



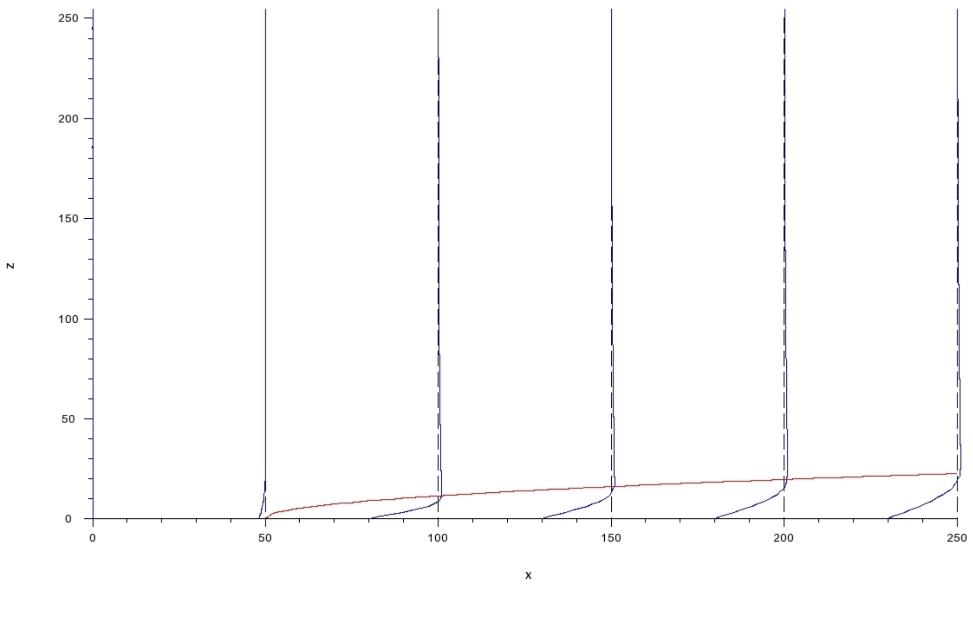


Figure 2.3: Fluidity vertical velocity profiles at regular intervals along the plate, together with the boundary layer thickness of the standard Blasius solution. (a) Case 1 with $\nu = 0.01$ and (b) Case 2 with $\nu = 0.1$.

Figure 2.4 compares the results with a parabolic approximation of the Blasius profile, ie.

$$u(z) = u_0 \left[\frac{2z}{\delta} - \left(\frac{z}{\delta} \right)^2 \right] \tag{2.13}$$

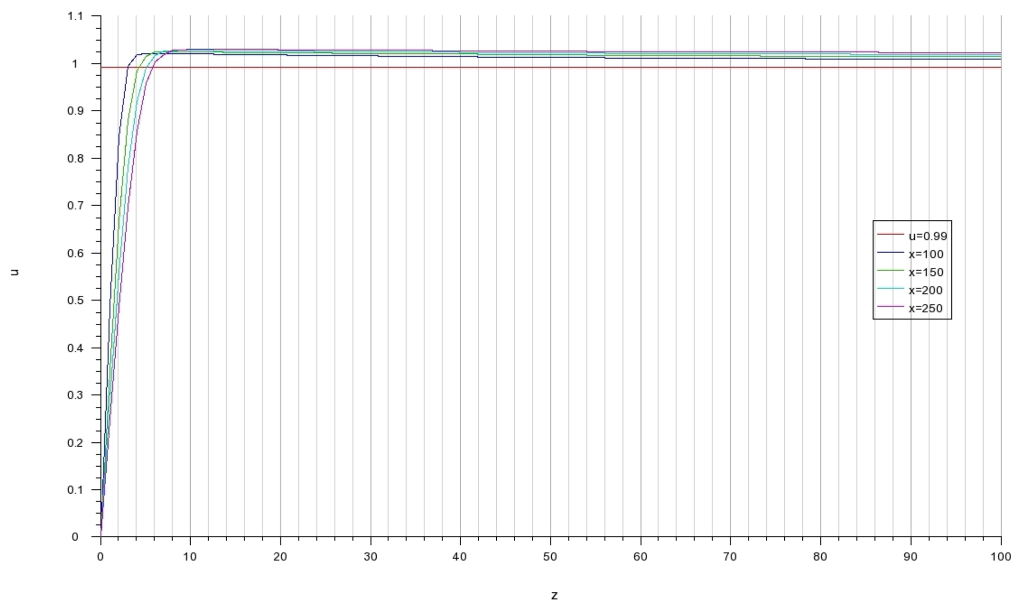
where δ is the theoretical boundary layer height.

This was plotted for the four velocity profiles at $x = (100, 150, 200, 250)$. All graphs have a truncated range in the vertical position, z, for greater legibility of features. As the parabolic profile is only defined up to a wall distance of $z = 1$, these profiles are only plotted up to that value.

One feature, clearly seen in fig. 2.3(a) at the lower viscosity is that the theoretical profile is almost identical to the calculated profile at $x = 100$ but that the velocities according to the parabolic profile are progressively less than those calculated by the model. Another feature is most prominent at the highest viscosity in figure 2.4(b). While the theoretical profiles are always monotonically increasing towards the free-stream velocity (and then by implication stay constant), the model profile overshoots and then reduces to a contant free-stream velocity, which is always slightly greater than the inflow velocity and increases with distance from the leading edge. While the increase of the final free-stream velocity above u_0 is consistent with the theory in section 2.3, the velocity bulge separating the boundary layer from the free stream is somewhat surprising and will be addressed in section 2.6.

2.5.2. COMSOL, fixed mesh

Figure 2.5 shows the displacement thickness of the developing boundary layer against the distance from the leading edge for a selection of Reynolds numbers. As expected, the boundary layer is thicker for smaller values of the Reynolds number, but a number of features deviate from the behaviour expected by the theoretical analysis in section 2.3. At the lowest values of the Reynolds number, the growth of the boundary layer is not continuous but seems to reach a plateau before a further increase terminated by a final drop. The plateau is



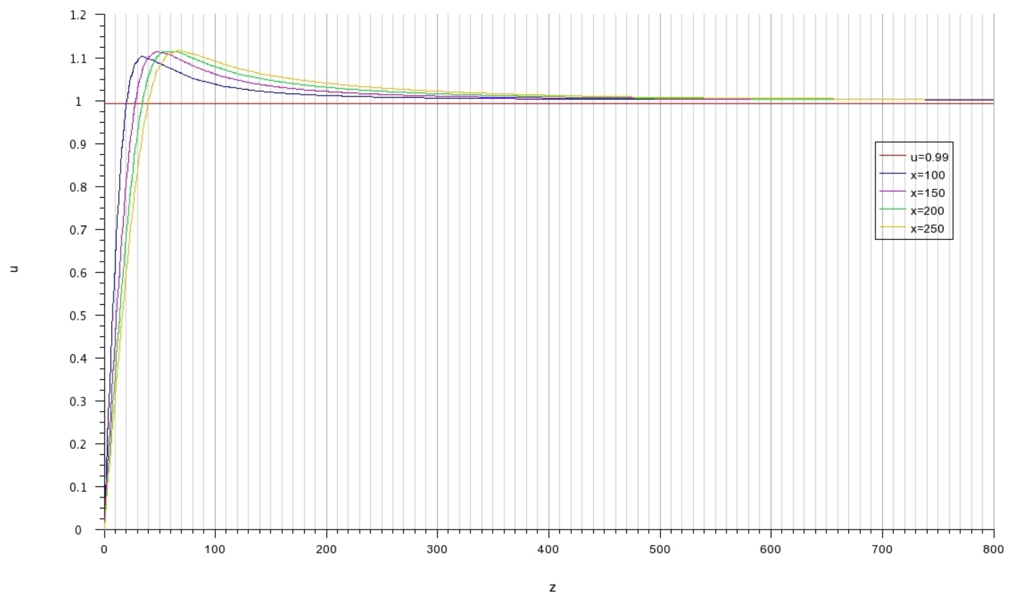


Figure 2.4: Comparison of vertical profiles of the streamwise velocity from the 3D adaptive model with the parabolic approximation of the standard laminar boundary layer. (a) Case 1 with $\nu = 0.01$ and (b) Case 3 with $\nu = 1.0$.

a robust feature of the solution for all meshes used. The final drop may be an artefact of the uniform outflow conditions at $x = 1$.

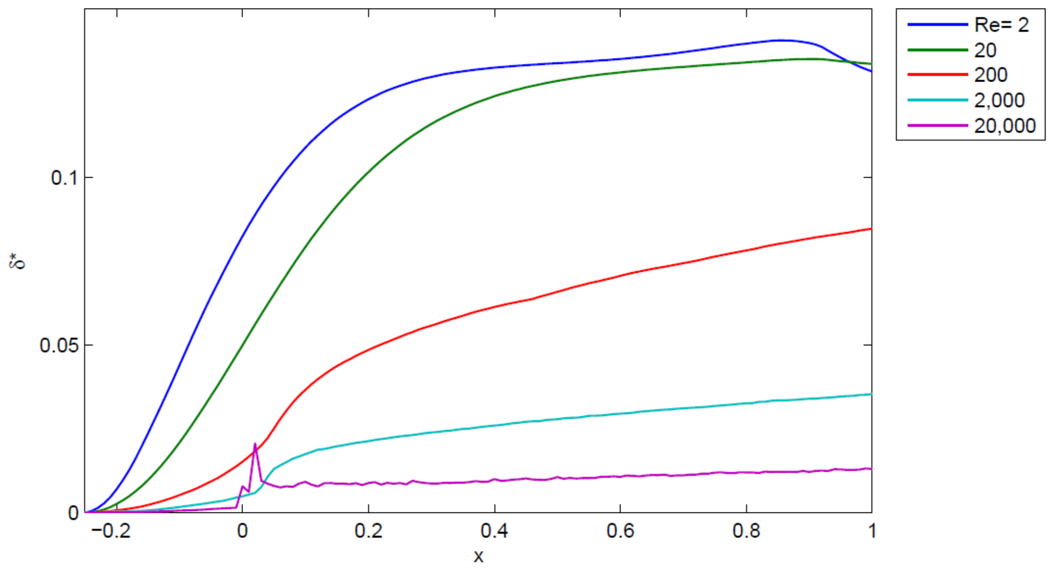


Figure 2.5: Development of the displacement thickness, calculated by the 2D fixed-mesh model for a selection of Reynolds numbers.

A further point to note is the fact that the presence of the plate has a significant upstream influence on the boundary layer. Even for relatively high Reynolds numbers, the flow experiences a slowing down near the lower boundary and a consequent acceleration in the flow interior as early as the left boundary. The generic shape of the boundary layer seems to be an initial slow growth, followed by a rapid increase which appears consistent with a square-root growth, and later on either by a progressively slower growth of the boundary layer or an approach to a limiting depth.

Another point to note is the spike of the displacement thickness at $Re = 2 \times 10^4$. This is clearly an artefact of the model resolution, as integrations at the higher resolution were characterised by a smooth boundary layer growth for Reynolds number up to $Re = 10^5$. Beside the insufficient resolution at the leading edge, the overall solution at $Re = 2 \times 10^4$ was insensitive to the resolution, including the evolution of the boundary layer further downstream from the leading edge. At higher Reynolds number still, $Re = 5 \times 10^4$ and

$Re = 10^5$, the expected boundary layer was comparable to the mesh size at the lower resolution. As a result, the thickness at the trailing edge approaches a limit given by the model resolution, as is demonstrated by figure 2.6 which shows the displacement thickness at the trailing edge against the Reynolds number for the Blasius layer, the theoretical model, and the 2D fixed-mesh Finite Element model at both levels of resolution. The Blasius solution, given by the dotted line follows the $\delta^* \propto Re^{1/2}$ law, and the theory developed in section 2.3 follows this largely although it gives a slightly deeper layer above a Reynolds number of $Re = 20$. The clear deviation from the Blasius layer is at low Reynolds numbers, where the boundary layer is thick enough to fill a substantial portion of the fluid depth, and therefore feels the influence of the upper boundary. The results from the Finite Element model are largely similar to those from the theory. Below $Re \leq 1000$, the computational model underpredicts the boundary layer and approaches a limiting value which is about a third of the channel depth while the channel depth itself is the the limiting value given by the theory. The stronger limiting effect of the finite channel depth is also reflected in the development of the boundary layer for $Re = 20$ and even more for $Re = 2$ in figure 2.5 with the flattening of the curve.

Since the integrations of the adaptive model showed a velocity bulge at the top of the boundary layer (cf fig. 2.3), the velocity profiles from the 2d model are also analysed and compared against the Blasius solution. Figure 2.7 shows two example profiles (dash-dotted lines) , normalised by the maximum velocity and the point at which this is reached. Superimposed on these is the Blasius solution In all cases shown here, the velocity reduces again above the maximum velocity. At the higher Reynolds number, which is in a region where the final displacement thickness is close to the theoretical value (cf fig. 2.6), the reduction is very slight. In this case, the velocity profile is virtually indistinguishable from the Blasius solution. At the lower Reynolds number, however, which is in a region where the channel depth begins to limit the boundary layer, according to figure 2.6, there is a pronouned jet above the

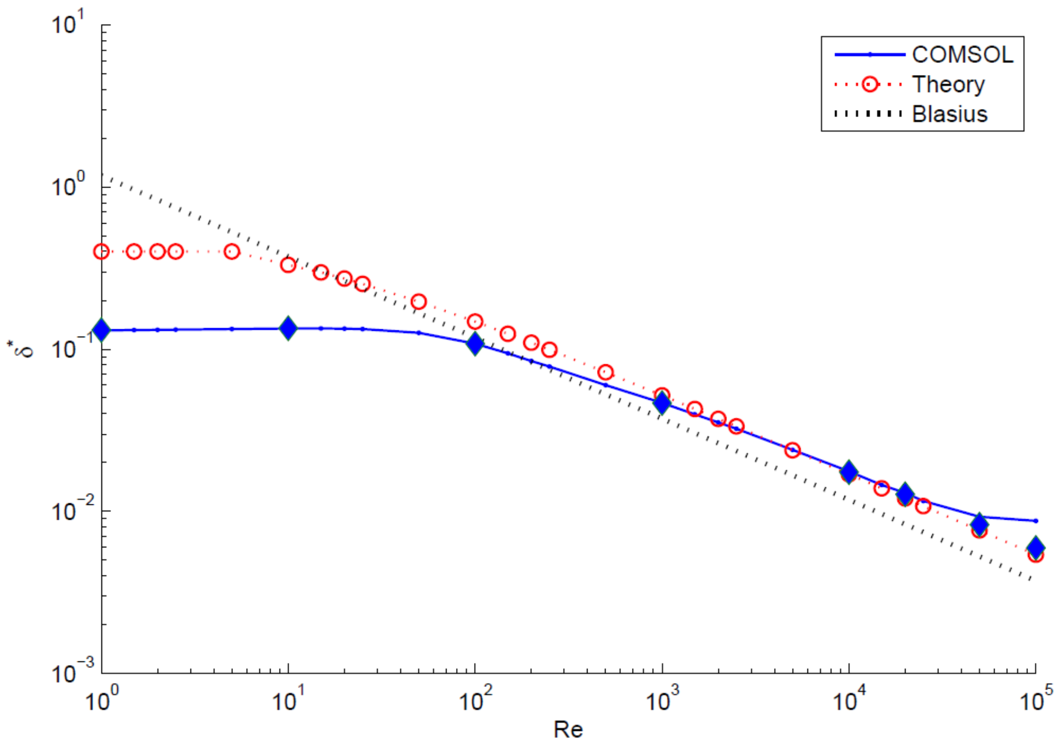


Figure 2.6: Comparison of the displacement thickness at the trailing edge of the plate against Reynolds number. The fluid depth is $H = 0.4$. The dots are the solutions of the finite-element model, the open circles from the theoretical model, and the dotted line shows the standard Blasius layer.

boundary layer, and the velocity profile is noticeably different from the Blasius solution.

It is now possible to cross-validate the different models with each other and against the simple theory. It is safe to assume that the two computational models are fully independent since the software packages were developed entirely independently, and also their respective configurations in this case are very different; one model was a 2D steady model with a fixed mesh while the other was a 3D time-dependent model with an adaptive mesh. In the first instance, a comparison of the general velocity profies and their evolution in figures 2.3 and 2.7, demonstrates that both models result in, at least qualitatively, similar velocity fields which are largely consistent with the theoretical laminar boundary layer, especially if the boundary layer thickness is only a small fraction of the channel depth.

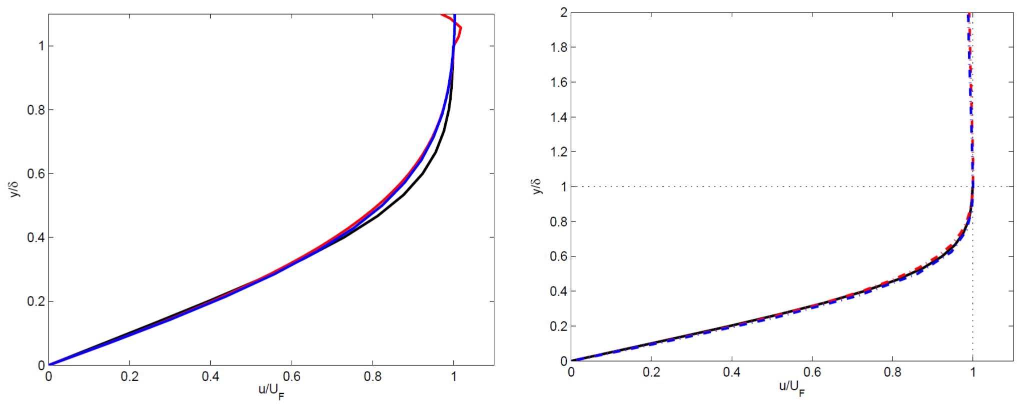


Figure 2.7: Velocity profile from integrations of the high-resolution model at a Reynolds number of (a) $Re = 100$ and (b) Re = 1000, respectively, at positions $x = 0.2$ and $x = 0.9$ (both dash-dotted). The solid curve is the standard Blasius layer.

In contrast to the theory but internally consistent, both types of models do not show the standard approach to a free stream velocity which then remains constant over the remainder of the channel. Instead, the velocities reach a value larger than the expected free-stream velocity and then reduce to a final free-stream velocity which is between the inflow velocity and that maximum. It is argued here that the bulge is not an artefact but a physical effect caused by the finite fluid depth. As the boundary layer develops, the average velocity in the fluid interior has to increase to maintain the mass flux through the channel. Streamlines within and near the boundary layer are displaced upwards where the displacement becomes progressively less with distance from the top of the boundary layer. As a result, the fluid near the top of the boundary layer is accelerated most and the far field is accelerated least. In the final balance, this then could lead to a velocity maximum at the top of the boundary layer with a velocity gradient which vanishes locally in the y-direction (but not in the x-direction)

Since the boundary layer theory uses a constant free-stream velocity to scale the velocities and identify the boundary layer thickness, it is debatable as to whether the free-stream velocity or the velocity maximum should be used in the analysis of the results. Using the complementary way of displaying the profiles, and comparing them to a theoretical thickness assuming a constant free-stream velocity in fig. 2.3, but to a profile scaled against the value and location of the maximum velocity in fig. 2.7, demonstrates that it is the velocity maximum which should be taken for the scaling, at least at sufficiently thin boundary layers where theory and model are indistinguishable within the boundary layer. The close correlation between the Blasius profile and the model results breaks down when the displacement thickness reaches about a third of the channel depth. If we remember that the actual boundary layer thickness, δ, is related to the displacement thickness of the Blasius layer by $\delta^* = 0.344\,\delta$, it becomes clear that the boundary layer fills the entire domain, and the assumption of the existence of a top of the boundary layer in a free stream with zero velocity gradient in the y-direction is no longer valid. As a result, one should not expect the real or computed flow to follow that theory.

2.6. Comparison between Fluidity and COMSOL

The displacement thickness for the three cases used in the 3D adaptive model against distance from the leading edge of the plate is shown in figure 2.8 by the dots. The solid lines in the figure are the results from the 2D fixed-mesh model using exactly equivalent conditions. All features of the developing boundary layer are reproduced by both models, including the initial slow increase upstream of the plate, followed by the sharp increase at the leading edge. Even the levelling out, followed by a final peak and drop in the thickness at low Reynolds numbers is reproduced in Case 1 of the adaptive model. As was observed, the local solution at the leading edge is sensitive to the model resolution but the global solution is not noticeably affected by it. This is also reflected here in the fact that the largest differences between the two models are found in that region.The development of the boundary layer upstream of the boundary is presumably a result of a pressure gradient due to the friction force at the plate and the required acceleration of the fluid outside the boundary layer.

2.7. Conclusions

Through a cross-validation of two independent computational models and validation against an extension of standard theory, it has been shown that the Fluidity finite-element model package, using an hr-adaptive unstructured mesh, provides a reliable and accurate representation of the developing laminar boundary layer over a flat plate. As part of the analysis, it was shown that two different Dirichlet boundary conditions at adjacent boundaries are likely to lead to the failure of the model as the adaptivity would try to increase the local resolution at the discontinuity of the imposed velocities resulting in rapidly increasing velocity gradients. The approach to bypass this problem was here to avoid such a case by adding an entry length with von-Neumann conditions adjacent to the Dirichlet conditions.

While providing this analysis within a wider programme to validate this

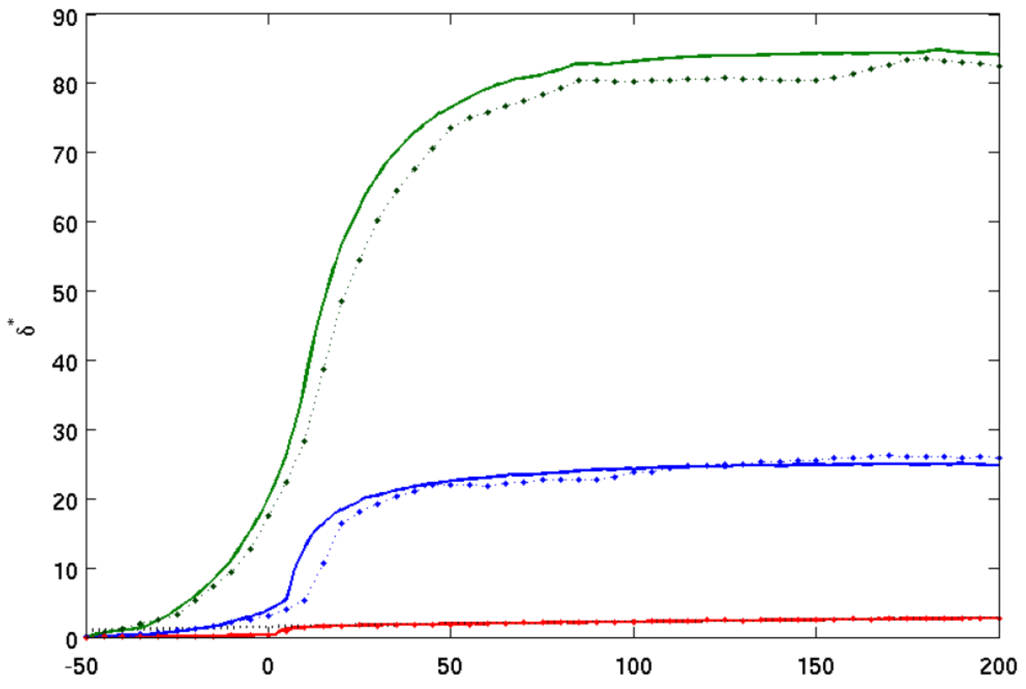


Figure 2.8: Displacement thickness against position on the plate from the Fluidity model (diamonds and dotted lines) and the COMSOL model (solid lines). From top to bottom: experiment numbers 1, 2, and 3 as listed in table 2.1.

model for a general range of applications, a new algorithm to interpolate data from an unstructured mesh was developed and tested. In this boundary layer case, with a highly anisotropic mesh, this algorithm could reduce interpolation errors by up to an order of magnitude compared to the methods used routinely in the literature.

Chapter 3

Existing turbine models

3.1. The actuator disc

For more than a century, actuator discs have been used to analyse fluid flow through propellers, and so turbines. Initially devised as a one-dimensional theory based on momentum arguments by Froude, it was later expanded upon by Glauert [34], who introduced blade element momentum theory. Then primarily solved by analytic methods, it has since been used incorporated into fully three dimensional Navier-Stokes simulations using what are known as vorticity-velocity methods and applying them to wind turbines [69] [55]. The general theory will be covered first in this section, followed by recent extensions and enhancements and their applications in wind turbine modelling.

3.1.1. Basic theory

We can think of the actuator disc as an infinitely thin disc that sits perpendicular to the flow, and can either model the injection of energy in the fluid, as is the case with propellers, or the extraction of energy, as would happen with a turbine. To this end, the model has been deployed to model axial-flow wind turbines and their wakes, based upon previous work by Glauert.

The actuator disc extracts pressure energy rather than kinetic energy, since the step-wise nature of the extraction of the latter requires infinite forces for this to happen. As the fluid approaches the disc, it begins to slow down due to the increase in pressure upstream of the the turbine. The streamtube, a circular arrangement of streamlines around the turbine, expands before and continues to do so downwind of the turbine. Inside this streamtube, the conservation of

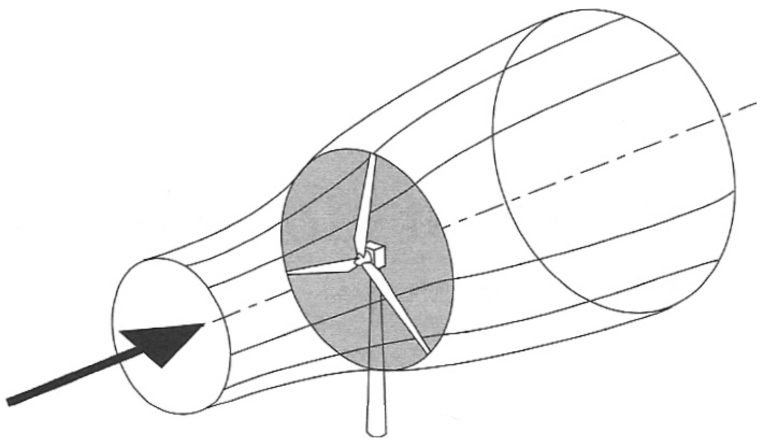


Figure 3.1: 3D streamtube in actuator disc theory (courtesy of Wind Energy Handbook)

mass applies, so that if we assume incompressible flow we can write

$$\rho A_0 u_0 = \rho A_d u_d = \rho A_w u_w \tag{3.1}$$

where

- A_0 and u_0 represent the cross-sectional area of the streamtube and speed of the fluid in the freestream
- A_d and u_d represent the cross-sectional area of the streamtube and the speed fo the fluid, at the actuator disc
- A_w and u_w represent the cross-sectional area of the streamtube and fluid speed downstream, in the wake

By removing energy from the fluid, the actuator disc decreases the flow velocity proportionally to the freestream velocity. We can say that

$$u_d = u_0(1 - a) \tag{3.2}$$

where a is the axial flow induction factor.

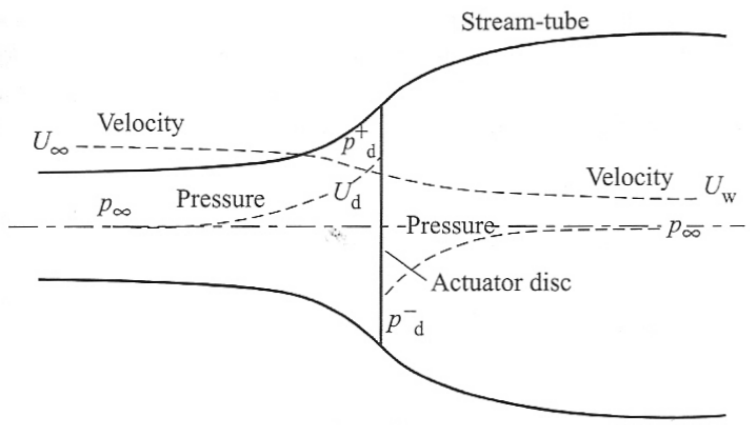


Figure 3.2: Side view of streamview (courtesy of Wind Energy Handbook)

3.1.2. Simple momentum theory

We can write rate of change in momentum in the fluid as

$$\frac{dp}{dt} = (u_0 - u_d)\rho A_d u_d \tag{3.3}$$

This loss comes from the kinetic pressure difference between either side of the disc, $(P_d^+ - P_d^-)$, and so $\frac{dp}{dt}$ becomes

$$\frac{dp}{dt} = (P_d^+ - P_d^-)A_d = (u_0 - u_d)\rho A_d u_0(1-a) \tag{3.4}$$

Bernoulli's theorem gives us

$$\frac{1}{2}\rho u_0^2 + P + \rho g h = K \tag{3.5}$$

where g is the acceleration due to gravity, h is the height, and K is a constant.

This can be shown [16] to give

$$\frac{1}{2}\rho(u_0^2 - u_w^2)A_d = (u_0 - u_w)\rho A_d u_0(1-a) \tag{3.6}$$

Hence

$$u_w = (1-2a)u_0 \tag{3.7}$$

When energy is extracted from the air, and power generated, a backthrust to the forward flow is generated. This is

$$F = (P_d^+ - P_d^-)A_d = 2\rho A_d u_0^2 a(1-a) \tag{3.8}$$

Thus the extracted power is

$$PW_{ex} = Fu_d = 2\rho A_d u_0^3 a(1-a)^2 \tag{3.9}$$

3.1.3. Power

The amount of power available in the wind is defined as

$$PW_0 = \frac{1}{2}\rho A_d u_0^3 \tag{3.10}$$

A measure of the efficiency of the turbine, the power coefficient C_p, is

$$\begin{aligned} C_p &= \frac{PW_{ex}}{PW_0} \\ &= 4a(1-a)^2 \end{aligned} \tag{3.11}$$

There is a theoretical limit to the amount of power that can be extracted in the actuator disc model, known as the Betz limit. This maximum of C_p occurs when

$$\frac{dC_p}{da} = 0 \tag{3.12}$$

Chain rule differentation gives us

$$\frac{dC_p}{da} = 4(1-a)(1-3a) = 0 \tag{3.13}$$

From which we can find the maximum value C_p:

$$C_{p,max} = \frac{16}{27} \tag{3.14}$$

This means the theoretical maximum efficiency of an actuator disc model is almost 60%.

3.1.4. The thrust coefficient

Another co-efficient of use is the thrust coefficient, C_T - normalised backthrust

$$\begin{aligned} C_T &= \frac{\rho u_d A_d (u_0 - u_w)}{\frac{1}{2}\rho u_0^2 A_d} \\ &= 4a(1-a) \end{aligned} \tag{3.15}$$

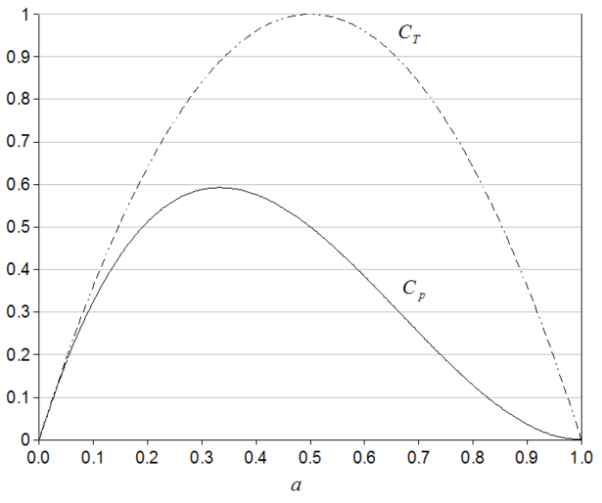


Figure 3.3: Graph of a versus C_T and C_p

When $a >= \frac{1}{2}$ the velocity of the fluid in the wake can become zero or negative; this needs to be corrected.

3.1.5. Blade element momentum theory and wake rotation

The above model assumes that the load on the rotor is evenly spread about the disc. In real rotors however, this is not the case, and so this leads us to Glauert's extension of Froude and Rankine's theory, which divides up the actuator disc into rings, and so the streamtubes become annular as can be seen in figure 3.4.

The rings are considered to be radially-independent of each other, which allows them to be dealt with individually. A further assumption is made in that the flow is considered to be symmetric about the rotor axis.

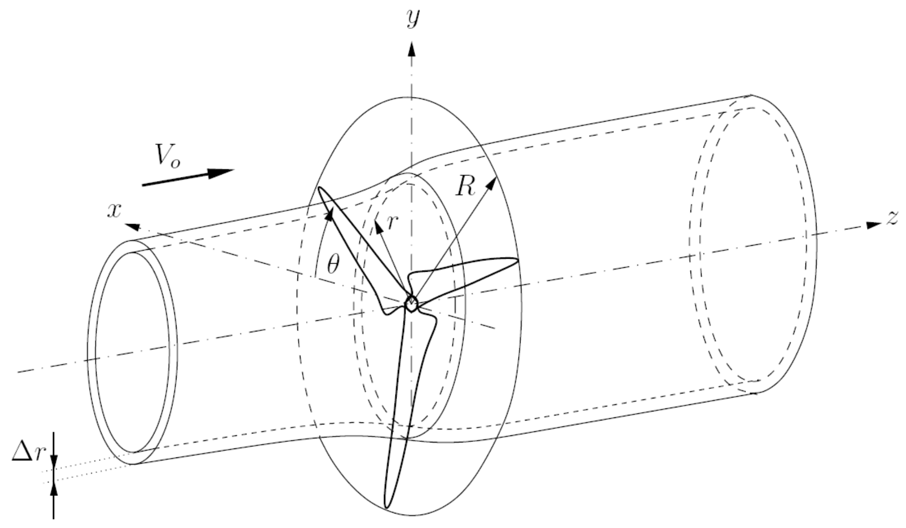


Figure 3.4: Streamtube through a rotor (Courtesy of Mikkelsen)

At this point we also add wake rotation, modelling the reaction of the fluid to the torque exerted by the rotor. We can say that immediately downstream, the tangential (orbital) velocity of the wake is

$$v_{tan} = 2\omega_{FL} r a' \tag{3.16}$$

where ω_{FL} is the angular velocity of the fluid at radius r from the rotor axis, and a' is the tangential flow induction factor.

Considering a ring at r with thickness δr, we can write the torque as

$$\delta\tau = \rho \delta A_d u_0 (1 - a) v_{tan} \tag{3.17}$$

where δA_d is the area of the annular ring. From this we can write the extracted power as

$$\delta PW = \delta\tau\omega_{FL} \tag{3.18}$$

From equation 3.10 we can also deduce

$$u_0^2 a(1 - a) = \omega_{FL}^2 r^2 a' \tag{3.19}$$

And furthermore, that

$$\frac{dC_P}{dr} = 8(1-a)a'\left(\frac{\omega_{FL}^2}{u_0^2 R_T^2}\right) r^3 \tag{3.20}$$

By differentiating equation 3.19 with respect to a', and setting $\frac{da}{da'} = 0$, we can find the values of a and a' that give the maximum power coefficient. These values are

$$a = \frac{1}{3} \tag{3.21}$$

$$a' = \frac{a(1-a)u_0^2}{r^2\omega_{FL}^2} \tag{3.22}$$

By plugging these into equation 3.20 and integrating with respect to r, between $r = 0$ and $r = R_T$, we find that

$$C_p = 4a(1-a)^2 = \frac{16}{27} \tag{3.23}$$

Which is an identical result to equation 3.14.

3.1.6. Aerofoils

The notion of blade element theory introduced in the previous section can be enhanced with the concepts of lift and drag forces acting on the fluid and aerofoil wing.

The force per unit span is defined as

$$\underline{F} = \underline{L} + \underline{D} \tag{3.24}$$

where $\underline{L}$ and $\underline{D}$ represent the lift and drag components respectively. The axial and orbital components can be expressed as

$$F_{axial} = Lcos\phi + Dsin\phi \tag{3.25}$$

and

$$F_{orbital} = Lsin\phi - Dcos\phi \tag{3.26}$$

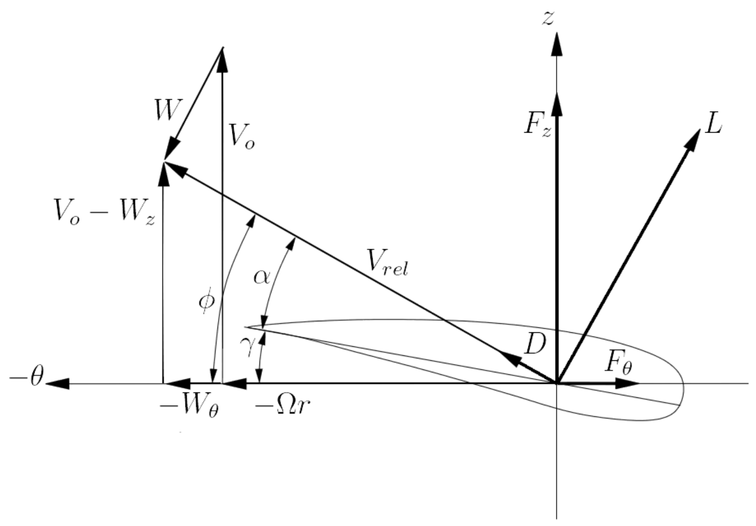


Figure 3.5: Cross-section of aerofoil (courtesy of Mikkelsen)

L and D can be obtained with the help of tabulated aerofoil data. F_{axial} and $F_{orbital}$ can be incorporated into the actuator framework using iterative methods for solution according to [55]. These need not be gone into in more detail here, however the point is made: the aerodynamic effect of turbine blades and the surface incidence to the flow are not effects that can be ignored.

Whilst the thesis model does not explicitly mention incidence angle, it does take into account the *effect* of the blade orientation, surface distribution and aerodynamic properties (such as lift) on the turbine's performance and the flow; and that properties such as blade angle can be dynamic. More on this is described in chapter 4.

3.2. Further extensions to actuator theory

3.2.1. The $\Psi - \omega$ model

This model uses the Navier-Stokes equations for numerically solving the actuator disc induced flow. It has the benefit of introducing viscosity; actuator theories discussed previously assume an inviscid fluid. Also, being time-dependent it is able to solve unsteady flow, typical of heavily-loaded turbines. Developed by Sørensen and colleagues [69] [68], it also forms the basis of Mikkelsen's work [55].

Rather than solving in cartesian co-ordinates using velocity and pressure as primitive variables, it switches to vorticity-velocity, or $\underline{\omega} - \underline{u}$ variables in cylindrical co-ordinates (r, θ, z) (note: for this section, ω denotes vorticity not angular velocity) . The curl operator $(\nabla\times)$ is applied to the Navier-Stokes equations to give

$$\frac{\partial \underline{\omega}}{\partial t} + \nabla \times (\underline{\omega} \times \underline{u}) = \nu \nabla \times (\nabla \times \underline{\omega}) + \nabla \times \underline{f} \tag{3.27}$$

$$\nabla \times \underline{u} = \underline{\omega} \tag{3.28}$$

$$\nabla . \underline{u} = 0 \tag{3.29}$$

where ν is the kinematic viscosity, and $\underline{f}$ represents body forces applied to the fluid.

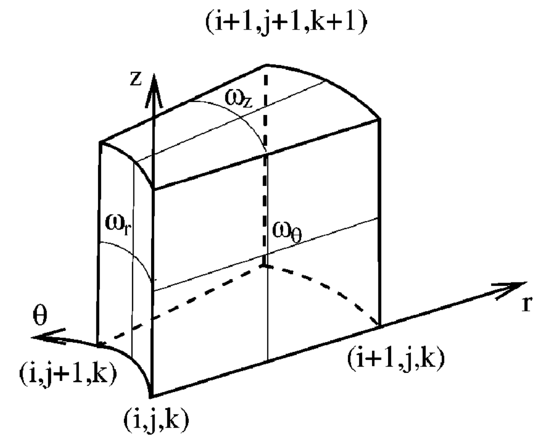


Figure 3.6: A finite segment in a $\underline{\omega} - \underline{u}$ model (courtesy of Sørensen 1998)

This is commonly deployed using a finite-difference scheme, divided into regular $(\delta\theta, \delta r, \delta z)$ segments of 'cheese cake' slices, as seen in figure 3.6. Time integration is by both Adams-Bashforth for convective and Crank-Nicolson for diffusive terms.

According to Mikkelsen, results compare reasonably with one-dimensional theory up to high levels of thrust, but for $C_T >= 1$ results deviate from this towards agreeing with experiments using a Nordtank stall-regulated turbine. This is surprising, given that the effects of turbulence viscosity are not being explicitly modelled. Moreover, Mikkelsen does not detail the downstream wake for distances greater than ≈ 3 turbine diameters, so it is difficult to see whether the model will hold to the rule-of-thumb of ≈ 20 diameters wake recovery, or that the velocity deficit profile that matches experiment.

Lastly, cylindrical co-ordinates reduce the feasibility of deploying this model in anything other than a wind-tunnel style environment; at large values of r resolution would become an issue as the finite segments grow too large. Clearly, a finite-element model capable of irregular meshes and based on cartesian co-ordinates, might present some benefits when attempting to simulate a wind turbine in more realistic environments involving greater scales (urban buildings, orography).

3.3. Full 3D model simulation

One impressive paper both demonstrates the complexity of modelling a wind turbine in its entirety and its current impractibility for large-scale simulation. Wußow et al [75] demonstrate a finite-volume model running under commercial CFD software that comprises of a moving mechanical model of the rotor, replete with nacelle (hub) and tower.

This paper was written in 2007 with, at the time of writing, recent computing technology to hand; despite this, the domain extends no further than 1 turbine rotor diameter upwind, and 3 downwind for a 10 minute simulation. Thus, for domains that may extend over 1km downstream ($>$ 20 diameters for large HAWTs) this in-depth level of modelling may not be appropriate for farm simulation – 4 million cells! – a cruder model is needed, which manages to capture both the response of the turbine to wind conditions, and of the wake to the turbine's performance. Nonetheless, it is a useful source for turbulence characteristics, particularly in its use of a von-Karman model for turbulence at the inflow boundary.

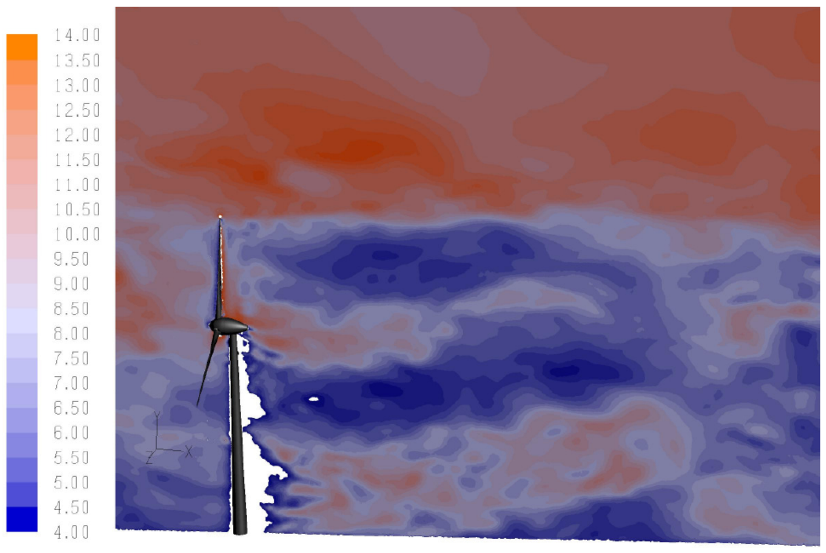


Figure 3.7: Velocity magnitude plot for $\overline{u}_{in} = 10ms^{-}1$ (courtesy of Wusow)

3.4. Wake modelling for wind farms

Attempts at simulating the full length (near to far) of wind turbine wakes have an interesting history, perhaps because of the necessary restrictions placed upon them by the limited computational power of the time, especially in wind farm configurations with many turbines. In their review, Crespo et al [20], detail both single and multiple wake models, which tend to employ a semi-emperical approach, using experimental data and theory to formulate analytical solutions for wake profiles.

Emperical laws for single wakes have been shown to agree well to measurements [6], according to Crespo. For multiple wakes, the simplest assumption is that of linear superposition of velocity deficits from different turbines as first suggested by Lissaman [47]; this led to artifically long wakes. Smith and Taylor [67] discovered that for two machines in a row, the wake velocity of the downstream turbine recovered more quickly. Crespo suggests that higher levels of turbulence downstream, generated by the upstream turbine, contribute to greater momentum diffusion in the wake of the second turbine. Fig. 3.8 shows how a variety of models fare against experiment, clearly demonstrating the over-long wakes of models using linear wake superposition.

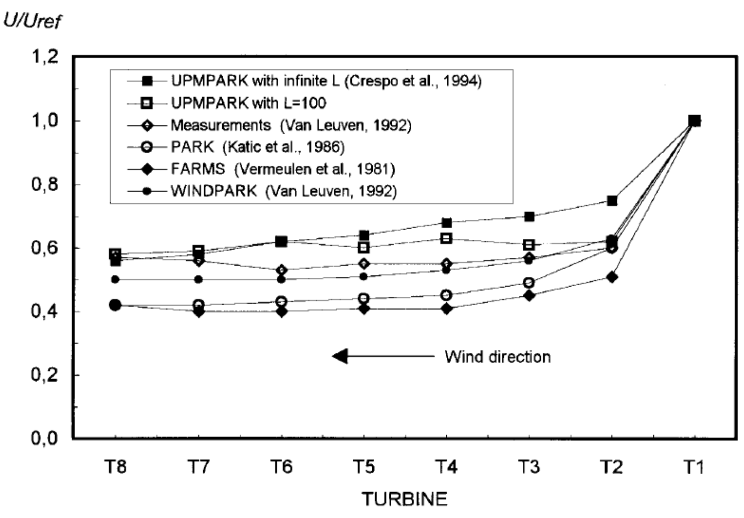


Figure 3.8: Velocity deficits behind downstream turbines: a comparison between various wake models and experiment (see Crespo et al [20] for details)

In more recent developments, Barthelmie et al [11] have been making the first steps toward full computational fluid dynamic simulations of wind farms with simple land features. This thesis will present a turbine model which is intended to be used in such a scenario.

Chapter 4

Turbine model design

4.1. Rationale

The thesis turbine model represents a departure from current schemes, as it neither deals with rigid bodies moving through fluid, nor boundary conditions. Rather, it applies body forces to the fluid over a cylindrical volume within the fluid, to affect power extraction, wake rotation, and turbulence.

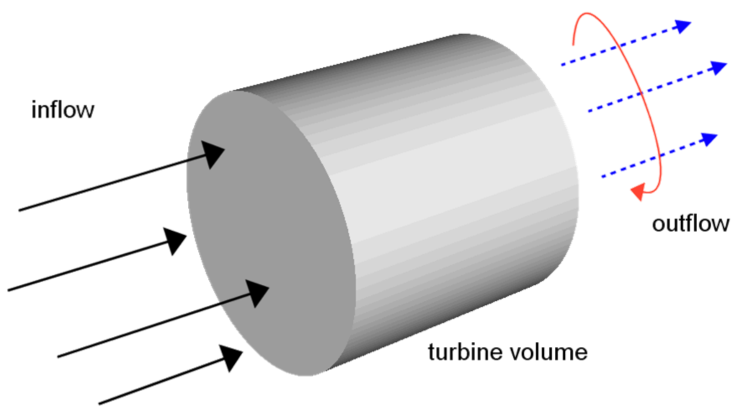


Figure 4.1: View of the turbine model volume

There are several reasons for this approach:

1. **Generality**. We make few assumptions about the geometry of the blades, other than that they rotate around a horizontal axis in a well-defined cylindrical volume.

2. **Complexity**. Ad-hoc boundary conditions can be difficult to implement, particularly in noncommercial code

3. **Computational cost**. In many models finite difference, volume or element boundary conditions require extra computing cycles as opposed to non-boundary grid nodes.

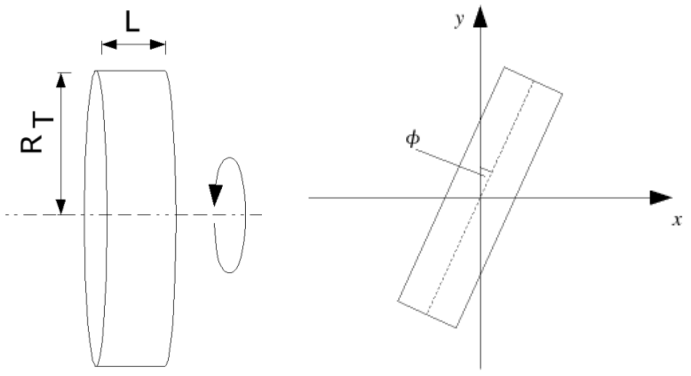
4. **Granularity**. By rejecting the notion of boundaries on a disc or turbine blade, we can control what happens to the fluid as it passes through volume occupied by the turbine and is acted upon by it, without losing generality. As there are no boundaries, this allows the resolution of the model to be entirely arbitrary.

4.2. Physical properties

Assume that we have a single turbine that is modelled as a shallow cylindrical volume, in which the fluid velocity is affected. The cylinder's rotational axis of symmetry is always perpendicular to the z-axis, and the turbine has the following global properties:

- position $\underline{x}_T$
- orientation angle ϕ
- turbine radius R_T
- hub radius R_H
- radius at start of tip section R_{tip}
- tip width fraction w
- length L
- flow factor f
- tip speed ratio λ
- power extraction efficiency parameter Q
- net blade solidity B and associated local solidity β

- hub local solidity β_H
- effective net blade solidity B' and effective local solidity β'
- blade pitch parameter α
- angular velocity of the fluid at the turbine outlet ω_{FL}
- angular velocity of the fluid with no power extraction (free-wheeling) ω_{NP}
- angular velocity of the (virtual) turbine blades ω_T
- power extracted by the turbine PW_{ex}
- power coefficient c_p
- u_{cutin} the cut-in flow speed of the turbine
- u_{cutout} the cut-out flow speed of the turbine
- Length of the tip section causing turbulence L_{tip}
- turbulence intensity at tip section of blade Ti



(a) Dimensions of the model volume (b) Orientation of the model

Figure 4.2: Specification of the model in three dimensions

The following denote values occuring at optimum performance of the turbine (hence the subscript *opt*):

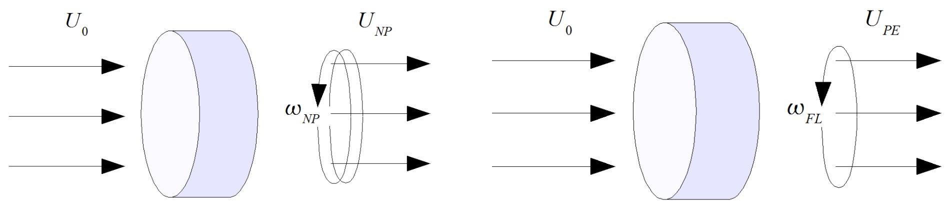
- turbine angular velocity $\omega_{T,opt}$
- effective solidity B'_{opt}
- free-stream fluid speed $u_{0,opt}$
- power coefficient $c_{p,opt}$
- power coefficient at cut-out fluid speed $c_{p,cutout}$
- turbulence intensity at optimum efficiency - Ti_{opt}

Also assume that the turbine rests in an incompressible and viscous fluid that has the following attributes:

- a fluid velocity field $\underline{u}(\underline{x}, t)$
- density ρ
- molecular dynamic viscosity μ

4.3. Method overview

The turbine model technique can essentially be thought of as a comparison between two turbines. One is free-wheeling, only converting linear momentum to angular momentum in the downstream wake; the other does the same, but also extracting power from the fluid flow.



(a) Turbine A, with no power extraction (freewheeling)

(b) Turbine B, a power-extracting turbine

Figure 4.3: Comparison between two turbine cases used in the model

Initially, we will only consider axial flow turbines with the intake surface perpendicular to the flow, and of constant solidity. The freestream velocity flows in a positive direction parallel to the x-axis, and is of magnitude u_0.

The process is essentially as follows:

1. ω_{NP} is calculated from the mean speed $\bar{u}_{in}$ of the fluid entering turbine A (this is likely to be less than u_0),
2. The resulting orbital components of the acceleration are calculated,
3. Based upon these, the backthrust in the fluid is found,
4. ω_{FL} is determined as a fraction of ω_{NP},
5. Axial and orbital forces are updated using ω_{FL},
6. The extracted power PW_{ex} is determined, and
7. The body forces are applied to the fluid.

4.3.1. Free-wheeling angular velocity

First, we define ω_{NP}, the angular velocity of the fluid with a free-wheeling turbine. To do this, first we assume that all of the forward kinetic energy of the fluid is converted into rotational kinetic energy, ie. that

$$KE_{in} = KE_{rotate} \tag{4.1}$$

or

$$\frac{1}{2}M_T\bar{u}_{in}^2 = \frac{1}{2}I\omega_{NP}^2 \tag{4.2}$$

Where M_T is the mass of fluid in the turbine. We can write I, the moment of inertia for a cylinder, as

$$I = \frac{1}{2}M_T R_T^2 \tag{4.3}$$

Therefore

$$\begin{aligned} M_T\bar{u}_{in}^2 &= \frac{1}{2}M_T R_T^2\omega_{NP}^2 \\ \bar{u}_{in}^2 &= \frac{1}{2}R_T^2\omega_{NP}^2 \end{aligned}$$

$$\omega_{NP} = \sqrt{2}\left(\frac{\bar{u}_{in}}{R_T}\right) \tag{4.4}$$

Now we assume that only a fraction of the linear momentum is converted to angular momentum, and that this fraction depends upon two things: the blade pitch parameter α and the turbine net solidity B, a measure of how solid the turbine appears to the oncoming flow. As this will both be linear in u, the above becomes

$$\omega_{NP} = \sqrt{2}\alpha B\left(\frac{\bar{u}_{in}}{R_T}\right) \tag{4.5}$$

Then we add the flow factor f, which parameterises for aerodynamic/hydrodynamic effects in real turbines, such as lift. This can be thought of as analogous to the tangential flow induction factor introduced by Sharpe [66].

$$\omega_{NP} = \sqrt{2}f\alpha B\left(\frac{\bar{u}_{in}}{R_T}\right) \tag{4.6}$$

However, due to the lack of boundary conditions in the model, calculation of $\bar{u}_{in}$ can be problematic. So, assuming that the cylinder is thin enough, a valid approximation would be that mean speed of the flow inside the cylinder is close to the mean speed entering it - in other words, $\bar{u}_{in} \approx \bar{u}$. Thus the above equation becomes

$$\omega_{NP} \approx \sqrt{2}f\alpha B\left(\frac{\bar{u}}{R_T}\right) \tag{4.7}$$

4.3.2. Calculating the power output

One option for calculating the power extracted by the turbine is through Bernoulli's equation. However, this is not a wise choice for estimating power, for two reasons:

1. Bernoulli's equation deals only with laminar flow. Any effective wind or marine turbine model must deal with turbulence, and so the loss to turbulent kinetic energy must be divorced from that lost to the turbine.

2. By its very nature, Bernoulli's equation can only examine the *net* energy extraction from the flow. To examine the effects of turbines on the performance of similar devices downstream, greater granularity is needed.

Instead, what we do is introduce a second angular velocity, ω_{FL}, which is the angular velocity of the fluid when the same turbine extracts power from the flow. This is defined as

$$\omega_{FL} = (1 - Q)\omega_{NP} \tag{4.8}$$

where Q represents the fractional loss of angular momentum due to power extraction. We limit Q to $0 \leq Q \leq 1$.

If we first write the difference in rotational kinetic energy between the free-wheeling and power extracting turbine, we get:

$$\begin{aligned} \Delta E &= \frac{1}{2}I\omega_{NP}^2 - \frac{1}{2}I\omega_{FL}^2 \\ &= \frac{1}{2}I(\omega_{NP}^2 - \omega_{FL}^2) \end{aligned} \tag{4.9}$$

The power extracted can be written as

$$\begin{aligned} PW_{ex} &= \frac{dE}{dt} \\ &\approx \frac{\Delta E}{\Delta t_T} \\ &= \frac{1}{2}I(\omega_{NP}^2 - \omega_{FL}^2)\frac{1}{\Delta t_T} \end{aligned} \tag{4.10}$$

Where Δt_T is the time taken for the fluid to travel through the turbine, ie. the time over which the energy extraction takes place. Hence

$$\Delta t_T = \frac{L}{\bar{u}} \tag{4.11}$$

where $\bar{u}$ is the mean speed of the fluid through the turbine. Plugging this in, we get

$$\begin{aligned} PW_{ex} &= \frac{1}{2} I(\omega_{NP}^2 - \omega_{FL}^2)\frac{\bar{u}}{L} \\ &= \frac{1}{4} M_T R_T^2(\omega_{NP}^2 - \omega_{FL}^2)\frac{\bar{u}}{L} \\ &= \frac{1}{4}\rho\pi R_T^4 L(\omega_{NP}^2 - \omega_{FL}^2)\frac{\bar{u}}{L} \end{aligned} \tag{4.12}$$

Therefore

$$PW_{ex} = \frac{1}{4}\rho\pi R_T^4(\omega_{NP}^2 - \omega_{FL}^2)\bar{u} \tag{4.13}$$

4.3.3. Turbine angular velocity

The turbine angular velocity ω_T is distinct from the angular velocity of the fluid ω_{FL}. ω_T ties the behaviour of the model to real turbines. First we need to find $\omega_{T,opt}$, which we can do using $u_{0,opt}$, found from turbine performance charts such as that in figure 4.4.

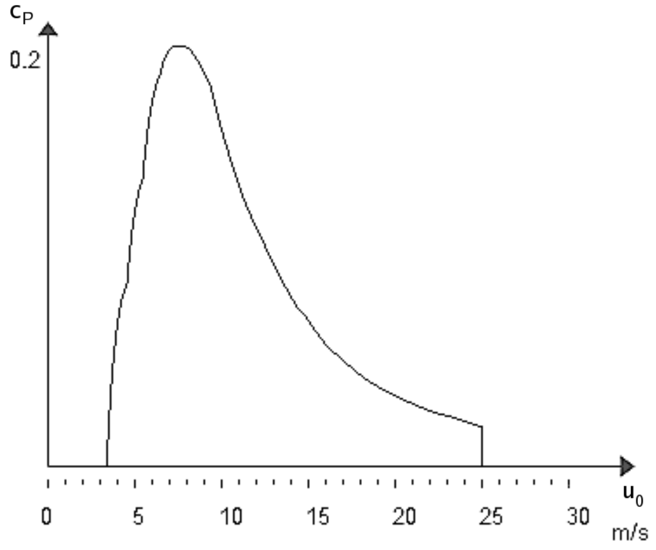


Figure 4.4: Typical performance graph from the Danish wind industry association website

The tip-speed ratio λ is the ratio of the orbital speed of the tip of the turbine blades to $u_{0,opt}$. If we assume λ to be constant, we can write

$$\begin{aligned} \omega_{T,opt} R_T &= \lambda u_{0,opt} \\ \omega_{T,opt} &= \lambda \left(\frac{u_{0,opt}}{R_T} \right) \end{aligned} \tag{4.14}$$

Now we turn to the power output, which we use to couple ω_T to the model. The optimum power extracted is

$$\begin{aligned} PW_{ex,opt} &= c_{p,opt} PW_{freestream} \\ &= c_{p,opt} \frac{1}{2} \rho \pi R_T^2 u_{0,opt}^3 \end{aligned} \tag{4.15}$$

If we then assume that the torque on the turbine shaft due to electromechanical friction and power extraction is linear with ω_T

$$\tau = K_\tau \omega_T \tag{4.16}$$

where K_τ is a constant, then

$$\begin{aligned} PW_{ex} &= \tau \omega_T \\ &= K_\tau \omega_T^2 \end{aligned} \tag{4.17}$$

The constant K_τ can be defined from optimum values

$$K_\tau = \left(\frac{PW_{ex,opt}}{\omega_{T,opt}^2} \right) \tag{4.18}$$

And so

$$PW_{ex} = \left(\frac{PW_{ex,opt}}{\omega_{T,opt}^2} \right) \omega_T^2 \tag{4.19}$$

Therefore the angular velocity of the turbine is

$$\omega_T = \left(\frac{PW_{ex}}{PW_{ex,opt}} \right)^{\frac{1}{2}} \omega_{T,opt} \tag{4.20}$$

4.3.4. Extending solidity

As mentioned in section 4.3.1, rather than just being a measure of the cross-sectional area of the turbine, B also quantifies the effect this cross-section has on the flow. In this subsection, we build upon this idea.

Non-uniform solidity

Previously it was assumed that the solidity - the blade density - was spatially invariant. Looking at a real turbine we can see that this is patently untrue.

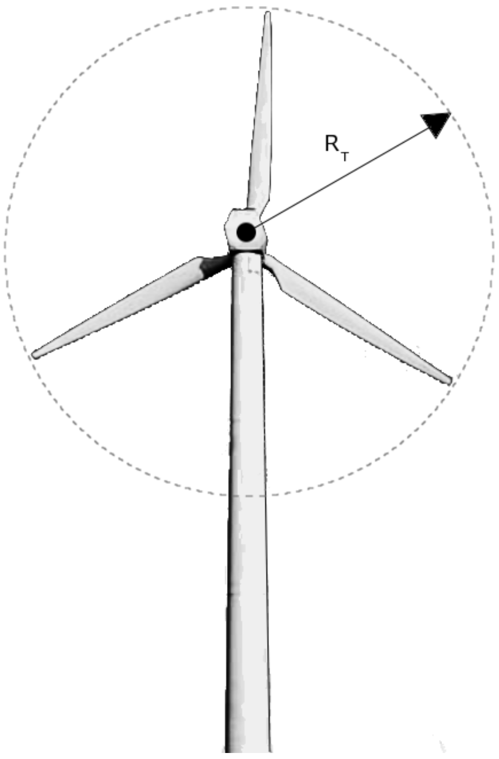


Figure 4.5: Cross-section of typical wind turbine

We need to construct a more realistic representation; this brings in the concept of the local solidity, β, which will capture the radial tapering of the blades. This is defined as a function distance from the axis of rotation, ie.

$$\beta = \beta(r) \tag{4.21}$$

where $r = \sqrt{(y^2 + z^2)}$.

If we assume that the turbine will have N blades each with a chord length $c(r)$. This gives us a local solidity at radius r of

$$\beta(r) = \frac{Nc(r)}{2\pi r} \tag{4.22}$$

As can be seen from the above equation, the local solidity becomes infinite at $r = 0$. In reality, there is a turbine hub from which the blades extend; and this hub has a radius of R_H. The hub is considered to have a constant local solidity (or solidity density if you prefer), called β_H. From this we can write

$$\beta(r) = \begin{cases} \frac{Nc(r)}{2\pi r} & \text{if } R_H < r \leq R_T \\ \beta_H & \text{if } r \leq R_H \end{cases} \tag{4.23}$$

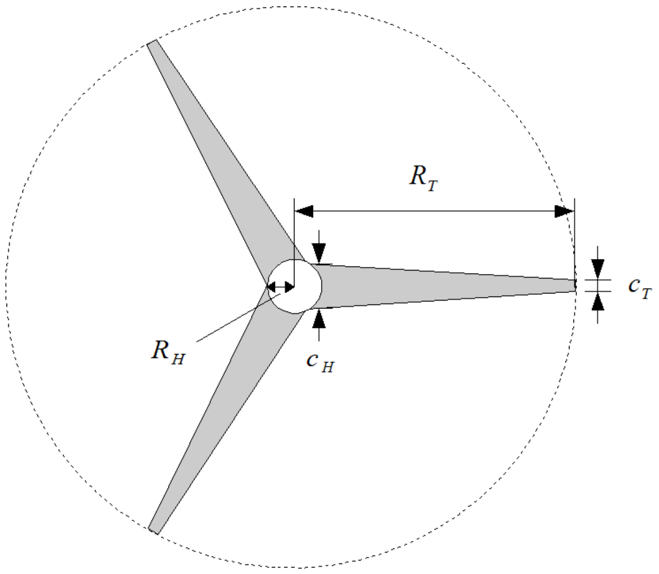


Figure 4.6: Simplified turbine blade model for N=3

For $\beta(r)$ to become useful, we need to find out what $Nc(r)$ is, since there is no direct notion of real blades within the model itself. To find this, we must use an equation for the net solidity:

$$\begin{aligned} B &= \frac{1}{\pi R_T^2}\int_0^{R_T} 2\pi r\beta(r)dr \\ &= \frac{1}{\pi R_T^2}\int_{R_H}^{R_T} 2\pi r\beta(r)dr + \beta_H\left(\frac{\pi R_H^2}{\pi R_T^2}\right) \end{aligned}$$

$$= \frac{1}{\pi R_T^2} \int_{R_H}^{R_T} Nc(r)dr + \beta_H \left(\frac{\pi R_H^2}{\pi R_T^2} \right) \tag{4.24}$$

If we assume that $c(r)$ is a linear function of r, then we can write:

$$B = \frac{1}{\pi R_T^2} \left[N\bar{c}(R_T - R_H) + \beta_H(\pi R_H^2) \right] \tag{4.25}$$

where $N\bar{c}(R_T - R_H)$ represents the cross-sectional area of the blades, and $\beta_H(\pi R_H^2)$ the hub. Since every term aside from $N\bar{c}$ is defined, we can calculate its value as

$$N\bar{c} = (B - B_H) \left(\frac{\pi R_T^2}{R_T - R_H} \right) \tag{4.26}$$

where the hub solidity is $B_H = \beta_H \left(\frac{R_H}{R_T}\right)^2$.

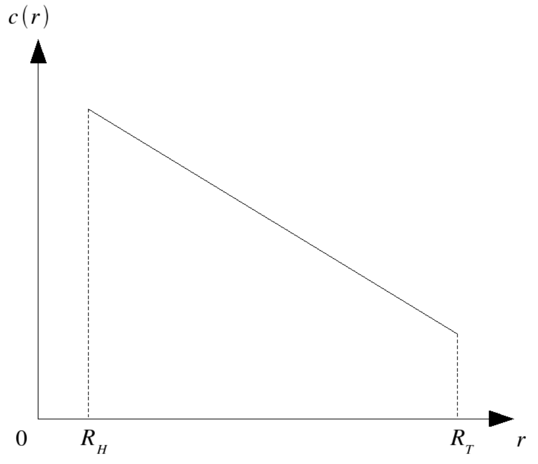


Figure 4.7: $c(r)$ as a linear function of r

We still do not have $Nc(r)$, so if we write Nc_H for $Nc(r = R_H)$ and similarly Nc_T for the tip, we can write

$$N\bar{c} = \frac{1}{2}(Nc_H + Nc_T) \tag{4.27}$$

Using the tip width fraction that states that $c_T = wc_H$, we now have

$$N\bar{c} = \frac{1}{2}Nc_H(1 + w) \tag{4.28}$$

Which is rearranged to give

$$Nc_H = \frac{2N\bar{c}}{1 + w} \tag{4.29}$$

As previously stated, $c(r)$ is assumed to be a linear function of r, so

$$Nc(r) = ar + b \tag{4.30}$$

Where a and b are constants.

These can be found from known values of Nc:

$$Nc(R_H) = aR_H + b = Nc_H \tag{4.31}$$

And if $R_T >> R_H$:

$$Nc(\frac{1}{2}R_T) = N\overline{c} = a\frac{1}{2}R_T + b \tag{4.32}$$

This gives us

$$a = \frac{(N\overline{c} - Nc_H)}{(\frac{R_T}{2} - R_H)} \tag{4.33}$$

and

$$b = Nc_T - aR_H \tag{4.34}$$

Hence $Nc(r)$ is defined, and so $\beta(r)$.

Consequences for equations Now that solidity is no longer spatially uniform, this means the turbine is no longer spatially uniform in its effect on the fluid: rather than deal with B and $\bar{u}$, we must treat these as spatially-varying properties, $\beta(r(\underline{x}))$ and $u(\underline{x})$. ω_{NP} in equation 4.7 must therefore be rewritten.

For a small segment of fluid occupying volume δV at radius r from the axis of the turbine, the moment of inertia is

$$\delta I = \delta M r^2 \tag{4.35}$$

where $\delta M = \rho\delta V$, the mass of the segment.

Revisiting the equations that gave us the definition of ω_{NP} (equation 4.7), we have:

$$\begin{aligned}\frac{1}{2}\delta M u(\underline{x})^2 &= \frac{1}{2}\delta I \omega_{NP}^2(r) \\ &= \frac{1}{2}\delta M r^2 \omega_{NP}^2(r) \end{aligned} \tag{4.36}$$

which reduces to, assuming positive roots only:

$$u(\underline{x}) = r\omega_{NP}(r) \tag{4.37}$$

Adding in the effects of α, $\beta(r)$ and f, we can rearrange this to give

$$\omega_{NP}(r) = f\alpha\beta(r)\left(\frac{u(\underline{x})}{r}\right) \tag{4.38}$$

Which is the resultant angular velocity on our small piece of fluid in the absence of any power extraction. The actual angular velocity, ω_{FL}, is

$$\omega_{FL}(\underline{x}) = (1-Q)\omega_{NP}(\underline{x}) \tag{4.39}$$

This means that our power calculation now becomes

$$PW_{ex} = \frac{1}{4}\rho\pi R_T^4(\overline{\omega_{NP}^2} - \overline{\omega_{FL}^2})\bar{u} \tag{4.40}$$

Where

$$\overline{\omega_{NP}^2} = \frac{1}{V_T}\int_0^{V_T} \omega_{NP}^2(\underline{x})dV$$

and

$$\overline{\omega_{FL}^2} = (1-Q)^2\overline{\omega_{NP}^2}$$

in the limit of $\delta V \to 0$. V_T is the volume of the turbine model cylinder.

But is there any further we can go in 'drilling down' our definition for PW_{ex}? Well, if we rewrite the power equation as

$$PW_{ex} = \frac{1}{4}\rho\pi R_T^4(\overline{\Delta\omega^2})\bar{u} \tag{4.41}$$

where $\overline{\Delta\omega^2} = \overline{\omega_{NP}^2} - \overline{\omega_{FL}^2}$.

Re-expressing this in terms of $\overline{\omega_{FL}^2}$ we have

$$\overline{\Delta\omega^2} = \left(\frac{1}{(1-Q)^2} - 1\right)\overline{\omega_{FL}^2} \tag{4.42}$$

$\overline{\omega_{NP}^2}$ must be solved by integration, and so

$$\begin{aligned}\overline{\omega_{NP}^2} &= \frac{1}{V_T}\int_0^{V_T} \omega_{NP}^2(\underline{x})dV \\ &= \frac{1}{V_T}\int_0^{V_T}\left(f\frac{\alpha}{R_T}\beta(r)u(\underline{x})\right)^2 dV \\ &= \left(\frac{f^2\alpha^2}{V_T}\right)\int_0^{V_T}\beta^2(r)\frac{u^2(\underline{x})}{r^2}dV \end{aligned}\tag{4.43}$$

Here lies the sticking point: analytically, we cannot refine our our definition of $\overline{\omega_{NP}^2}$ any further and so cannot calculate the power extracted, since we do not know $u^2(\underline{x})$. In practice – that is to say, in simulation – we numerically integrate $\omega_{NP}^2(\underline{x})$ and so calculate $\overline{\omega_{NP}^2}$.

From this, we can define $\overline{\omega_{FL}^2}$ and hence $\overline{\Delta\omega^2}$, which then allows us to find PW_{ex}. Precisely how this is done will be detailed in the numerical modelling section.

Effective solidity

There is a problem with our current formulation of power extraction: it can never extract enough power. B for real turbines is of the order of a few percent. In practice, setting B to realistic values gives nothing near the theoretical Betz limit of 59% power extraction nor the rated efficiencies of real turbines of 10-40%. Thus our notion of what solidity actually *is* must be extended, through what it *does*. By saying that solidity is a measure of the effect of the turbine on the flow, rather than just a measure of the cross-sectional area of it, we come to the concept of *effective solidity*.

We shall denote the effective net solidity B', and the local effective solidity β' – these we will substitute for B and β in our power extraction and angular momentum equations.

Net effective solidity As the effective solidity scaling factor must be both non-dimensional and global, it can by extension be applied to the net solidity.

Since the hub and nacelle generate no lift and the blades clearly do, we write the net effective solidity as

$$B'(\omega_T) = [1 + \kappa f_B(\omega_T)]B_{bl} + B_H \tag{4.44}$$

Where B_{bl} is the solidity of the blades, defined as $B_{bl} = B - B_H$, κ is a scaling constant, and $f_B(\omega_T)$ is a function of ω_T.

f_B represents the non-linear interaction between the flow and the blades due to their rotation, which gives rise to effective solidity: it is a parameterisation of such behaviour. Originally it was a simple linear function with $f_B = \omega'$ (where $\omega' = \frac{\omega_T}{\omega_{T,opt}}$), but this was found to produce insufficient power output at $u_0 < u_{0,opt}$ when compared with performance graphs for known wind turbines. Higher effective solidity at lower fluid speeds would allievate this problem, and so a non-linear function was needed.

The criteria for candidate functions were:

1. Symmetry: $f_B(\omega_T) = f_B(-\omega_T)$
2. $f_B(\omega_T = 0) = 0$
3. $f_B(\omega_{T,opt}) = 1$
4. A non-linear bulge at lower ω_T; ie. above the line $f_B = \omega_T$ at $\omega_T < \frac{1}{2}$
5. Near-linear behaviour for $\omega_T > \omega_{T,opt}$

The Vestas V52 turbine was chosen, and a variety of functions evaluated for effectiveness (see fig. 4.8) by comparision with the turbine's known performance figures. Eventually the following equation was decided upon:

$$f_B(\omega_T) = |\omega'|^{1/3} \tag{4.45}$$

With f_B determined, now we can define B' unambiguously. If we take the effective solidity when the turbine blades rotate at $\omega_{T,opt}$, we have

$$B'_{opt} = B'(\omega_{T,opt}) = [1 + \kappa f_B(\omega_{T,opt})]B_{bl} + B_H \tag{4.46}$$

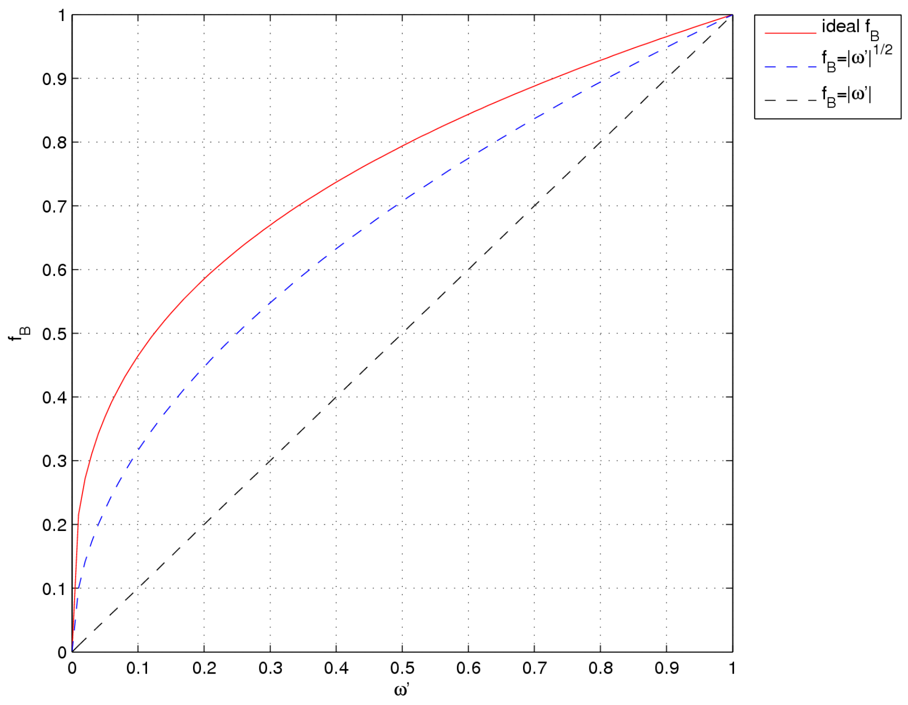


Figure 4.8: A selection of candidates for f_B. The dashed blue line is the linear case, underperforming at $u_0 < u_{0,opt}$, as did the square root function albeit to a lesser extent. The solid red line typifies the ideal function, the cubic root.

Since we know B'_{opt} and $\omega_{T,opt}$ already, this can be rearranged to give

$$\kappa = \frac{1}{f_B(\omega_{T,opt})}\left(\frac{B'_{opt} - B_H}{B_{bl}} - 1\right) \tag{4.47}$$

Thus κ can be determined.

Local effective solidity In the blade section, the effective solidity in a small volume segment varies as the local orbital speed of the blades through the fluid, instead of simply just the turbine angular velocity ω_T. Thus

$$\beta'(r) = \begin{cases} [1 + krf_B(\omega_T)]\beta(r) & \text{if } R_H < r \leq R_T \\ \beta_H & \text{if } r \leq R_H \end{cases} \tag{4.48}$$

where k is a scaling constant of units $\frac{1}{distance}$. This can be determined from the net effective solidity. Recalling equation 4.24, we can write

$$\begin{aligned}
B' &= \frac{1}{\pi R_T^2}\int_0^{R_T} 2\pi r\beta'(r)dr \\
&= \frac{1}{\pi R_T^2}\int_{R_H}^{R_T} [1 + krf_B(\omega_T)]2\pi r\beta(r)dr + B_H \\
&= B + \frac{1}{\pi R_T^2}\int_{R_H}^{R_T} kf_B(\omega_T)2\pi r^2\beta(r)dr \\
&= B + \frac{kf_B(\omega_T)}{\pi R_T^2}\int_{R_H}^{R_T} rNc(r)dr \qquad (4.49)
\end{aligned}$$

As k is constant, we can determine this through optimum values, which are known already. Thus

$$B'_{opt} = B + \frac{kf_B(\omega_{T,opt})}{\pi R_T^2}\int_{R_H}^{R_T} rNc(r)dr \qquad (4.50)$$

Expanding out $Nc(r)$ from equation 4.30, we have

$$\begin{aligned}
B'_{opt} &= B + \frac{kf_B(\omega_{T,opt})}{\pi R_T^2}\int_{R_H}^{R_T} r(ar+b)dr \\
&= B + \frac{kf_B(\omega_{T,opt})}{\pi R_T^2}\int_{R_H}^{R_T} (ar^2+br)dr \\
&= B + \frac{kf_B(\omega_{T,opt})}{\pi R_T^2}\left[\frac{1}{3}ar^3 + \frac{1}{2}br^2\right]_{R_H}^{R_T} \\
&= B + \frac{kf_B(\omega_{T,opt})}{\pi R_T^2}\left[\frac{1}{3}a(R_T^3 - R_H^3) - \frac{1}{2}b(R_T^2 - R_H^2)\right] \\
&= B + kS \qquad (4.51)
\end{aligned}$$

Where S is a constant which we can find, since a and b have already been determined from equations 4.33 and 4.34 respectively.

This gives us

$$k = \frac{(B'_{opt} - B)}{S} \qquad (4.52)$$

This is the scaling constant for the local effective solidity.

4.3.5. Limiting turbine performance

With increasing flow strength, the angular velocity of a turbine, ω_T will naturally increase also. However, at higher speeds real turbines are subjected to forces which cause excessive wear and tear; as a result, most try to limit ω_T by altering the blade pitch or using designed-in blade stall.

The blade parameter α can be thought of as mimicking the effect of pitch or stall on the turbine; essentially, it limits the effect the flow can have on the turbine.

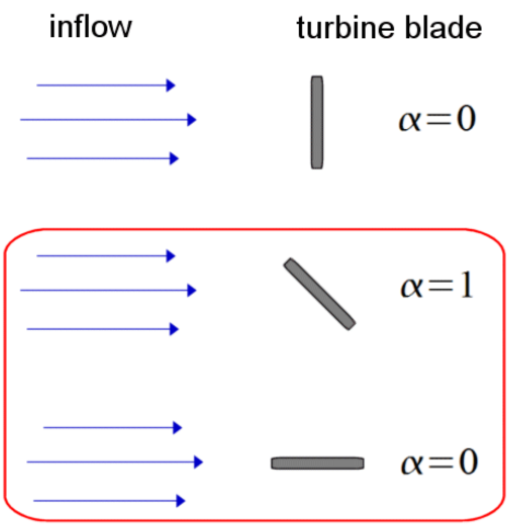


Figure 4.9: Possible physical interpretations of the blade factor. We are interested in those encircled.

We say that at $\alpha = 1$ the turbine is at its most efficient, and at $\alpha = 0$ it is completely inefficient. Thus

$$0 \leq \alpha \leq 1 \tag{4.53}$$

Calculating α for 'hard' limiting turbines

Many modern horizontal axial wind turbines actively monitor their performance, and adjust blade pitch accordingly to maintain a constant power output at higher freestream velocities. Axial flow marine turbines also do this, rather than have cavitation bubbles forming on the blades during stall, which can lead to excessive wear on blades' surfaces [30]. Since the power output does not increase beyond $u_{0,opt}$, the angular velocity of the turbine, ω_T does not increase beyond $\omega_{T,opt}$ and so

$$\omega_{T,max} = \omega_{T,opt} \tag{4.54}$$

This is not the case for soft-limited turbines, which will be returned to later.

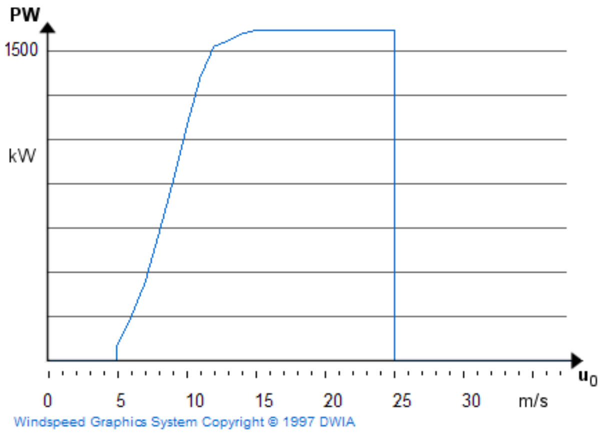


Figure 4.10: Wind speed versus hard-limited power outpout of a NEG Micon 1500 turbine (Courtesy of Danish Wind Industry Association website)

Also, we state the following conditions must hold for α:

$$\alpha(\omega_T \leq \omega_{T,max}) = 1 \tag{4.55}$$

$$\alpha(\omega_T > \omega_{T,max}) < 1 \tag{4.56}$$

That is to say above $\omega_{T,max}$, α must decrease to keep ω_T close to its maximum value. We do this via a correction to α, $\Delta\alpha$

$$\alpha' = \alpha + \Delta\alpha \tag{4.57}$$

In numerical modelling terms, α' would represent the blade factor at the next time step, and α at the last: more on this in section 4.4.

An oscillatory problem We can view the pitch control mechanism of a turbine and the interconnected behaviour of the turbine as a damped harmonic oscillator.

This interpretation of the control system first came about when a former, rudimentary mechanism was introduced (one that just decreased α at a fixed rate when $\omega_T > \omega_{T,max}$): depending on relaxation parameter for α, oscillations of both α and ω_T could be produced. In fact, it was very difficult to get them *not* to oscillate. Clearly, not only did α affect ω_T and vice versa; the rates of change of both were also interlinked.

The starting point in such an approach would be the damped spring equation, ie.

$$m\ddot{X} + c\dot{X} + KX = 0 \tag{4.58}$$

where X is the displacement from equilibrium, and K, c, and m are constants.

In our case $X = \Delta\omega_T = (\omega_T - \omega_{T,max})$, thus

$$m(\ddot{\Delta\omega_T}) + c(\dot{\Delta\omega_T}) + K(\Delta\omega_T) = 0 \tag{4.59}$$

Which can be rewritten as

$$m\ddot{\omega}_T + c\dot{\omega}_T + K\omega_T - K\omega_{T,max} = 0 \tag{4.60}$$

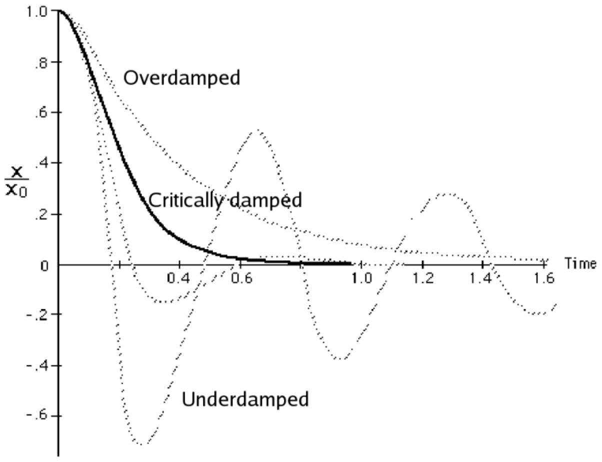


Figure 4.11: Motion of a damped oscillator over time

Thus the system is capable of three types of oscillation: overdamped, underdamped, and critically damped. K, c, and m are defined by the physical and behaviour characteristics of the turbine, such as physical inertia, lift, blade response to ω_T; they are also affected – indirectly via fluid dynamics calculations – by properties of the fluid, such as u_0, μ and ρ.

Because of the last set of dependencies, K, c and m are impossible to calculate precisely. What we can be sure of is that if we model the turbine blade response (via α) as a spring system, for different flow strengths the system will exhibit all three types of damped oscillation previously mentioned. As a result, care must be taken in modelling the physical response of the system to the fluid strength, if it is to demonstrate stable behaviour in a variety of conditions.

Towards a solution When the system was chosen to behave as a spring – with $\ddot{\omega}_T$ a function of equilibrium displacement – numerical simulations showed oscillation. Furthermore, the critical-damped equation constants were highly sensitive to flow speed: a new model was needed.

Typically, the simulated turbine would 'overshoot', so that $\dot{\omega}_T$ would still be large at $\Delta\omega_T = 0$. Thus, the first stated condition to avoid this is

$$\dot{\omega}_T(\omega_T = \omega_{T,max}) = 0 \tag{4.61}$$

This also implies that $\dot{\omega}_T$ is a function of ω_T. For simplicity, we make this linear, with maximum and minimum values of $(\dot{\omega}_T)_{max}$ and $-(\dot{\omega}_T)_{max}$ respectively.

But what does this mean? It can be best envisaged as a driver accelerating a car towards a certain speed. If our driver aims to travel at 50mph, say, he or she simply does not accelerate hard towards that limit and instantly lift their foot off the accelerator pedal at the right moment (and by the right amount) when they reach 50mph; rather, they ease off the pedal slowly, decreasing the car's acceleration until they achieve the correct speed. It is damping, of a sort – but active, rather than passive.

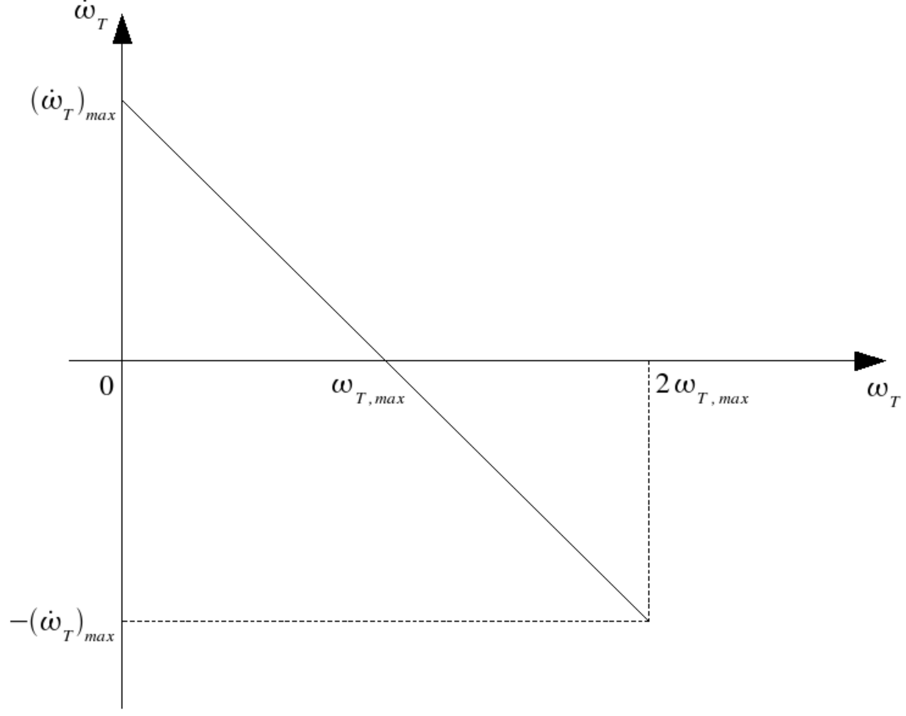


Figure 4.12: ω_T versus $\dot{\omega}_T$

Defining the bounds to our linear equation, we have

$$(\dot{\omega}_T)_{max} = \frac{d\omega_T}{dt} \approx \frac{\Delta\omega_T^*}{\Delta t^*} \tag{4.62}$$

If we then define Δt^* as Δt_α, the time taken for the blades from optimum to parallel-flow position (see relaxation subsection in numerical modelling), and $\Delta\omega_T^*$ as $\omega_{T,max}$, the maximum allowable angular velocity of the turbine, then we have

$$(\dot{\omega}_T)_{max} = \frac{\omega_{T,max}}{\Delta t_\alpha} \tag{4.63}$$

In the model, $\dot{\omega}_T$ continuously fluctuates as a result of unsteady flow; it needs to be adjusted to resemble figure 4.12 – the *target* values. The above equation allows us to express the target value for $\dot{\omega}_T$ as

$$\begin{aligned}(\dot{\omega}_T)_{targ} &= (\dot{\omega}_T)_{max} - \left[\frac{(\dot{\omega}_T)_{max}}{\omega_{T,max}}\right]\omega_T \\ &= \frac{\omega_{T,max} - \omega_T}{\Delta t_\alpha}\end{aligned} \tag{4.64}$$

We then calculate the normalised difference between the actual value of $\dot{\omega}_T$, and the target value

$$n = \left[\frac{(\dot{\omega}_T)_{targ} - \dot{\omega}_T}{(\dot{\omega}_T)_{max}} \right] \tag{4.65}$$

α_{targ} would be difficult to calculate, given that the link between it and $\dot{\omega}_T$ depends upon CFD calculations. What we can do however, is adjust α via $\Delta\alpha$ until $\omega_T \approx \omega_{T,max}$ (the condition $0 < \alpha < 1$ notwithstanding).

If we define the maximum rate of change of α as

$$\dot{\alpha}_{max} = \frac{1}{\Delta t_\alpha} \tag{4.66}$$

We can then write the corrective term to α as

$$\Delta\alpha = \Delta t \left(\frac{\dot{\alpha} + n\dot{\alpha}_{max}}{2} \right) \tag{4.67}$$

Where $\dot{\alpha}$ is the current rate of change of α, and Δt is the time over which the change to α is applied.

Calculating α for soft limiting turbines

Older wind turbines often relied solely on stall to limit their performance at higher wind speeds, such as the Kuriant 18. For completeness, the turbine model was extended to accomodate this; however the behaviour can easily be turned off to revert to hard limiting.

A simplified specification of the problem can be seen in figure 4.14. This assumes the excess power beyond $u_{0,opt}$ can be approximated by a straight line.

Below $u_{0,opt}$, ω_T never exceeds $\omega_{T,opt}$ and so $\alpha = 1$, $\omega_{T,max} = \omega_{T,opt}$ always; therefore the behaviour at the lower speeds is unmodified. We are concerned with the behaviour of the turbine - specifically the calculation of α - in the region of $u_{0,opt} < u_0 < u_{0,cutout}$. In this area, $\omega_{T,max} > \omega_{T,opt}$.

Since we already know PW_{opt} from $c_{p,opt}$, and to calculate $PW_{opt,cutout}$ we must also specify $c_{p,cutout}$ and $u_{0,cutout}$ too. These can be readily ascertained from performance graphs for axial-flow wind and marine turbines, such as are

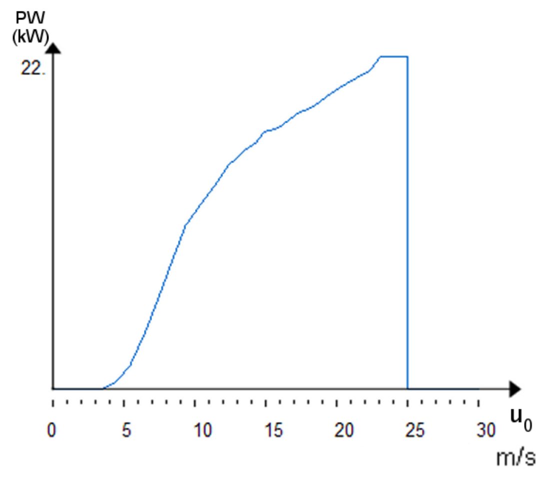


Figure 4.13: Wind speed versus soft-limited power output of a Kuriant 18 turbine (Courtesy of Danish Wind Industry Association website)

available on www.wind-turbine.org (last accessed April 2008). If we assume a linear relationship between u_0 and u_{max} where $u_0 > u_{0,opt}$, and that $u_0 = 0$ when $u_{max} = 0$, then we can express one as an approximation of the other:

$$u_0 = \left(\frac{u_{0,cutout}}{u_{cutout}}\right) u_{max} \tag{4.68}$$

Remembering the linear increase in power with u_0 beyond $u_{0,opt}$, we can make a stab at the maximum currently allowable power for the turbine:

$$PW_{max} = PW_{opt} + \left(\frac{\Delta PW}{\Delta u_0}\right)(u_0 - u_{0,opt}) \tag{4.69}$$

where $\Delta PW = (PW_{cutout} - PW_{opt})$ and $\Delta u_0 = (u_{0,cutout} - u_{0,opt})$.

We can then calculate $\omega_{T,max}$, the maximum operating angular velocity as

$$\omega_{T,max}(u_{0,opt} < u_0 < u_{0,cutout}) = \sqrt{\frac{PW_{max}}{K_\tau}} \tag{4.70}$$

K_τ being the constant previously defined in equation 4.18.

We can now calculate $\Delta\alpha$ as per section 4.3.5, with our redefined $\omega_{T,max}$.

Cut-in and cut-out speeds

In both wind and marine turbines, there are both minimum flow speeds below which the turbine will no longer function, and maximum speeds beyond which

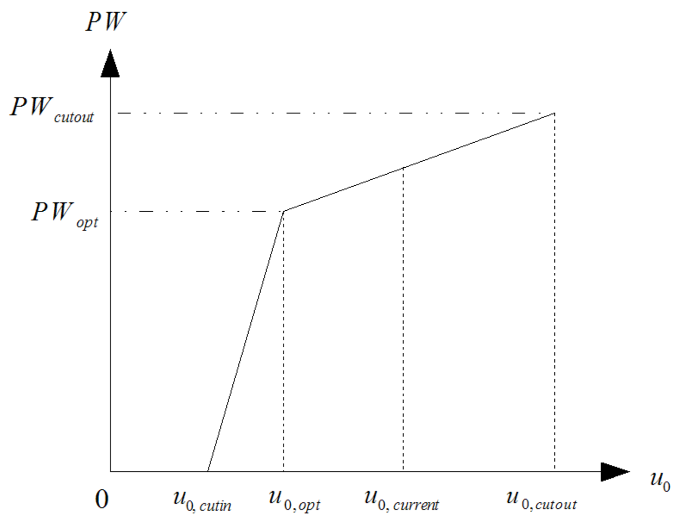


Figure 4.14: Idealised graph of power output for soft-limited performance

the turbine will shut down to avoid damage to the turbine. We call these *cut-in* and *cut-out* speeds respectively; we shall label them u_{cutin} and u_{cutout}

Real turbines have flow speed indicators attached to their structures, but we will instead take the maximum flow speed within the turbine, ie.

$$u_{max} = \max\left(|\underline{u}(\underline{x})|\right) \tag{4.71}$$

where $\underline{x} \in V_T$

This gives us a value to compare to u_{cutin} and u_{cutout}. Therefore, we can define the conditions under which the turbine will not function:

$$u_{max} < u_{cutin} \tag{4.72}$$

$$u_{max} > u_{cutout} \tag{4.73}$$

If we set $\alpha = 0$ under the above conditions, then the turbine will effectively stop working, and the values for ω_{NP}, ω_{FL}, ω_T and PW_{ex} will become zero. It should be noted that due to body forces applied to the flow by the turbine, u_{max} will be lower than the corresponding u_0. Thus, u_{cutin} and u_{cutout} have to be determined experimentally to provide the desired behaviour at specific values of u_0.

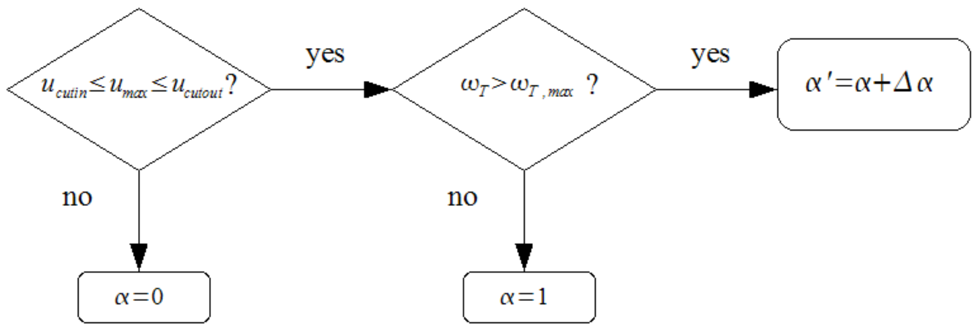

$u_{cutin} \le u_{max} \le u_{cutout}$?
yes
$\omega_T > \omega_{T,max}$?
yes
$\alpha' = \alpha + \Delta\alpha$
no
no
$\alpha = 0$
$\alpha = 1$

Figure 4.15: Flowchart for calculating α

Combining pitch, stall and cut-in/out

Putting all this together, we can devise a simple flowchart for calculating α . It is conditionally a two-stage process, firstly comparing u_{max} then ω_T, and is detailed in figure 4.15.

4.3.6. Turbulence

Turbulence plays an important part in the performance of turbines, as was first clearly stated by Lissaman [47]. The vortices generated at the blade tips, and the flow separation that occurs at high angles of attack, both generate turbulence which allows greater mixing of the surrounding fluid with the near-stagnant flow behind the turbine, through a process known as *momentum diffusion*. This effectively transfers kinetic energy to the more central parts of the wake. In earlier revisions of the turbine model which did not model turbulence, significant recirculation would occur at $u_{0,opt}$ in the inner parts of the wake several diameters downstream, thereby reducing the efficiency of the model turbine as predicted by momentum theory. As this clearly effects the wake and the power output, the effect of the vorticies at least must be modelled, if not the vortices themselves.

The turbulent rings

Consider a thin disc at the tail-end of the turbine, of L_{tip} width, which consists of an outer ring of $(R_T - R_{tip})$ thickness, and an inner ring of radius R_{tip}. The outer ring is where the vortex turbulence is generated, and the inner ring where lower-level turbulence from the rotor blades is created. Within this ring at the axis is the hub section.

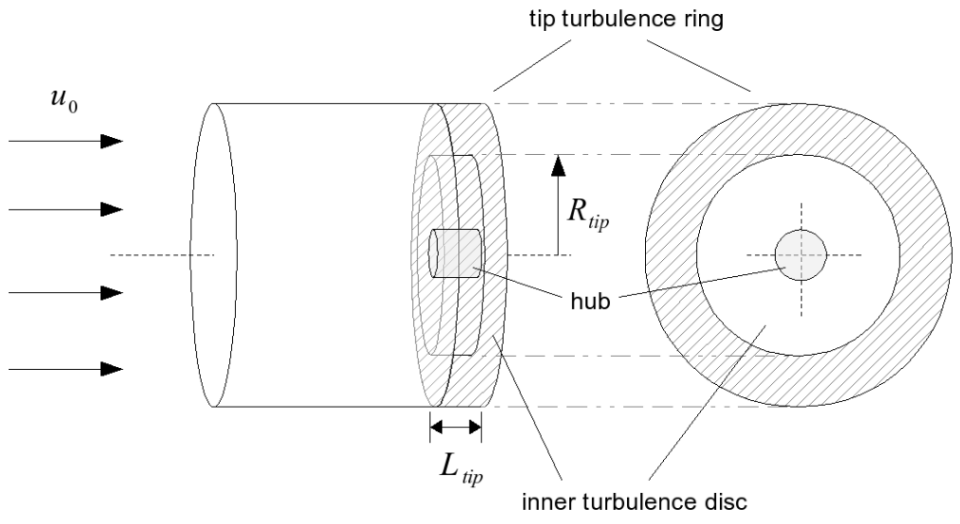


Figure 4.16: Pseudo-3D side and plan view showing the turbulent trailing vortex rings of the model

We then choose our optimum turbulence intensity at the blade tips, $Ti_{x,opt}$. Turbulence intensity is the standard deviation of fluid speed or velocity over time, normalised by u_0 so it lies between 0 and 1. $Ti_{x,opt}$ defines this in the x direction, when the blades are rotating at $\omega_{T,opt}$, and from wake measurements this can be defined as approximately within the range

$$0.12 \leq Ti_{x,opt} \leq 0.20 \tag{4.74}$$

With the upper end of the scale representing larger wind turbines (ie. $>$ 20 m diameter).

Three simplifying assumptions are made at this point:

1. The tip turbulence intensity Ti_x at any given moment varies linearly with the angular velocity of the turbine blades

2. No turbulence is generated by the turbine when the blades are not rotating, ie. when $\omega_T = 0$

3. The hub section does not generate any turbulence.

This leads to the equation calculating the time-dependent tip turbulence intensity in the x-direction:

$$Ti(t)_x = Ti_{x,opt}\left[\frac{\omega_T(t)}{\omega_{T,opt}}\right] \tag{4.75}$$

We then turn to the radial component of the turbulence intensity. Orbital turbulence is ignored, since this will be partially induced by the rotation of the wake by the turbine algorithm, and also since we are primarily looking at the tip-induced turbulence which encourages the outer layers of the wake to mix with the stagnant inner core. From Vermeer et al, pp 494 [73] we can make a rough approximation to radial turbulence

$$Ti(t)_{rad} \approx \frac{1}{2}\,Ti_x \tag{4.76}$$

Now we look at the turbulence generated by the length of the blades, in the inner disc section. The turbulence profiles in Hassain et al, pp2260 [37], show turbulence dropping to roughly half that near the tips. Thus if we write

$$\underline{Ti} = \begin{bmatrix} Ti_x \\ Ti_{rad} \end{bmatrix} \tag{4.77}$$

Then

$$\underline{Ti}_{inner} \approx \frac{1}{2}\,\underline{Ti} \tag{4.78}$$

Lastly, within the hub volume we define the turbulence as

$$\underline{Ti}_H = 0 \tag{4.79}$$

These turbulence components are then applied to the flow in a way that shall be detailed in the numerical modelling section below.

4.4. Numerical modelling

In this part, we take the equations of the previous section, and develop them for discrete numerical simulation. The model itself was developed to operate with Fluidity, a CFD solver from Imperial College, London [63]. This uses finite element analysis with unstructured adaptive meshes, however the model itself is generic enough to be applied to finite difference or finite volume techniques.

4.4.1. Translation, rotation and volume definition

First, we translate and rotate the co-ordinate system and velocity field so that centroid of the marine turbine is at $(0,0,0)$ and the blade 'surface' is aligned to the y-axis:

$$\underline{x}' = R_{-\phi} \begin{bmatrix} x - x_T \\ y - y_T \\ z - z_T \end{bmatrix} \tag{4.80}$$

where $R_{-\phi} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$

And

$$\underline{u}'(\underline{x}') = R_{-\phi}\underline{u}(\underline{x}) \tag{4.81}$$

Dropping the $'$ for brevity, we then define the volume that the turbine occupies, V_T, as

$-\frac{L}{2} \leq x \leq \frac{L}{2}$ and $\sqrt{(y^2+z^2)} \leq R_T$

4.4.2. Velocity changes inside the turbine

About nodes and elements

In a typical simulation run, at each time-step the turbine algorithm calculates the body forces acting on the fluid.

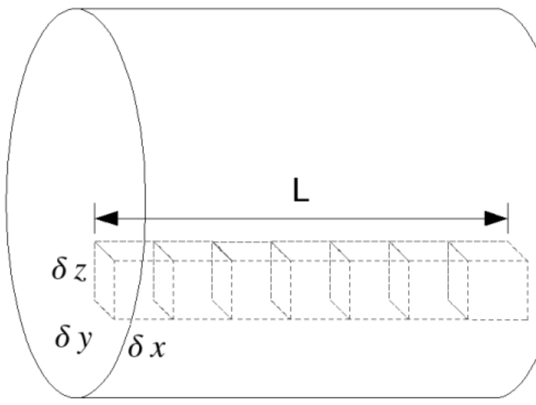


Figure 4.17: A row of cubic elements in the turbine volume

At any point in time t, there are N nodes within the volume V_T inhabited by the turbine. N is a function of t, as are the nodes contained within V_T. It is at these nodes we specify the body forces. These nodes form the corners of the elements, but we do not need to know the specifics of their geometry, since the model is node-centric rather than element-centric.

The properties of any turbine node i are denoted as $\underline{x}_i$, $\underline{u}_i$, etc. – so x_i is valid $\forall i$, where $1 \leq i \leq N$.

Angular velocity and local solidity in the blade volume

To recall, the angular velocity of the fluid in a small volume δV leaving the free-wheeling turbine at radial distance r is calculated, with effective solidity included, as:

$$\omega_{NP}(r) = f\frac{\alpha}{r}\beta'(r)u(\underline{x}) \tag{4.82}$$

We re-write this in discrete form as:

$$\omega_{NP,i} = f\frac{\alpha}{r_i}\beta'(r_i)u_i \tag{4.83}$$

where

$$r_i = \sqrt{y_i^2 + z_i^2} \tag{4.84}$$

This gives us enough to calculate the body forces on each element.

In the blade volume

From a numerical point of view, the body force on the fluid is specified in terms of *force density,* ie. the force per unit volume. This means that we have to first consider the force acting over the whole of the turbine first.

When in free-wheeling (no power extraction) mode, the model will increase the angular velocity of the fluid passing through the turbine by $\omega_{NP}(r)$. The angular acceleration of the fluid is uniform throughout the volume for the same radial distance, so that nodes near the turbine intake can be treated similarly to those near the outlet provided they share the same value of r.

Problem: initial hypothesis Supposing we have a mesh node, i, which has physical co-ordinates $\overline{x}_i$, satisfying the condition $R_H \leq r_i < R_T$. We *could* write the change in the y and z components of velocity for that node i due to orbital forces as

$$\Delta v_{NP,i} = -\omega_{NP,i} z_i \tag{4.85}$$

$$\Delta w_{NP,i} = \omega_{NP,i} y_i \tag{4.86}$$

These equations give us the backthrust component of the velocity:

$$\Delta u_i = -\sqrt{(\Delta v_{NP,i}^2 + \Delta w_{NP,i}^2)} \tag{4.87}$$

We now calculate the actual orbital velocity changes actually applied to the fluid. These are:

$$\Delta v_i = -\omega_{FL,i} z_i \tag{4.88}$$

$$\Delta w_i = \omega_{FL,i} y_i \tag{4.89}$$

There is a problem with above equations, however. How do we calculate $\omega_{NP,i}$ and $\omega_{FL,i}$?

Solution: the normalised solidity function $\omega_{NP,i}$ and $\omega_{FL,i}$ are spatially-varying, and cannot universally be constructed in a relaxed form (see section 4.4.6). While a fixed-mesh, finite-difference scheme would have little trouble, due to the nature of hr-adaptive FEM, it is awkward to relax either without computationally expensive interpolation, as the point $\underline{x}_i$ may not exist in the next time step. To get around this, we introduce what programmers call a 'hack'. Firstly, we calculate the spatial mean angular velocity of fluid in the free wheeling turbine:

$$\overline{\omega}_{NP} = \frac{1}{N}\sum_{i=1}^{N}\omega_{NP,i} \tag{4.90}$$

If we then introduce what we might call a *normalised solidity function*, for which the following must be true $\forall i$:

$$\omega_{NP,i} = \overline{\omega_{NP}}\beta^*(r_i) \tag{4.91}$$

$\beta^*(r_i)$ is essentially the local (effective) solidity scaled so that the total effective solidity is 1, therefore

$$\beta^*(r_i) = \frac{\beta'(r_i)}{B'} \tag{4.92}$$

This allows us to rewrite the velocity changes as

$$\Delta v_{NP,i} = -\overline{\omega_{NP}}\beta^*(r_i)z_i \tag{4.93}$$

$$\Delta w_{NP,i} = \overline{\omega_{NP}}\beta^*(r_i)y_i \tag{4.94}$$

And so we can calculate Δu_i. If we now similarly define $\omega_{FL,i}$ as

$$\omega_{FL,i} = \overline{\omega_{FL}}\beta^*(r_i) \tag{4.95}$$

Where

$$\omega_{FL} = \frac{1}{N}\sum_{i=1}^{N}\omega_{FL,i}$$

This gives us the orbital components of the velocity changes:

$$\Delta v_{FL,i} = -\overline{\omega_{FL}}\beta^*(r_i)z_i \tag{4.96}$$

$$\Delta w_{FL,i} = \overline{\omega_{FL}}\beta^*(r_i)y_i \tag{4.97}$$

This makes it much easier to relax the acceleration equations, since $\overline{\omega_{NP}}$ and, by proxy, $\overline{\omega_{FL}}$ are both global properties of the turbine. Also, by taking the spatial average of ω_{NP} this has the additional benefit of ironing out any spatial 'hiccups' in the velocity fields which could further serve to destabilise the algorithm.

In the hub volume

The volume that contained the hub is treated as a special case. Unlike the blade volume around it, it does not generate circulation. It should not necessarily extract energy from the flow either; it can either allow the flow to pass freely, or act as a simple momentum sink.

We shall consider a set of M points contained in the hub volume, ie. where $r_i < R_H$ for $i = \{1, 2, ...N\}$, for calculating body forces generated by the hub.

The hub itself occupies an insignificant portion of the main model volume. For $R_H \approx \frac{1}{10}R_T$, the number of nodes in the hub volume will be $M \approx \frac{1}{100}N$, where N is number of nodes in the total volume. As $M << N$ and the contribution to $\bar{u}$ of the hub is so small, it is safe to use global values such as $\overline{\Delta\omega^2}$ and $\bar{u}$ when dealing with the blade volume.

We make the assertion that if the solidity density of the hub is unity ($\beta_H = 1$), then the fluid will be decelerated to near stagnation at the outflow. This affects the proportion of the momentum that is removed. For each node i, the changes in the velocity components are

$$\Delta u_{FL,i} = -\beta_H\overline{u}_H \tag{4.98}$$

$$\Delta v_{FL,i} = 0 \tag{4.99}$$

$$\Delta w_{FL,i} = 0 \tag{4.100}$$

4.4.3. Modelling turbulence

Revisiting the turbulence ideas discussed in section 4.3.6, we now ascribe some dimensions. L_{tip} is defined so that it is just over one element thick: this ensures that turbulence forces will be applied in as thin a volume as possible. Therefore

$$L_{tip} = 2(\overline{\Delta x}) \tag{4.101}$$

Now to define the points contained in the tip ring, and the inner ring – no turbulence is generated in the hub volume. We only apply our turbulence to a node $\underline{x}_i$ within the turbine when

$$(\frac{L}{2} - L_{tip}) < x_i < \frac{L}{2} \tag{4.102}$$

The radial constraints are

1. $R_{tip} \leq r_i < R_T$: tip turbulence ring
2. $R_H < r_i < R_{tip}$: inner turbulence ring

where $r_i = \sqrt{y_i^2 + z_i^2}$

We now have to calculate the changes in velocity for each point that satisfies these conditions. In a concession to the LES turbulence scheme Fluidity employs [45] [54], it was decided that by simply applying turbulence velocity changes to the fluid, turbulence effects such as turbulence viscosity would be triggered in a satisfactory manner. It has two components: radial, and axial.

The radial component of the velocity change is $\Delta u_{rad,Ti}$ and x-axis component Δu_{Ti}. These are defined for point i as

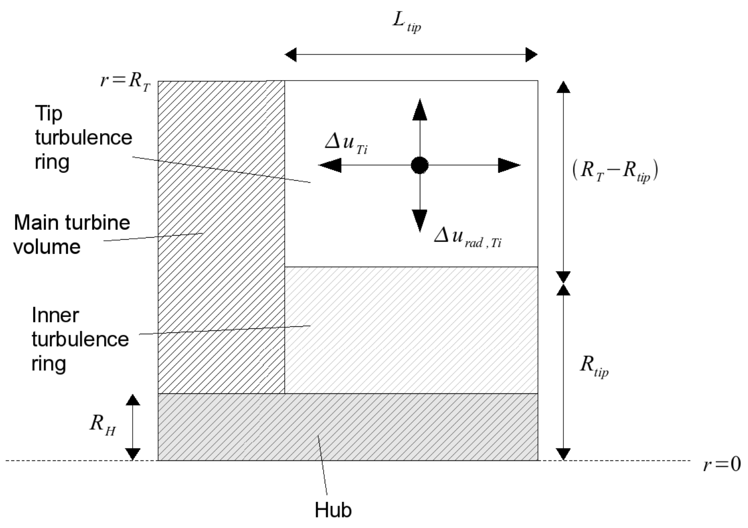


Figure 4.18: View of radial and x-axis components of turbulent velocity changes

$$\Delta u_{rad,Ti} = \begin{cases} Ti_{rad,inner} \; . \; u_0 G(t) & \text{if } r_i < R_H \\ Ti_{rad} \; . \; u_0 G(t) & \text{if } R_H \leq r_i < R_T \end{cases} \tag{4.103}$$

$$\Delta u_{Ti} = \begin{cases} Ti_{x,inner} \; . \; u_0 G(t) & \text{if } r_i < R_H \\ Ti_{x} \; . \; u_0 G(t) & \text{if } R_H \leq r_i < R_T \end{cases} \tag{4.104}$$

Where $G(t)$ is a Gaussian random number generator – Gaussian, because the characteristics of turbulence spectra for turbine generated turbulence are not known, and this distribution is the simplest, reasonable assumption; Gaussian distributions are found in many natural phenomena.

Since we do not know u_0, we can approximate it in the following manner, which was found through extensive prelimary testing to be reasonably accurate:

$$u_0 \approx 1.1 \; u_{max} \tag{4.105}$$

We then calculate the y and z components

$$\Delta v_{Ti} = \left(\frac{y_i}{r_i}\right) \Delta u_{rad,Ti} \tag{4.106}$$

$$\Delta w_{Ti} = \left(\frac{z_i}{r_i}\right) \Delta u_{rad,Ti} \tag{4.107}$$

Next we have to establish a time-scale over which to apply the acceleration. Since we do this roughly over the length of one element, we can write this as

$$\Delta t_{Ti} = \frac{L_{tip}}{\bar{u}} \tag{4.108}$$

4.4.4. Per-node force terms

Now we need to express the changes in velocity as body forces acting on the fluid. Before these can be calculated, we need to know the time over which the non-turbulent forces act – this is the time the fluid takes to pass through the turbine model's cylindrical volume. Recalling the mean speed of the fluid through the turbine as

$$\bar{u} = \frac{1}{N} \sum_{i=1}^{N} u_i \tag{4.109}$$

and the mean speed of the fluid through the hub as

$$\bar{u}_H = \frac{1}{M} \sum_{i=1}^{M} u_i \tag{4.110}$$

We can then define the time for fluid to travel through the turbine as

$$\Delta t_T = \begin{cases} \frac{L}{\bar{u}_H} & \text{if } r_i < R_H \\ \frac{L}{\bar{u}} & \text{if } R <_H \leq r_i < R_T \end{cases} \tag{4.111}$$

We still have to get the net force applied over an element, $\underline{F}_i$. Including the turbulence terms, we have acceleration vector components

$$\dot{u}_i = \left(\frac{\Delta u_i}{\Delta t_T}\right) + \left(\frac{\Delta u_{Ti}}{\Delta t_{Ti}}\right) \tag{4.112}$$

$$\dot{v}_i = \left(\frac{\Delta v_i}{\Delta t_T}\right) + \left(\frac{\Delta v_{Ti}}{\Delta t_{Ti}}\right) \tag{4.113}$$

$$\dot{w}_i = \left(\frac{\Delta w_i}{\Delta t_T}\right) + \left(\frac{\Delta w_{Ti}}{\Delta t_{Ti}}\right) \tag{4.114}$$

The force per unit volume for element i will then be

$$\underline{F}_i = \rho \begin{bmatrix} \dot{u}_i \\ \dot{v}_i \\ \dot{w}_i \end{bmatrix} \tag{4.115}$$

ie.

$$\underline{F}_i = \rho \underline{\dot{u}}_i \tag{4.116}$$

4.4.5. Power

Recalling the analytic form of the power equation (3.9)

$$PW_{ex} = \frac{1}{4}\rho\pi R_T^4(\omega_{NP}^2 - \omega_{FL}^2)\bar{u} \tag{4.117}$$

It is clear that, since ω_{NP}^2 and ω_{FL}^2 now vary with r due to local solidity, that we have to use mean values instead, ie.

$$PW_{ex} = \frac{1}{4}\rho\pi R_T^4(\overline{\omega_{NP}^2} - \overline{\omega_{FL}^2})\bar{u} \tag{4.118}$$

We write this as

$$PW_{ex} = \frac{1}{4}\rho\pi R_T^4(\overline{\Delta\omega^2})\bar{u} \tag{4.119}$$

$\overline{\Delta\omega^2}$ can be expressed in numerical terms as

$$\overline{\Delta\omega^2} = \frac{1}{N}\sum_{i=1}^{N}\left(\omega_{NP,i}^2 - \omega_{FL,i}^2\right) \tag{4.120}$$

Bearing in mind that $\omega_{FL,i} = (1-Q)\omega_{NP,i}$, this becomes

$$\overline{\Delta\omega^2} = \frac{1}{N}\left[1-(1-Q)^2\right]\sum_{i=1}^{N}\omega_{NP,i}^2 \tag{4.121}$$

When the mean x-component of the velocity, $\overline{u}$ has been calculated

$$\overline{u} = \frac{1}{N}\sum_{i=1}^{N}u_i \tag{4.122}$$

Then the power extracted by the turbine can be derived, giving

$$PW_{ex} = \frac{1}{4} N \rho \pi R_T^4 \left[\left(1 - (1-Q)^2\right) \sum_{i=1}^{N} \omega_{NP,i}^2 \right] \sum_{i=1}^{N} u_i \tag{4.123}$$

4.4.6. Relaxation of variables

Relaxation is a technique by which the calculation of variables at each iteration may be stabilised. Without relaxation, these variables may fluctuate in an erratic manner, particularly when there is an interdependence between them and the flow field – as is the case here. Only global calculated properties of the turbine are relaxed: elemental node values could not be, since adaptivity meant that at the next iteration no guarantees could be that a particular node would still exist. Further to that, new nodes could be created, which would not have a previous value with which to relax to.

If we have a set P of global properties of the turbine, then p_i is the i^{th} member of that set; at time-step n, this is written $p_i(n)$. To calculate p_i for the next time-step, $n+1$, we first calculate the tentative, non-relaxed new value of this property, p_i^*. Then we put this into a relaxed form, and so calculate p_i for the next time-step

$$p_i(n+1) = (1-\gamma_i)p_i^* + \gamma_i p_i(n) \tag{4.124}$$

where γ_i is the relaxation parameter for that global turbine property p_i, and

$$0 < \gamma_i < 1 \tag{4.125}$$

for all i.

List of relaxed variables

Only a select few properties were relaxed, since many are derived from others and thus not directly responsible for stability. The relaxation parameter γ_{vel} relaxes variables directly related to fluid velocity and the rotation of the turbine blades. These variables are:

$$(\overline{u})_{n+1} = (1-\gamma_{vel})(\overline{u})^* + \gamma_{vel}(\overline{u})_n \quad (4.126)$$

$$u_{max,n+1} = (1-\gamma_{vel})u^*_{max} + \gamma_{vel}u_{max,n} \quad (4.127)$$

$$(\overline{\omega}_{NP})_{n+1} = (1-\gamma_{vel})(\overline{\omega}_{NP})^* + \gamma_{vel}(\overline{\omega}_{NP})_n \quad (4.128)$$

$$(\overline{\Delta\omega^2})_{n+1} = (1-\gamma_{vel})(\overline{\Delta\omega^2})^* + \gamma_{vel}(\overline{\Delta\omega^2})_n \quad (4.129)$$

$$(4.130)$$

Please note that the blade parameter α is not here – α is a unique case requiring special treatment, detailed below.

Calculation of relaxation parameters

As well as introducing numerical stability to the model, relaxation can also be seen as an algorithmic interpretation of inertia. In reality, inertia and momentum of the turbine blades about their axis of rotation, and in blade pitch response, introduce a delayed response of the turbine to a change in flow conditions. We call these delays *relaxation times*, and we can make reasonable estimates of what they should be. The values presented are for wind turbines only: those for marine turbines differ, and are detailed in section 6.4.

Wind turbines can take up to a minute or more to speed up to full power, and so taking into account the inertia of the fluid itself we can set

$$t_{relax,vel} \approx 20s \quad (4.131)$$

The blade pitch adjusts more quickly, and so we can write approximately

$$t_{relax,\alpha} \approx 15s \quad (4.132)$$

For $t_{relax,vel}$ to be translated into γ_{vel}, we first ascribe a minimum value, to prevent the model becoming unstable:

$$\gamma_{min} = 0.5 \quad (4.133)$$

The tentative value is calculated as

$$\gamma_{vel}^{*} = 1 - \left(\frac{\Delta t}{t_{relax,vel}} \right) \tag{4.134}$$

where Δt is the timestep size of the simulation.

And so, the relaxation parameter is defined thus

$$\gamma_{vel} = \begin{cases} \gamma_{vel}^{*} & \text{if } \gamma_{vel}^{*} > \gamma_{min} \\ \gamma_{min} & \text{if } \gamma_{vel}^{*} < \gamma_{min} \end{cases} \tag{4.135}$$

As for α, if we recall the definition of $(\dot{\omega}_T)_{max}$ in (4.63):

$$(\dot{\omega}_T)_{max} = \frac{\omega_{T,max}}{\Delta t_\alpha}$$

Δt_α is effectively our relaxation time for α, so the equation above becomes

$$(\dot{\omega}_T)_{max} = \frac{\omega_{T,max}}{t_{relax,\alpha}} \tag{4.136}$$

Furthermore, from (4.67)

$$\Delta\alpha = \Delta t \left(\dot{\alpha} + n\dot{\alpha}_{max} \right)$$

It can be seen that as Δt is the simulation time step size, $\Delta\alpha$ will decrease in magnitude also; behaviour that we would expect from the relaxed equations using γ_{vel}.

Chapter 5
Software design

5.1. Introduction

5.1.1. Design ethos

From the outset, the turbine model was designed as an external component to 'piggy-back' on top of a computational fluid dynamics (CFD) software package, with no dependency upon that package apart from flow field variables.To this end, while it currently relies on the Imperial College finite element CFD software, there is essentially no reason why it could not interface with commerical packages such as CFX, which support external code [7]. Furthermore, while the turbine model should be capable of running on a modest, single-CPU workstation, it should be able take advantage of additional computing power where available.

This approach has fundamental technical implications:

1. **Code independence**. Routines within the turbine module cannot utilise routines within a particular CFD package: it must be effectively self-contained, to maintain portablity, reliability and interoperability. This is consistent with modular programming philosophy.

2. **Narrow external API**. Following from above, the API (application programming interface) visible to the external CFD program should be minimal: that is to say, only pass on data required to model the turbine, and no more. It also means that any data passed back from the module will also be minimal: that information passed back to the calling program is no more than it requires to perform fluid dynamics calculations.

Essentially, any information which can be passed to and from the turbine module which created a dependence on a particular CFD package, was rejected.

3. **Parallelism**. To fully exploit additional computing resources, the turbine module should be capable of running across multiple processors - this assumes the underlying CFD software is capable of parallelism also.

5.1.2. Interface to the module

The intention was that Fluidity would call the turbine module each time-step prior to its Navier-Stokes solver, so that any terms passed back from the module could be used to solve the flow for that time-step.

Due to the narrow API requirement, information passing was kept to a bare minimum. The data flow can be represented as in the following diagram.

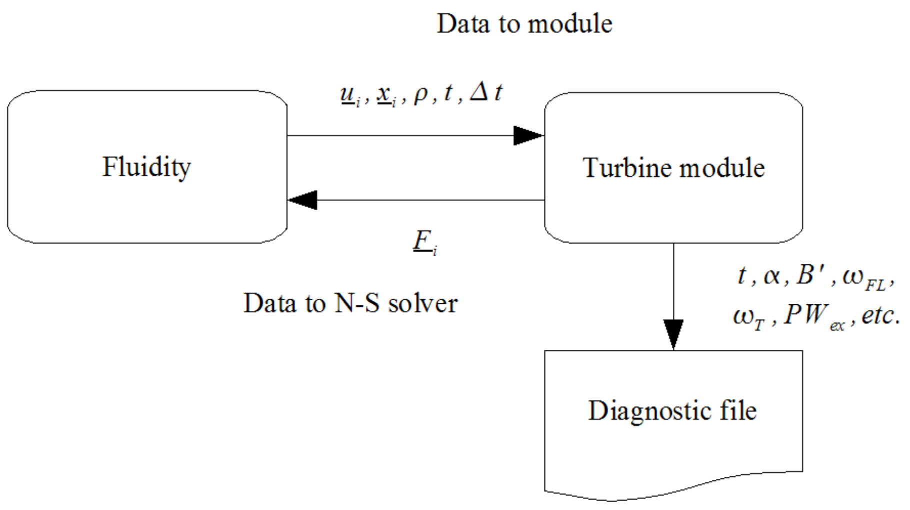

Data to module
$\underline{u}_i, \underline{x}_i, \rho, t, \Delta t$
Fluidity
Turbine module
$\underline{F}_i$
Data to N-S solver
$t, \alpha, B', \omega_{FL}, \omega_T, PW_{ex}, etc.$
Diagnostic file

Figure 5.1: Flow of data to and from the turbine module per timestep

Essentially, physical properties of the fluid are passed in, and force per-unit-volume terms are passed back to the CFD program for solution of Navier-Stokes: this is done via a bridging interface - this will be discussed further in section 5.2.

5.1.3. Parallel programming

Multiple instruction, multiple data

Fluidity uses a MIMD (multiple instruction multiple data) parallel programming model and so this was the technique chosen for the turbine model. It is a common choice for implementing parallelism in numerical simulation, and allows for easy deployment on workstations, computing clusters, and supercomputers, since the necessary software is readily available for each.

In a MIMD data model, several processes run in parallel on different processors, each having its own local memory. Typically, the computational problem has been carved up into similarly-sized chunks, and each portion allocated to a seperate processor – usually each one runs a separate instance of the same program, but with different data. This means that, while the array A in your program may be filled with certain values on one processor, the same array may have completely different values on a different processor. In fact, it is more than likely that A will be associated with a different part of the solution space.

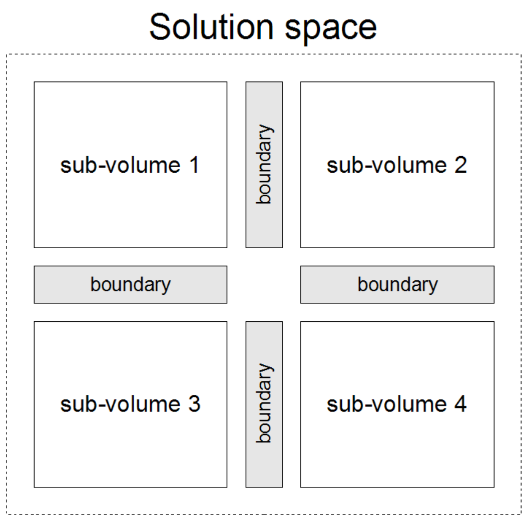


Figure 5.2: MIMD division of solution space with boundaries

Each process has a boundary in the computational problem, which it shares with neighbouring processes – in Fluidity these are called *halos* – and so to keep the values in this shared space current, each process regularly communicates with its neighbours and exchanges information regarding the boundary regions.

The strength of using MIMD techniques in parallel computing is that, due to the subdivision of the solution space into smaller, more manageable chunks, large computational problems can be tackled with an adequate number of processors. However, there are implications for their use, which affect the nature of the problems that can be solved, and how algorithms can be implemented:

1. **Communication overhead**. As all data is local to each process, interprocess communication becomes essential for computation – but at a price. This communication, through a shared memory bus or especially across a local network, introduces latency which must be minimised if full CPU utilisation is be achieved. As such, this affects the design of any algorithm employing MIMD techiniques.

2. **Increased computational complexity**. Again, due to the local nature of simulation data, the calculation of global variables is not always straightforward: collation is required, but bearing in mind any communication induced latency.

These had design consequences for the turbine module, which shall be discussed in more detail in later sections.

The master-slave technique

Master-slave protocol is often used in a MIMD environment, where one process (the master) has control over all the other processes (the slaves). While each process still solves its own part of the problem, the master process takes on the additional responsibilities of co-ordinating the slaves, and of handling data output. The turbine model was implemented using the master-slave technique.

Often in this situation all the processes run the same program, with conditional statements in the code to branch off into either master or slave behaviour. A simulation might consist of the following steps:

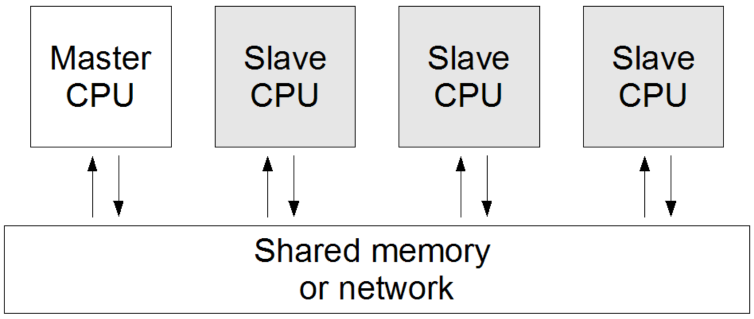


Figure 5.3: MIMD division of solution space with boundaries in master-slave arrangement

1. Start master process
2. Start slave processes
3. Each process identifies itself as either the master, or a slave
4. Initialise simulation and sub-divide problem
5. Start of iterative loop: solve subvolume locally
6. Exchange boundary data
7. Global variables calculation

   - **If master:** receive data from slaves, calculate global values and output results

   - **If slave:** send data to master

8. End of iterative loop. Repeat loop until end conditions met.

MPI (Message-Passing Interface)

MPI is a high-level communications protocol for implementing parallel software. It is hardware independent, and supports MIMD-style programming. One of its main virtues is that while it can utilise a number of communications infrastructures such as shared memory bus or TCP/IP, this remains invisible to the programmer. Amongst the hardware arrangements it can support are:

1. Single-core workstations (ie. older workstations/PCs). Commonly, one process will run only, as 'master'.
2. Multi-core workstations/servers. Here, communications run on the shared-memory bus.
3. Distributed computing (Beowulf) cluster. TCP/IP sockets are used for interprocess communication, since the processes will resides on a seperate machine.
4. Supercomputers (Cray , IBM). The MPI implementation will be vendor-specific, so as to utilise the unique hardware underneath.

While MPI is not an officially sanctioned standard, it is well-defined and so broadly used that it has many software distributions that are interoperable, and several which are freely available. Such distributions provide MPI libraries (for compiling MPI-aware programs), and MPI binaries (for running them). For this PhD, MPICH from the Argonne National Laboratory in the US was chosen [53], as it is available at no cost and supports a wide range of computing platforms. Fluidity and MPI were successfully installed and run on the following hardware:

1. Single core workstation running Linux (corryvreckan.eps.hw.ac.uk)
2. Multiprocessor server running Linux (moskstraumen.eps.hw.ac.uk)
3. Heriot Watt's distributed computing cluster (hwcluster.hw.ac.uk)

5.1.4. Development environment

In the preliminary stages, several programming languages were considered for implementing the turbine model: C, C++, Fortran 77 and Fortran 90. Due to Fortran's reputation as the *lingua franca* of numerical computing, it would be a convenient route for future collaborations. This reduced the choice down to Fortran 77 or its successor, Fortran 90. Fortran 90 offers a variety of benefits over Fortran 77, such as:

1. Derived types, ie. user defined variable types.

2. Dynamic memory allocation – arrays that have their size determined and memory allocated at run-time.

In particular, such features of Fortran 77 such as common blocks become redundant when using Fortran 90's derived types, which offer an elegant method of encapsulating data, and thus subroutine parameters. This shall be discussed in more detail below.

The Intel Fortran compiler 'ifort' was chosen as the compiler for use in this PhD, under the free academic license, given that it out-performs other free compilers such as GFortran and G95 on the intended hardware platforms of AMD and Intel-based PCs, across a variety of computational problems [5].

5.2. The CFD module interface

This essentially acts as a bridge between the CFD package (Fluidity) and the turbine module itself. Our main reason for this is the 'code independence' clause: that is, to ensure that there is no Fluidity-dependent code within the module itself. It can be thought of as a routine which mediates the data Fluidity provides, presenting it to the turbine module in a format it can process (and vice versa). In this way, the design of the module itself can be kept in a generic form such that if, in future, support for another CFD package is desired, all that is required is the writing of another bridging interface.

The call to the module interface was placed within a loop for iteratively solving non-linear terms in the main controlling routine, prior to the source and absorption terms being added to the momentum equations (see figure 5.4).

5.2.1. Variables passed from Fluidity

At each time-step, Fluidity calls the module interface, and passes several fluid simulation state variables. These are listed in table 5.1; all are one-dimensional arrays and real numbers unless otherwise stated.

Name	**I/O**	**Description**
X, Y, Z	in	Arrays describing the position of each mesh node: $X = \{x_1, x_2, ...x_N\}$ for a mesh of N nodes; similarly for Y and Z.
U, V, W	in	The x, y and z components of the flow field. $U = \{u_1, u_2, ...u_N\}$; similarly for V and W.
ρ	in	Density of fluid at each node. The turbine algorithm currently only deals with incompressible fluids, and so only the first value is used. This allows for future expansion to compressible flows, however.
XAB, YAB, ZAB	out	x, y and z components of momentum absorption terms, of unit $\frac{1}{time}$. These are uninitialised prior to calling the module, to be returned with values calculated by the turbine model.
$XSOU$, $YSOU$, $ZSOU$	out	The components of the momentum source terms. As with the absorption terms, these are passed empty and returned with calculated values.
t_{acc}	in	Number containing the accumulated simulation time.
Δt	in	The time-step size for the simulation.
I_{nonlin}	in	Fluidity can do several non-linear iterations to solve Navier-Stokes, per time-step; this number tells which iteration it is on.
N_{nonlin}	in	The total number of non-linear iterations per time-step.
F_{remesh}	in	Boolean flag which is set to *true* if finite element mesh has been adapted (ie. nodes moved, or added/deleted) since last time-step.

Table 5.1: List of all variables passed to module by Fluidity and returned

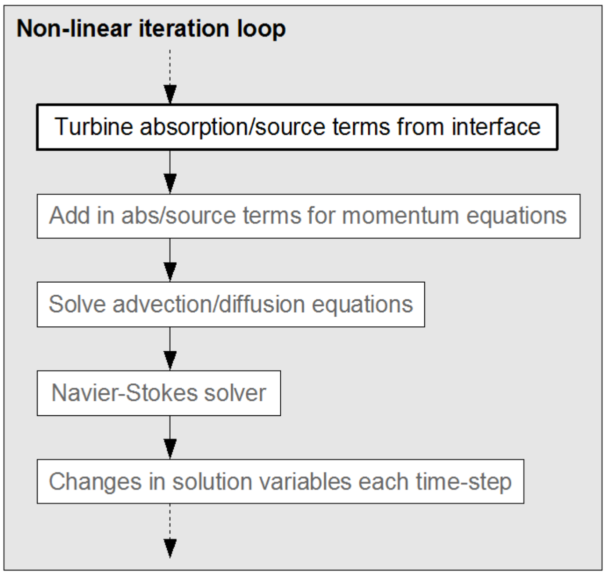

Non-linear iteration loop
Turbine absorption/source terms from interface
Add in abs/source terms for momentum equations
Solve advection/diffusion equations
Navier-Stokes solver
Changes in solution variables each time-step

Figure 5.4: Position of module interface call in Fluidity's main control routine

5.2.2. Calling the module

The variables passed from Fluidity are simply passed straight to the module for calculation. However, it is not simply a straightforward case of calculating the absorption and source terms from the node values each time the module is called; Fluidity has a few behaviour peculiarities which necessitate conditional operation.

To this end, we have two flags which control the behaviour of the turbine module.

1. F_{update}. A boolean flag set to $true$ or $false$. When passed to the turbine module, this tells it to update (ie. recalculate) global variables of the turbine such as ω_T etc., or whether to simply re-use current values.

2. F_{output}. This boolean flag tells the module whether it should output diagostic data such as ω_T, PW_{ex} for this time-step.

The global variables of the turbine,such as ω_{FL} (from which the absorption and source terms are calculated),are only updated when $F_{update} = true$. This is just after Fluidity has run its adaptive mesh algorithm: when adapting, it

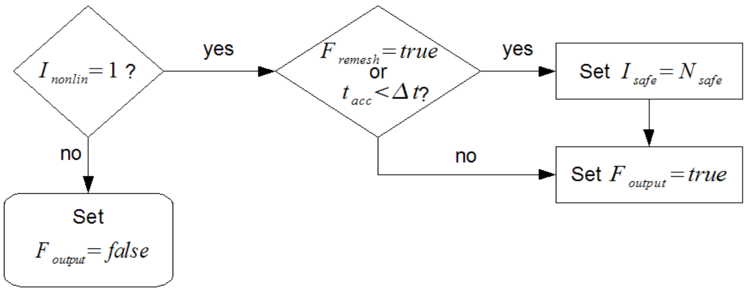


Figure 5.5: Flowchart deciding on output and remesh time-step counter

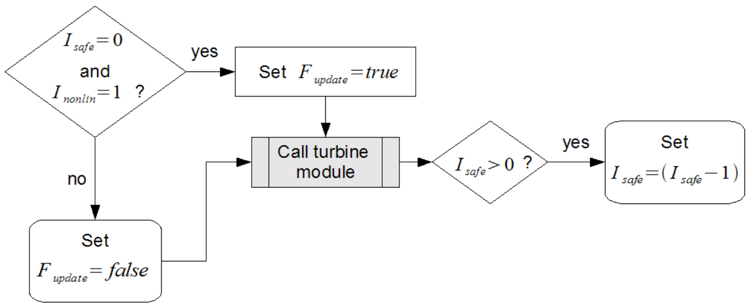


Figure 5.6: Decision flowchart for output flag and calling the turbine module

relies on interpolation to gradually correct the boundary regions to realistic conditions through a 'smearing effect'. Thus fluid properties in these regions are unreliable for N_{safe} timesteps afterwards, and so are temporarily ignored. Global turbine variables are frozen briefly, and momentum absorption/source terms are calculated using these until the mesh safety counter I_{safe} reaches zero. Experimentally $N_{safe} = 3$ gave the best results.

The interface routine also checks whether it is on the first non-linear iteration or not. If it is not, F_{output} and F_{update} are set to $false$. This way, the turbine variables are only updated and output once per time-step. The turbine module does however still calculate and return the absorption and source terms, as these are still required by the Navier-Stokes solver.

5.3. The module

The module itself can be thought of as divided into two stages: initialisation, and processing. The initialisation stage takes place only once, creating the data structures for the turbine or turbines, which are then kept as persistent variables between iterations. The processing stage, which occurs for each time-step where $F_{update} = true$, can be thought of us having three sections:

1. **Preparation** where turbine nodes are found and their coordinate system changed
2. **Calculation** where turbine and nodal variables are calculated
3. **Variable collection and output** where parallel variables are polled and data written to file.

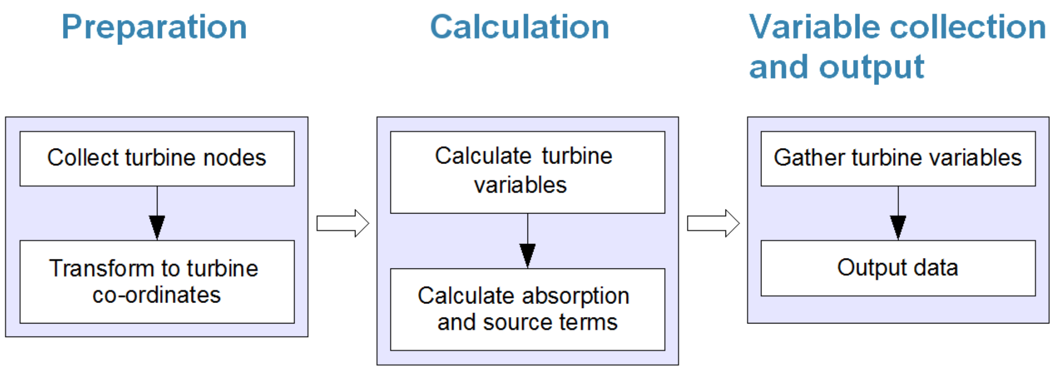

Preparation
Calculation
Variable collection and output
Collect turbine nodes
Transform to turbine co-ordinates
Calculate turbine variables
Calculate absorption and source terms
Gather turbine variables
Output data

Figure 5.7: Subdivisions of turbine module and main subroutines

These sections will be covered in more detail below.

5.3.1. Preparation

Collecting turbine nodes

Here, the program determines which nodes are contained by the turbine volume V_T and adds them to a list of nodes for later calculations. In a serial program, this would be straightforward; however, in parallel it is quite possible that only some instances of Fluidity within a parallel simulation contain turbine nodes.

Or indeed, that nodes within the turbine volume reside on all of them – fig. 5.8 demonstrates this more clearly.

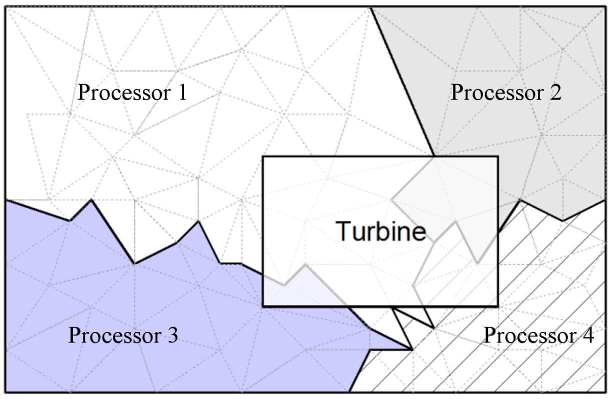


Figure 5.8: An example of the turbine volume cohabiting across several processors. Each shaded mesh represents a subdomain of the computational volume that resides on a processor.

The basic algorithm is detailed below. Assume that we are building a set of nodes $S = \{n_1, n_2, ...\}$ that are contained within the turbine volume.

For each processor subdomain p:

1. For each node i in p, check if in V_T.
2. If i is in V_T, add it to S.
3. Send local copy of S to all other processors
4. Receive copies of S from other processors, call them S'.
5. For each node i in S, and each node j in S':
    (a) If $|\underline{x}_i - \underline{x}_j| < \epsilon$ where ϵ is some pre-defined error tolerance (ie. are considered duplicate nodes from the subdomain halo), then let $|\underline{u}|_i = |\underline{u}|_j$ if $|\underline{u}|_j > |\underline{u}|_i$.
    (b) Otherwise, add node j to S

In this way, a list of all nodes from all the subdomains contained in V_T was built up.

Transforming to turbine co-ordinate system

Once the set of turbine nodes S was built up, then the nodes had to be rotated and translated to a turbine's co-ordinate systems. Referring to section 4.4.1, for each node i in S, then the necessary position and velocity transformations are

$$\underline{x}_i' = R_{-\phi} \begin{bmatrix} x_i - x_T \\ y_i - y_T \\ z_i - z_T \end{bmatrix} \tag{5.1}$$

$$\underline{u}_i' = R_{-\phi}\, \underline{u}_i \tag{5.2}$$

where $\underline{x}_i'$ and $\underline{u}_i'$ represent the transformed positions and velocities which are used to calculate the forces upon the fluid.

Correcting halo values

As previously mentioned in section 5.2.2, the node values within the halo sections of the subdomains are unreliable. To this end, a crude interpolation algorithm was used to 'fill in the blanks' where necessary.

Returning to our set S, for each node i in S, if $|\underline{u}_i| < \epsilon_u$ and $r_i > R_H$, where ϵ_u is a given, small error tolerance, then we let

$$u_i = \max(u) + \left(\frac{du}{dx}\right)^* d_i \tag{5.3}$$

where d_i is taken at the x displacement of node i from the inlet of the turbine, and $\left(\frac{du}{dx}\right)^*$ is an approximation to the gradient of u defined as $\frac{du}{dx}^* = 2\left(\frac{\overline{u}-\max(u)}{L}\right)$.

The y and z components of the velocity are specified as

$$v_i = -\omega_{FL}^* z_i \tag{5.4}$$

$$w_i = \omega_{FL}^* y_i \tag{5.5}$$

where $\omega^*_{FL} = (1-Q)\sqrt{2} f \alpha \beta' \left(\frac{d}{L}\right) u$, an estimation of the local fluid angular velocity.

5.3.2. Calculation

The blade pitch parameter α

Before any other variables α for the current time-step was calculated; see section 4.3.5 for details.

Main variables

Here, the main turbine variables are calculated and expressed in relaxed form. The first step is to calculate the instantaneous means of these variables, which are

1. The mean free-wheeling angular velocity of fluid within the turbine volume V_T, defined as

$$\overline{\omega}_{NP} = F\alpha \frac{1}{N} \sum_{i=1}^{N} \left(\frac{\beta'_i u_i}{r_i} \right) \tag{5.6}$$

.

2. The mean value of u of the x-component of the node velocities within V_T.

$$\overline{u} = \frac{1}{N} \sum_{i=1}^{N} u_i \tag{5.7}$$

3. The mean value of u within the hub volume

$$\overline{u}_H = \frac{1}{M} \sum_{j=1}^{N} u_j \tag{5.8}$$

where $j = 1...M$ represents the nodes contained within the hub volume.

4. $\overline{\Delta\omega^2}$, a component of the power equation, which is defined in (4.120) as

$$\overline{\Delta\omega^2} = \frac{1}{N} \sum_{i=1}^{N} \left(\omega^2_{NP,i} - \omega^2_{FL,i} \right) \tag{5.9}$$

These were then relaxed as per section 4.4.6.

5.3.3. Variable collection and output

In a parallel simulation where N_P processor subdomains contain parts of V_T, it stands to reason that there will also be N_P copies of the turbine variables. Even though each CFD process has gone through the same node collection routine as the others, it was found that small discrepancies crept in between the values held on each processor.

This presented a problem: since the time-dependent turbine variables were to be saved to file, which sets of values were to be used? Furthermore, to prevent file contention, only one process was allowed to write to the results file – the master.

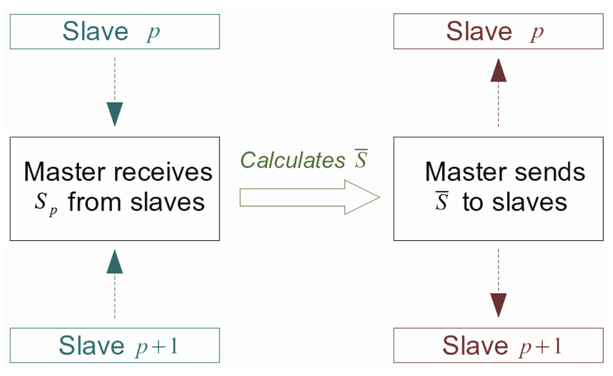

Slave p
Master receives S_p from slaves
Slave $p+1$
Calculates $\overline{S}$
Slave p
Master sends $\overline{S}$ to slaves
Slave $p+1$

Figure 5.9: The collation of turbine variables in parallel simulation

The solution was to synchronise the turbine variables at each time step, and write those values to file. It was decided that, rather than have one processor's variables overwrite those of the others, a weighted average would be calculated which would be used by all the simulation processes. Assuming we have set of turbine variables for each processor p, for the current and last timesteps, defined as $S_p = \{\overline{u}_p(t), \overline{u}_p(t - \Delta t), (\omega_{FL})_p(t), (\omega_{FL})_p(t - \Delta t), ...\}$, then the procedure for synchronisation was as follows:

1. **If master process:**

   (a) Await copies of S_p from slave processes

(b) When received, calculate weighted average of set as

$$\overline{S} = \frac{1}{M} \sum_{p=1}^{N_P} M_p S_p \tag{5.10}$$

where M is the total number of mesh nodes on all the processors, and M_p the mesh nodes on each processor p (both excluding nodes collected from other subdomains).

(c) Send $\overline{S}$ to each slave.

(d) Write $\overline{S}$ to results file.

2. **If slave process:**

(a) Send local copy of S_p and M_p to master

(b) Receive $\overline{S}$ from master

(c) Set $S_p = \overline{S}$.

In this way, each S_p would remain close in value to every other. As more significant deviations usually occured in subdomains where M_p was small, the weighted average of $\overline{S}$ insured that they would not unduly influence the results.

Chapter 6

Experiments and experimental process

6.1. Overview

6.1.1. Aims

The model was validated by testing it at a variety of different windspeeds. This had three points of attack.

1. By running a succession of simulations each with different freestream velocity u_0 via a Dirichlet inlet condition, and comparing the performance of the turbine model as function of u_0 with that of real turbines.
2. Looking at the wake structure for each value of u_0, and comparing it with known theory and experimental data.
3. Comparision of turbulence profiles with turbulence measurements of real wind turbines.

6.1.2. Structure

The experiments can be thought of as separated into two sections. The first section concerns itself with wind turbines in a wind tunnel: this can be seen as an exercise in validation of the turbine model. Even though the eventual target was to model axial-flow marine turbines, the wealth of experimental data and theory regarding wind turbines and their wakes made them a compelling first port of call.

With marine turbines, there is a comparative dearth of detailed, published experimental data: partly through marine power being relatively new territory, and presumably partly through protection of commercial interests. Thus the assumption is made that that a horizontal axis marine turbine is, in effect, similar to a horizonal axis wind turbine. We can say this – even though axial-flow marine turbines cannot use stall to regulate due to blade scouring from cavitation [30], so instead relying on blade pitch control – because the model makes no distinction between the two, lumping both into its blade parameter α.

Therefore, the second section concerns itself with transforming the wind turbine model into one of a marine turbine, based on as few assumptions as possible. In this part, the simulations progress from a simple water tunnel to more realistic environments, introducing bottom drag and a vertical velocity gradient at the inlet.

6.2. The turbine presence field

Throughout prelimary simulation runs, one problem that presented itself was ensuring the the turbine volume would contain enough nodes so that the model would perform effectively. Whilst the turbine-induced velocity gradients would appear to trigger Fluidity's adaptive mesh algorithm, if the turbine relaxation time was large, nodes would be removed from the turbine volume until none were left – at that stage, the turbine effectively disappears.

To ensure that this never happened, a non-advective, non-diffusive tracer property was introduced to the fluid. It was defined as a function of radial and axial distance, and constrained to the limits $0 \leq f_{pres} \leq 1$. To calculate the turbine presence field, an algorithm would search for nodes within a greater cylinder encompassing the turbine volume, with dimensions $L' = 5L$ and $R'_T = 2R_T$. This would ensure that even if the initial mesh nodes lay outwith the turbine volume, the adaptive algorithms would still be triggered to concentrate nodes within it.

For a given node i, the turbine presence was calculated as

$$f_{pres,i} = \left[\left(1 - \frac{2|x_i|}{L'}\right) \cdot \left(1 - \frac{1 - r_i}{R'_T}\right)\right]^2 \qquad (6.1)$$

where $r_i = \sqrt{y_i^2 + z_i^2}$

A variety of alternatives were tried, such as simple linear ramp functions, but this one performed the most consistently. Figure 6.1 shows the effect of several adaptive sweeps.

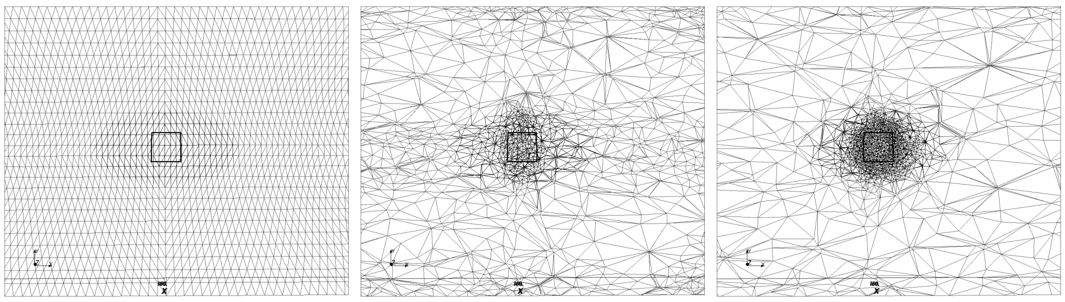


Figure 6.1: A planar slice through a three-dimensional mesh, on three successive sweeps from left to right. The central square is the turbine volume.

6.3. Wind turbines

6.3.1. Choice of turbine

The Vestas V52 horizontal axis wind turbine was chosen for the wind turbine simulations, produced by Vestas, who have almost a third of the wind turbine market worldwide according to their website [3]. The V52 is still in production with 2100 in use across the world [8]. Moreover, it features active blade pitch control, in what Vestas call 'OptiTip', which is very similar to the control of the power output by α in the turbine model. For these reasons, it was deemed a good candidate for modelling.

6.3.2. Defining turbine properties

Solidity and effective solidity

The net solidity B can be found using photographs of said turbine, and a graphics manipulation program such as the freely available GIMP. The process is as follows.

1. Firstly, an image must be taken of the turbine effectively 'head on', ie. where the camera has been positioned pointing directly at the nacelle, and parallel to the turbine axis. Figure 6.2 shows a V52 taken from the media section of the Vestas website.

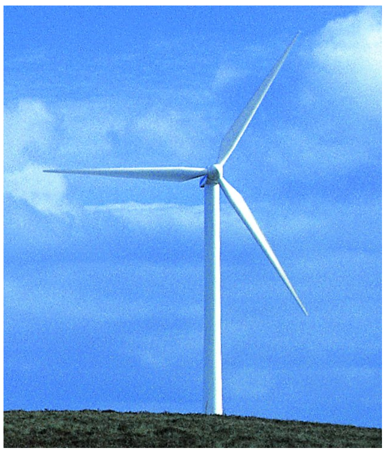


Figure 6.2: V52 from Vestas website

2. Then we select the blades and nacelle in the image, and remove the background as can be seen in figure 6.3.

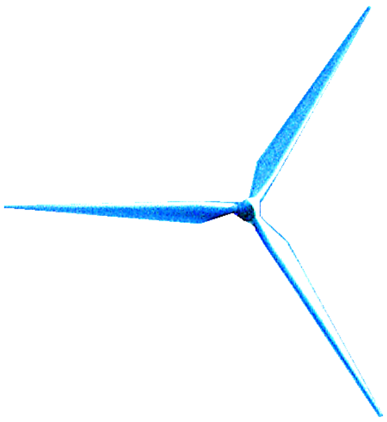


Figure 6.3: V52 blades selected from photograph

3. The blades are then coloured black, and a circle of a different colour fitted to the blades, eg. as in figure 6.4.

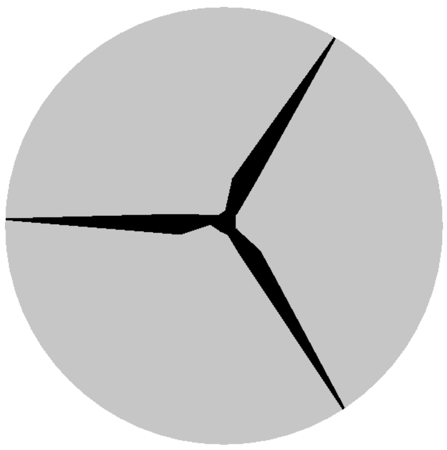
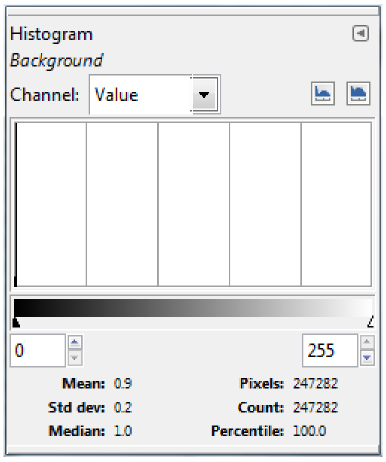


Figure 6.4: V52 blades contrasted and encircled

4. Having reduced the image to two colours, discounting the background, we can can look at a histogram of the image colours, and calculate the percentage of blade pixels from the total number of pixels in the image (again, ignoring the background) – the V52 histogram can be seen in in figure 6.5.



```
Histogram
Background
Channel: Value

0                                   255
     Mean: 0.9             Pixels: 247282
  Std dev: 0.2              Count: 247282
   Median: 1.0         Percentile: 100.0
```

Figure 6.5: V52 image histogram

Depending on how the Vestas image was manipulated, the net solidity came out at $B \approx 0.04$.

The effective solidity however cannot be determined in such a way. While there is little reference to it in literature, after some trial and error $B' = 0.25$ was decided upon as a good value.

Hub solidity is express in terms of the local solidity. As the hub/nacelle appears completely solid to the flow, we set

$$\beta_H = 1.0 \tag{6.2}$$

That is, $\beta_H = \beta$ where $0 < r < R_H$.

Blade and hub cross-sectional geometry

From inspection of figure 6.4 and using published specifications, we see make further reasonable estimates:

- Total turbine radius defined in [8] as $R_T = 26\,\mathrm{m}$.
- A tip-width fraction of $w \approx \frac{1}{10}$
- $R_H \approx \frac{1}{10} R_T$, also assumed by Sharpe [66].
- R_{tip} is expressed as a fraction of R_T. It defines the radial distance from the hub axis beyond which tip turbulence operates on the flow. The value is set to ensure that the thin tubular volume created contains at least a set of mesh nodes right around, and from end to end. Thus it is dependent upon the minimum allowed size of elements; too small and no nodes would be contained within the annular volume, and so no tip turbulence would be generated.

  For the V52 simulations, $R_{tip} = \frac{3}{4} R_T$.
- L, the length of the turbine volume. This length is constrained by the minimum element size of the mesh (smaller elements means a larger mesh, and slower simulation), and the Reynolds number of flow through the blade volume: if we make L too large, Re_x will adversely affect the flow.

  Since no data suggesting a suitable value could be found on the V52, from observation of similar wind turbines, the cylindrical volume a real V52 occupies would have $L \approx 2 - 3\,\mathrm{m}$. Thus, setting $L = 10\,\mathrm{m}$ gives the model an Re_x of the same order of magnitude as the real tubine.

Parameters from performance graphs: optimum power efficiency, cut-in/cut-out speeds, etc.

These parameters were sourced from the wind power calculator on the Danish Wind Industry Association website [2] and from the V52 brochure [8]. In figure 6.6 we can see the power coefficient as a function of wind speed.

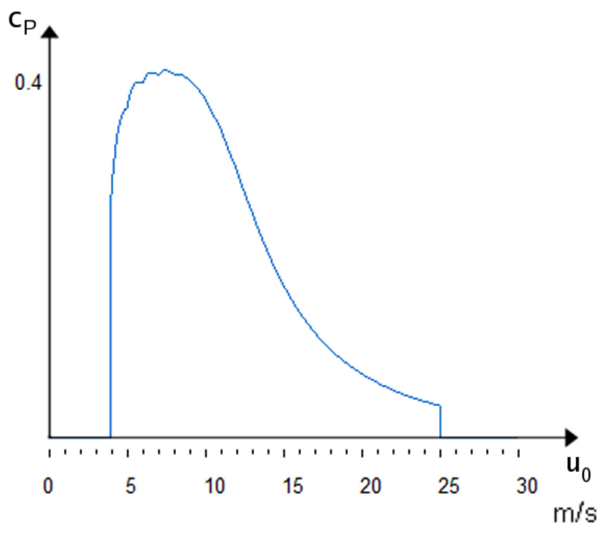


Figure 6.6: Wind speed versus c_P for Vestas V52 (Courtesy of Danish Wind Industry Association website)

- $u_{0,opt}$, the freestream windspeed at peak efficiency. This is u_0 when C_p is at its maximum. The Danish Wind Industry website wind power calculator gave this as 11 m/s; the V52 brochure as 13 m/s. After some deliberation, from figures 6.6 and 6.7 it was decided to set $u_{0,opt} \approx 12$ m/s.

- $c_{p,opt}$, the optimum power efficiency. From graph we can take this as $c_{p,opt} = 0.4$.

- $\omega_{T,opt}$, the angular velocity of the turbine blades when the turbine is operating at peak efficiency (ie. at $u_0 = u_{0,opt}$).

  If we make reasonable assumptions from the V52 RPM statistics (see Rotor section in [8]), then we can estimate $\lambda = 7$ which gives us $\omega_{T,opt} \approx 3.17$.

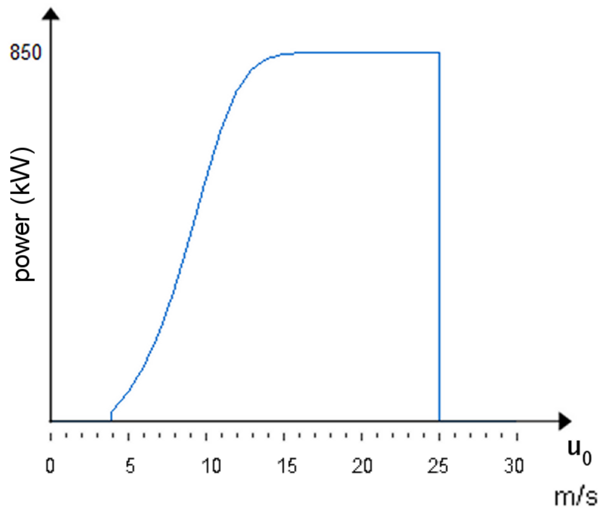


Figure 6.7: Wind speed versus PW_{ex} for Vestas V52 (Courtesy of Danish Wind Industry Association website)

- $u_{0,cutout} = 25\,\mathrm{m/s}$ from figure 6.6 and [8].
- $u_{cutin} = 4.0\,\mathrm{m/s}$ and $u_{cutout} = 23.5\,\mathrm{m/s}$ are effectively the values of $max(u_i)$ (where i is any node within the turbine volume), at $u_{0,cutin}$ and $u_{0,cutout}$ inlet wind speeds respectively.

  It should be noted that these were found imperically; even though the turbine model uses the approximation $u_0 \approx 1.1\,max(u_i)$ for turbulence calculations (4.105), during simulation it was found that close to cut-in and cut-out wind speeds the approximation was unsatisfactory.
- $c_{P,cutout}$ – the power coefficient just below the turbine cutout speed. Given the air density, cut-out speed and maximum power output (from figure 6.7, this was calculated to be $c_{P,cutout} \approx 0.042$.

Fluid dynamic properties

The flow factor f, and the momentum extraction efficiency Q, were perhaps the most difficult parameters to define, due to their abstract nature. However, they do have parallels in actuator disc theory – f is related to the blade lift coefficient in that it multiplies the effect of airflow across the blades, ie. the lift-generated thrust; and Q to the axial flow induction factor, in that it represents the portion of forward fluid momentum lost due to energy extraction.

Using these as guides initially I set $f = 2.0$ and $Q = 0.5$. The turbine model was then run with an Dirichlet inflow condition of $u_0 = 12\,\mathrm{m/s}$, and these values adjusted until $c_{P,opt}$ occured at $u_0 \approx u_{0,opt}$, where $\alpha \approx 1$. Thus the two fluid dynamic properties were ascertained experimentally as $f = 1.95$ and $Q = 0.65$.

Turbulence

Ti_{opt} represents the tip turbulence generated when $u_0 = u_{0,opt}$. This can be set as $Ti_{opt} = 0.15$ (Hossain [37]). Gomez-Elvira [35] has this slightly higher at $Ti_{opt} = 0.20$, but it should be noted that this only an approximate value, since some turbulence will be generated by the rotation of the wake itself.

Relaxation times

We estimate that such wind turbines taken several minutes to achieve full speed in peak conditions, therefore a reasonable relaxation time would be $t_{relax,vel} = 20\,\mathrm{s}$

Similarly, the blade relaxation time was set to a lower value (in order that it could quickly counteract over-performance. Thus $t_{relax,\alpha} = 10\,\mathrm{s}$

6.3.3. Vestas V52, without inlet turbulence

Overview

The simulation domain consisted of a rectilinear channel – a cuboid wind tunnel – as seen in figure 6.8, with a Dirichlet inlet condition on the left end at $x = 0$, and an open bounday at the right end. The four walls had a no-slip condition, thus frictionless.

Wind tunnel dimensions

The wind tunnel was made long enough to ensure that the wind turbine wake could fully develop (ie. 20 diameters downwind), and yet enough clearance

Name	**Symbol**	**Value**
Radius of turbine	R_T	26 m
Hub radius	R_H	$\frac{1}{10}R_T$
Hub local solidity	B_H	1.0
Tip width fraction	w	$\frac{1}{10}$
Tip radius	R_{tip}	$\frac{3}{4}R_T$
Turbine volume length	L	10 m
Net solidity	B	0.04
Optimum net effective solidity	B'_{opt}	0.25
Flow factor	f	1.95
Momentum extraction efficiency	Q	0.65
Optimum freestream speed	$u_{0,opt}$	12 m/s
Optimum blade angular velocity	$\omega_{T,opt}$	3.17 rad/s
Optimum power coefficient	$c_{p,opt}$	0.4
Freestream wind speed at turbine cutout	$u_{0,cutout}$	25 m/s
Power coefficient just below cut-out windspeed	$c_{P,cutout}$	0.042
$max(u_i)$ at cut-in and cut-out	u_{cutin}, u_{cutout}	4.0 m/s, 23.5 m/s
Tip-generated turbulence at $u_0 = u_{0,opt}$	Ti_{opt}	0.15
Turbine angular velocity relaxation time	$t_{relax,vel}$	20 s
Blade angle factor relaxation time	$t_{relax,\alpha}$	10 s

Table 6.1: List of physical properties of model of Vestas V52

given to the front and sides so that blockage of the flow by the turbine would not significantly affect the flow. Thus

$$V_{tunnel} = L_x \times L_y \times L_z = 1400\,\text{m} \times 400\,\text{m} \times 400\,\text{m} \tag{6.3}$$

Turbine positioning

The turbine was positioned in the centre of the incoming flow near the left end, with its axis parallel to it. Its position was given as

$$\underset{\sim}{x}_T = \begin{bmatrix} 200\,\text{m} \\ 200\,\text{m} \\ 200\,\text{m} \end{bmatrix} \tag{6.4}$$

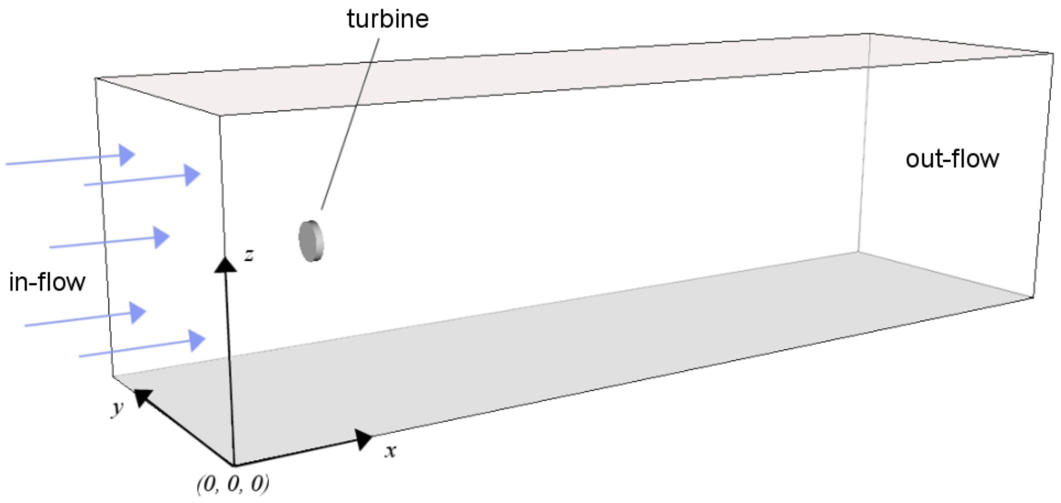


Figure 6.8: The Vestas V52 simulation domain

Fluid properties

The air was taken to be incompressible, and assuming a temperature of 15^o C and idealised standard atmospheric conditions, we can assume the fluid dynamical properties are as in table 6.2.

Inlet conditions and adaptive mesh configuration

For each simulation run, the inlet wind speed was changed, to see how the turbine model would perform in different conditions. These inlet wind speeds

Name	**Symbol**	**Value**
Density	ρ	$1.23\,\mathrm{kg\ m^{-3}}$
Dynamic viscosity	μ	$1.80 \times 10^{-5}\,\mathrm{Pa\ s}$

Table 6.2: Fluid properties of simulation air

u_0 **(m/s)**
5
9
10
12
15
17.5
20
22.5
25

Table 6.3: List of inlet wind speeds

are listed in table 6.3.

The tetrahedral mesh has two sets of parameters: those which are invariant with the inlet flow speed u_0, and those which are not.

The variable mesh parameters mainly concern themselves with sensitivity to velocity gradients. As u_0 changes for each experiment, these must be altered so that the adaptive algorithm does not under represent features of the flow by excessive deletion of mesh nodes, or by over representing them by adding so many nodes the simulation slows to a crawl. These variables are listed in table 6.5, and do not vary linearly with u_0, since the behaviour of the turbine and the velocity gradients it induces do not vary linearly with u_0.

Name	**Symbol**	**Value**
minimum x length of element	$x_{ele,min}$	2.0 m
minimum y length of element	$y_{ele,min}$	2.0 m
minimum z length of element	$z_{ele,min}$	2.0 m
maximum x length of element	$x_{ele,max}$	20.0 m
maximum y length of element	$y_{ele,max}$	20.0 m
maximum z length of element	$z_{ele,max}$	20.0 m
initial divisions along x-axis	N_x	140
initial divisions along y-axis	N_y	40
initial divisions along z-axis	N_z	40

Table 6.4: List of invariant settings for adaptive mesh

Name	**Symbol**
error tolerance for $\frac{\delta u}{\delta x_i}$	Δe_u
error tolerance for $\frac{\delta v}{\delta x_i}$	Δe_v
error tolerance for $\frac{\delta w}{\delta x_i}$	Δe_w

Table 6.5: List of variable adaptive parameters

u_0 (**m/s**)	Δe_u	Δe_v	Δe_w	Δt (s)	$LTIME$ (s)
5	0.05	0.05	0.05	0.15	600
9	0.1	0.2	0.2	0.09	600
10	0.15	0.2	0.2	0.08	900
12	0.25	0.1	0.1	0.05	900
12.5	0.2	0.2	0.2	0.05	600
15	0.2	0.2	0.2	0.05	500
17.5	0.225	0.2	0.2	0.04	429
20	0.3	0.25	0.25	0.035	375
22.5	0.35	0.275	0.275	0.03	333
25	0.4	0.3	0.3	0.025	300

Table 6.6: Inlet wind speed, error tolerances, time-step size and simulation duration

6.3.4. Vestas V52, with inlet turbulence

Overview

These set of simulations are identical to those in section 6.3.3, differing only in the inlet conditions. To investigate the effect of incoming turbulent flow on the modelled turbine, an algorithm was implemented to generate turbulence at the inlet on the left.

Generating inlet turbulence

The inlet turbulence algorithm works on the same principle as the tip-generated turbulence in section 4.3.6, ie. generating turbulence by accelerating the fluid at each node within a defined volume in a Gaussian random manner, to kick-start the LES turbulence modelling. This volume was defined as a thin narrow strip near the entrance to the wind tunnel, as in figure 6.9.

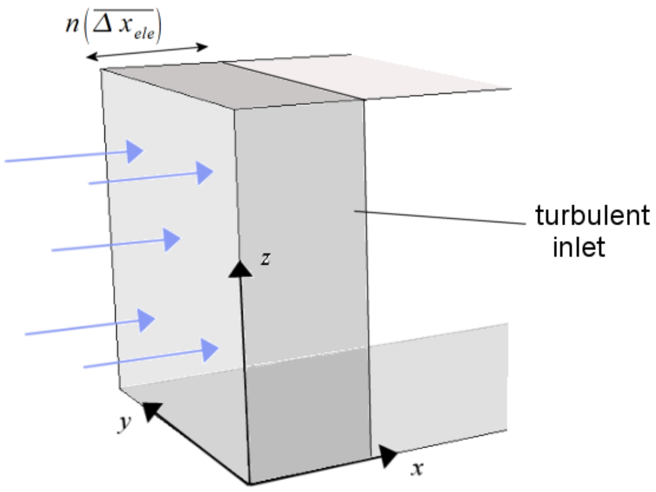


Figure 6.9: The turbulent inlet of the wind tunnel

A node is considered in this strip if

$$0 < x_i < n(\overline{\Delta x_{ele}}) \qquad (6.5)$$

where $\overline{\Delta x_{ele}}$ is the mean x-component length of an element within the turbine volume – a good indicator of the general element dimensions. n is a small positive number to modestly increase the length of the turbulent strip; $n = 5$ gave the best results.

The turbulence algorithm was anisotropic, and so there were three separate components, one for each axis: Ti_x, Ti_y, and Ti_z. From the Danish standard DS472 (Wind Energy Handbook [16], pp21-22), we can define the y and z components as a function of Ti_x:

$$Ti_y = 0.8\, Ti_x \tag{6.6}$$

$$Ti_z = 0.5\, Ti_x \tag{6.7}$$

Furthermore, this specification allows us to define Ti_x independently of the freestream windspeed u_0. Figure 6.10 shows the upper and lower limits of this standard.

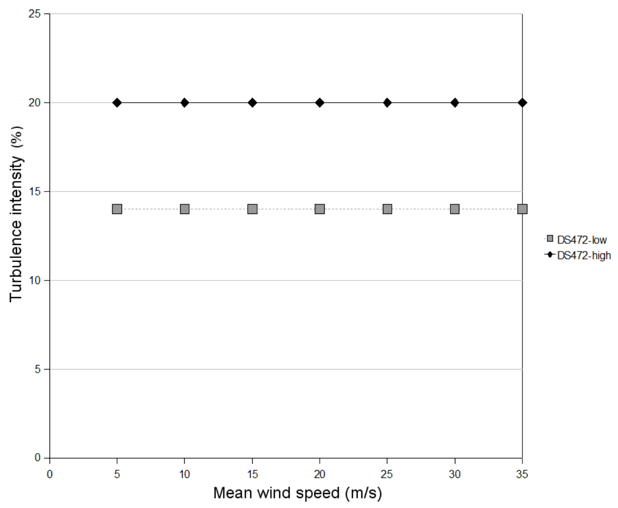


Figure 6.10: DS472 standard turbulence intensities for Ti_x at 50 m height

To put our turbulence comfortably within acceptable levels, $Ti_x = 0.15$ was chosen for all wind speeds.

In a similar fashion to equation (4.103), we calculate the changes in velocity for node i within the inlet region as

$$\Delta u_{inlet,i} = Ti_x\, G_x(t)\, u_0 \tag{6.8}$$

$$\Delta v_{inlet,i} = Ti_y \, G_y(t) \, u_0 \tag{6.9}$$

$$\Delta w_{inlet,i} = Ti_z \, G_z(t) \, u_0 \tag{6.10}$$

Where $G_k(t)$ are independent, normalised Gaussian random number generators, similar to before. To calculate the acceleration, we must calculate the time the fluid takes to travel through turbulent inlet. This is

$$\Delta t_{inlet} = \frac{(\overline{\Delta x_{ele}})}{u_0} \tag{6.11}$$

where u_0 is a good approximation of the x-component of the velocity within the inlet region.

The force-per-unit-volume applied at each node i to create turbulence is therefore

$$\underline{F}_{inlet,i} = \left(\frac{\rho}{\Delta t_{inlet}}\right) \begin{bmatrix} \Delta u_{inlet,i} \\ \Delta v_{inlet,i} \\ \Delta w_{inlet,i} \end{bmatrix} \tag{6.12}$$

6.4. Marine turbines

6.4.1. Choice of marine turbine

This was not difficult, since there are no open-water commercial axial-flow marine current turbines in production to this date. However, Marine Current Turbines Ltd. have produced a working prototype in Seaflow, which was installed 3 km from the coast of North Devon in England in 2003 (see press release [48]). This turbine was chosen as the basis of the model, using a mixture of the published technical details and assumptions from the wind turbine model.

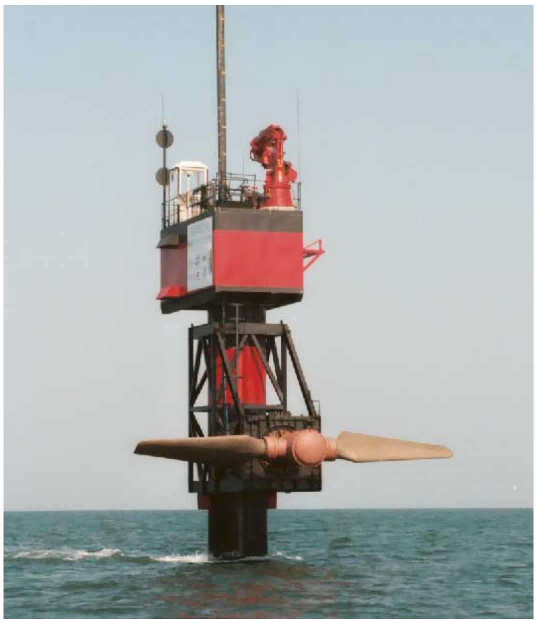


Figure 6.11: The Seaflow turbine, with blades raised for maintainence (courtesy of MCT website [1])

6.4.2. Defining turbine properties

Solidity and effective solidity

The process for defining b is identical to that used for the turbine

1. We take a photograph of Seaflow close to 'head on'. In this case, the best example to be found was figure 6.11.

2. Removing the background, clipping out the tower, and doing a little perspective correction, we arrive at figure 6.12.

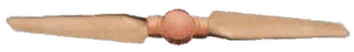


Figure 6.12: Perspective-corrected picture of Seaflow blades and hub

3. After that, the blades and hub were contrasted black, then a grey circle fitted to the blade tips, as can be seen in figure 6.13.

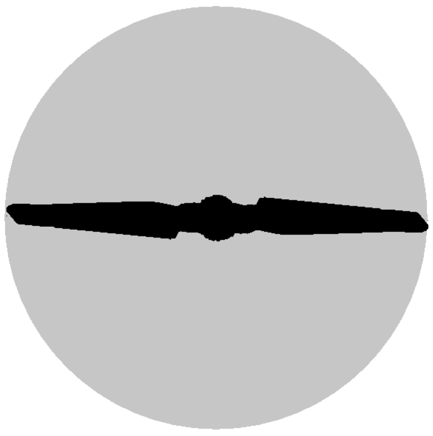
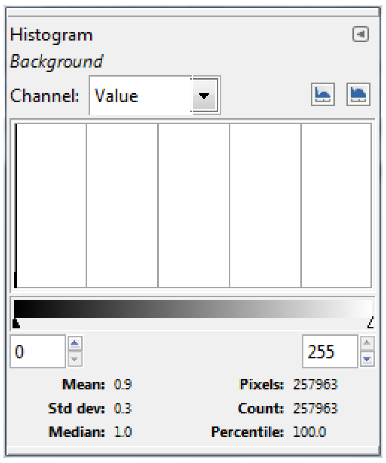


Figure 6.13: Seaflow blades and hub contrasted and encircled

4. Excluding the background the image is now two colours, and this produces the figure 6.14, the histogram of the indexed colours in the figure 6.13.



Histogram
Background
Channel: Value

0 255

Mean: 0.9 Pixels: 257963
Std dev: 0.3 Count: 257963
Median: 1.0 Percentile: 100.0

Figure 6.14: Seaflow image histogram

The smallest peak (left corner of the histogram) represents the number of pixels that are the blades and hub. This put $B \approx 0.1$.

As far as the effective solidity is concerned, due to the lack of any other data it was decided to keep this at a similar value for wind turbines, so $B' = 0.25$. Again, considering the hub completely solid to the flow, we have

$$\beta_H = 1.0 \tag{6.13}$$

Blade and hub cross-sectional geometry

By using figure 6.13 and published information, further estimates can be made:

- Total turbine radius defined in DTI Report [59] as $R_T = 5.5\,\mathrm{m}$.
- From figure 6.13, calculated tip-width fraction of $w \approx 0.6$
- $R_H \approx \frac{1}{10} R_T$, from inspection of same figure.
- The tip radius was set to $R_{tip} = \frac{3}{4} R_T$, as before.
- L, the length of the turbine volume. For practical concerns – that the turbine volume would capture some nodes, whilst allowing a reasonably-large minimum size of element – this was set to $L = 5\,\mathrm{m}$. This is approximately 5 times length of Seaflow ($\approx 1\,\mathrm{m}$).

Parameters from performance graphs: optimum power efficiency, cut-in/cut-out speeds, etc.

Unlike wind turbines, detailed information on performance characteristics for axial flow marine turbines is not readily and universally available, however we can draw on data from published reports, as well as making decisions to based upon what is known about the nature of marine current turbines.

- Determining the flow speed at peak efficiency, $u_{0,opt}$. Data from Marine Current Turbines Ltd. [48] states that Seaflow generates a maximum of 300 kW of power at in a tidal flow of 2.7 m/s; this speed will be $u_{0,opt}$.
- The optimum power efficiency, $c_{P,opt}$. In the DTI report on Seaflow, this is $c_{P,opt} = 0.40$. To check this for the specified $u_{0,opt}$ and maximum power extracted of 300 kW, we will revisit the equation for power extracted via axial flow turbines

$$PW_{ex} = c_P \frac{1}{2} \rho \pi R_T^2 u_0^3 \tag{6.14}$$

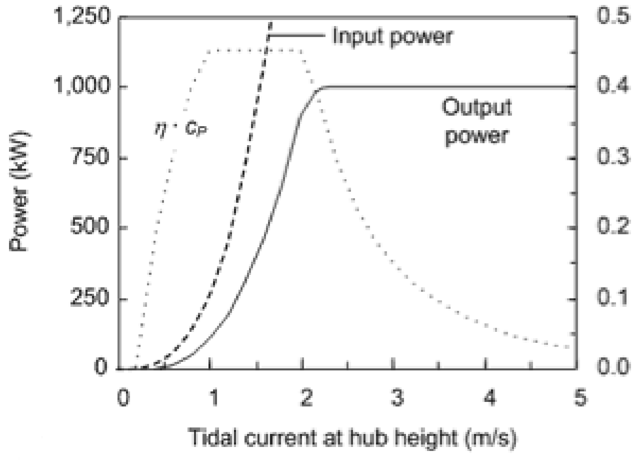


Figure 6.15: Typical graph for a tidal current turbine showing flow speed versus PW_{ex} for a marine turbine (courtesy of Marine Current Turbines Ltd.)

If we plug in the optimum values for PW_{ex}, R_T, u_0, and assume the density of the seawater is $1027\,\mathrm{kg\,m^{-3}}$ (this shall be justified later), then $c_P = 0.31$, which is 20% less efficient than suggested; this is the value we will use.

- $\omega_{T,opt}$, the optimum angular velocity. According to Fraenkel [30], blade-tip speed needs to be limited to prevent cavitation occuring on the surface of the turbine blades: he suggests a limit of $12-15\,\mathrm{m/s}$. If we take $15\,\mathrm{m/s}$ as our maximum, then with $R_T = 5.5\,\mathrm{m}$ that gives a maximum angular velocity of $\omega_{T,opt} = 2.73\,\mathrm{rad/s}$

  This also gives a tip-speed ratio of $\lambda = 2.02$: much lower than would be expected for a horizontal axis wind turbine.

- The freestream flow speed at cutout, $u_{0,cutout}$. Since Seaflow does not actually cut out at higher flow rates, we set this to an impossibly high value, $u_{0,cutout} = 10\,\mathrm{m/s}$.

- $u_{cutin} = 0$, as Seaflow does not actually have a 'cut in' mechanism such as wind turbines have.

- From above, and assuming a hard-limited 300 kW maximum power output, $c_{P,cutout} = 0.0061$ (even though the cutout speed is never attained, this is needed for the equations).

Fluid dynamic properties

As with the Vestas V52 model, these parameters were refined through experimental runs, initially lifted directly from that model. The flow factor, interestingly, had to be raised to $f = 2.5$, to give $c_{P,opt} = 0.31$ at $u_0 = 2.70\,\mathrm{m/s}$. Q remained unaltered at $Q = 0.65$.

Turbulence

In the absence of any data to the contrary, the turbulence setting was taken from that for the Vestas V52 model, and so $Ti_{opt} = 0.15$ at $u_0 = u_{0,opt}$.

Relaxation times

Given that the difference in Reynolds numbers in flow over wind turbines and marine current turbines indicates a more turbulent environment for the latter, the relaxation times were longer than those for the Vestas to ensure a degree of stability in operation. Thus, $t_{relax,vel} = 60\,\mathrm{s}$, and $t_{relax,\alpha} = 20\,\mathrm{s}$.

6.4.3. Seaflow water channel with inlet turbulence

Overview

In a similar vein to the wind turbine simulations, a rectilinear channel was used as the simulation doman. The aim was to create a rudimentary tidal channel containing seawater (figure 6.17), with a Dirichlet inlet condition on left similar to before and an open boundary to the right. Again, the four remaining walls are frictionless.

Generating inlet boundary conditions

Turbulence Here, the same mechanism for generating turbulence was used as in section 6.3.4. Again, the turbulence intensity was set with $Ti_x = 0.15$.

Name	**Symbol**	**Value**
Radius of turbine	R_T	5.5 m
Hub radius	R_H	$\frac{1}{10}R_T$
Hub local solidity	B_H	1.0
Tip width fraction	w	0.6
Tip radius	R_{tip}	$\frac{3}{4}R_T$
Turbine volume length	L	5 m
Net solidity	B	0.1
Optimum net effective solidity	B'_{opt}	0.25
Flow factor	f	2.5
Momentum extraction efficiency	Q	0.65
Optimum freestream speed	$u_{0,opt}$	2.7 m/s
Optimum blade angular velocity	$\omega_{T,opt}$	2.73 rad/s
Optimum power coefficient	$c_{p,opt}$	0.31
Freestream wind speed at turbine cutout	$u_{0,cutout}$	10 m/s
Power coefficient just below cut-out windspeed	$c_{P,cutout}$	0.0061
$max(u_i)$ at cut-in and cut-out	u_{cutin}, u_{cutout}	0 m/s, 9.0 m/s
Tip-generated turbulence at $u_0 = u_{0,opt}$	Ti_{opt}	0.15
Turbine angular velocity relaxation time	$t_{relax,vel}$	20 s
Blade angle factor relaxation time	$t_{relax,\alpha}$	10 s

Table 6.7: List of physical properties of model of Seaflow

Ramping boundary values For the wind tunnel simulations, the Dirichlet inlet condition was simply set as $u = u_0$ and the simulation allowed to progress. However, a similar approach could not be adopted for the marine simulations,. As $u \to u_{0,opt}$, the adaptive algorithm would 'blow up', filling the unstructured mesh with tiny elements and thereby lengthening run time considerably. Setting $\underline{u}(\underline{x}) = \underline{u}_0$ for every point in the mesh also produced this result, so an alternative approach was needed.

It was decided that the best solution to this problem was 'ramped' boundary conditions. This involves linearly increasing u at the boundary from 0 to u_0, as can be seen in figure 6.16.

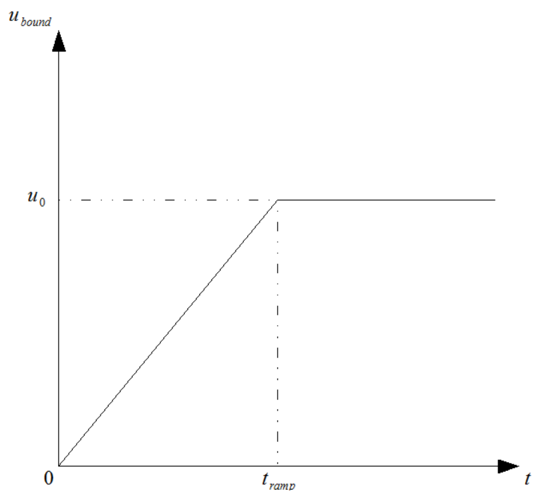


Figure 6.16: Ramped boundary condition for u

With u_0 and t_{ramp} the time to ramp up the boundary specified, a simple algorithm was implemented to set u at the boundary at time t:

$$u_{bound}(t) = \begin{cases} \left(\frac{t}{t_{ramp}}\right) u_0 & t < t_{ramp} \\ u_0 & t \geq t_{ramp} \end{cases} \tag{6.15}$$

Through experiment it was found that for $u_{0,opt} = 2.7\,\mathrm{m/s}$, $t_{ramp,opt} = 3000\,s$ gave good results. This gives

$$\dot{u}_{bound} = 0.0015\,\mathrm{ms}^{-2} \tag{6.16}$$

At first glance, it might seem that prudence would suggest optimum results would be acheived by maintaining this gradient. However, as the error tolerances become smaller with u_0, so would $\dot{u}_{bound}$ – this meant keeping t_{ramp} constant with decreasing u_0.

Water channel dimensions

These had to be close to what would be expected in realistic conditions. Anecdotal evidence from the length of ship propeller wakes, suggested that wake recovery in axial flow marine turbines may be considerably longer than 20 rotor diameters – so as a precaution the tunnel length was set to considerable longer than would have adequate for a wind turbine of similar size. As to width, this was set to what could be called a plausible minimum for a tidal strait, certainly enough wide enough that shore effects could be ignored. Lastly, the depth of the channel was set to that of the Seaflow test site near Foreland Point in North Dorset, England as mentioned in the DTI report.

This gives the dimensions of the channel as

$$V_{tunnel} = L_x \times L_y \times L_z = 800\,\mathrm{m} \times 200\,\mathrm{m} \times 40\,\mathrm{m} \tag{6.17}$$

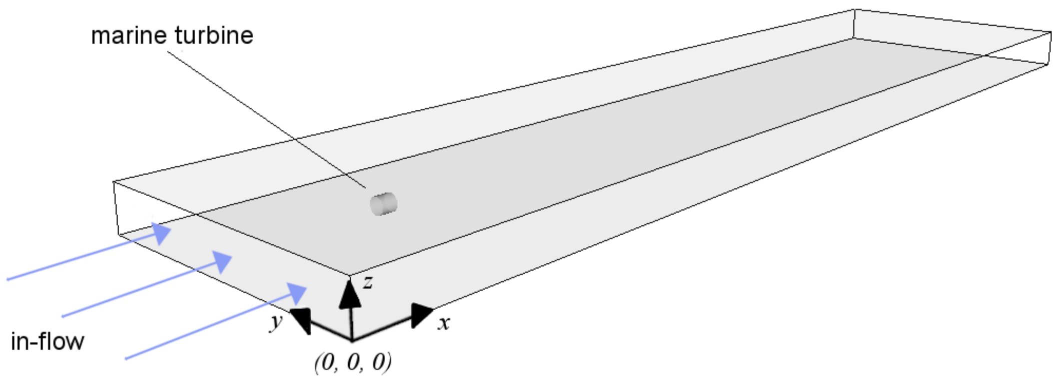


Figure 6.17: The Seaflow simulation domain

Turbine positioning

Again, the turbine was positioned in the centre of the incoming flow towards the inlet with its axis parallel to it. It was set at half-depth, giving its position as

$$\underline{x}_T = \begin{bmatrix} 100\,\mathrm{m} \\ 100\,\mathrm{m} \\ 20\,\mathrm{m} \end{bmatrix} \tag{6.18}$$

Fluid properties

The seawater was assumed incompressible, and with a temperature of 15^o C under normal conditions this gave the fluid properties as presented in table 6.8.

Name	**Symbol**	**Value**
Density	ρ	$1027\,\mathrm{kg\,m^{-3}}$
Dynamic viscosity	μ	$1.50 \times 10^{-3}\mathrm{Pa\,s}$

Table 6.8: Fluid properties of simulation seawater

Varying inlet conditions and adaptive mesh configuration

The final (or *target)* inlet water speed was varied; these water speeds are shown in table 6.9.

u_0 **(m/s)**
0.5
1.0
1.5
2.0
2.5
2.7
3.0
4.0
5.0

Table 6.9: List of inlet water speeds

The parameters below are invariant with u_0, and so did not change between simulation runs.

Name	Symbol	Value
minimum x length of element	$x_{ele,min}$	1 m
minimum y length of element	$y_{ele,min}$	1 m
minimum z length of element	$z_{ele,min}$	1 m
maximum x length of element	$x_{ele,max}$	20.0 m
maximum y length of element	$y_{ele,max}$	20.0 m
maximum z length of element	$z_{ele,max}$	20.0 m
initial divisions along x-axis	N_x	100
initial divisions along y-axis	N_y	20
initial divisions along z-axis	N_z	10

Table 6.10: List of invariant settings for marine adaptive mesh

As before, the error tolerances were adjusted for each value of u_0 to give an accurate representation of the flow without adversely affecting simulation speed via oversensitive mesh adaption. These can be seen in table 6.11 .

6.4.4. Seaflow with bottom drag

Overview

This is a modification of the previous experiment in section 6.4.3 to simulate a more realistic environment, in so much that bottom drag is introduced, with consequent velocity profile. Fluidity does not currently have 3D free surface modelling implemented in parallel, so this was excluded. Instead, the rigid lid was extended upwards to 80 m to ameliorate the effects of the artificially-restricted channel surface, which would accelerate the flow around the turbine. An inlet velocity with a peak of $u_0 = 4\,\text{m/s}$ was chosen.

Generating inlet boundary conditions

The Dirichlet inlet conditions were ramped as before, but with a vertical scaling factor in addition to the time scaling factor. One approach may be to

$u_0\,(\mathbf{m/s})$	Δe_u	Δe_v	Δe_w	Δt (s)	$LTIME$ (s)
0.5	0.01	0.007	0.007	5.4	6000
1.0	0.027	0.0065	0.0065	2.7	6000
1.5	0.03	0.007	0.007	1.8	6000
2.0	0.035	0.007	0.007	1.35	6000
2.5	0.04	0.007	0.007	1.08	6000
2.7	0.045	0.007	0.007	1.0	6000
3.0	0.05	0.0085	0.0085	0.9	5500
4.0	0.07	0.0085	0.0085	0.675	5000
5.0	0.08	0.015	0.0015	0.54	4500

Table 6.11: Inlet water speed, error tolerances, time-step size and simulation duration

model the sea bottom with a no-slip condition, but this would lead to a steep velocity gradient which, in turn, would lead to Fluidity's adaptive algorithm adding excessive resolution to the mesh at the bottom. Thus a slip condition is introduced with friction specified by the absorption coefficient at each node, and the appropriate velocity profile at the inlet specified.

To find this, according to Drago [23], the velocity profile can be stated as a function of height from the sea bed

$$u_0(z) = \frac{u_\tau}{K} \ln\left(\frac{z}{z_R}\right) \tag{6.19}$$

Where $K = 0.41$ is the universal von Karman constant, u_τ is the frictional velocity and z_R is the seabed roughness. We can take $z_R = 0.4\,\text{m}$, a reasonable estimate for a tidal strait [77]. However, since our boundary conditions are time-dependent due to the ramp time in (6.15), the boundary condition becomes

$$u_{bound}(z,t) = \begin{cases} \left(\frac{t}{t_{ramp}}\right) u_0(z) & t < t_{ramp} \\ u_0(z) & t \geq t_{ramp} \end{cases} \tag{6.20}$$

This leaves the definition of the frictional velocity; if we say that z_S is the height of the surface then at time t

$$u_S(t) = \frac{u_\tau(t)}{K} \ln\left(\frac{z_S}{z_R}\right) \tag{6.21}$$

This means that if we can define $u_S(t) = u_0(z_S, t)$, then

$$u_\tau(t) = \frac{K u_0(z_S, t)}{\ln\left(\frac{z_S}{z_R}\right)} \tag{6.22}$$

And so the time-dependent inlet vertical velocity profile is specified.

Turbulence at the inlet

This was generated in an identical manner to before, with the turbulent intensities remaining the same. However, since u_0 has becomes a function of height, this changes the force acting on the fluid due to turbulence in equations (6.8), (6.9) and (6.8). These now become

$$\Delta u_{inlet,i} = Ti_x\, G_x(t)\, u_0(z_i, t) \tag{6.23}$$

$$\Delta v_{inlet,i} = Ti_y\, G_y(t)\, u_0(z_i, t) \tag{6.24}$$

$$\Delta w_{inlet,i} = Ti_z\, G_z(t)\, u_0(z_i, t) \tag{6.25}$$

Furthermore Δt_{inlet} in (6.11) must be redefined as

$$\Delta t_{inlet} = \frac{(\overline{\Delta x_{ele}})}{u_0(z_i, t)} \tag{6.26}$$

Thus the force-per-unit-volume can be defined for the bottom drag velocity profile, as per equation (6.12).

Bottom drag

The drag coefficient is needed to effect bottom roughness, and must match the inlet velocity profile given. This is specified within Fluidity as

$$C_D = \left[\frac{K}{ln\left(\frac{z_R + z_N}{z_R}\right)} \right]^2 \tag{6.27}$$

Where z_N is the distance from the seabed to the closest element nodes just above it; in the marine turbine simulations the minimum element size was 2.5 m and so a good estimate of this would be $z_N = 2.5\,\mathrm{m}$. This gives a drag co-efficient of $C_D \approx 0.043$.

Finally, if we assume a linear bottom friction law [33], we can write the drag force-per-unit-volume vector as

$$\underline{F}_{drag} = -C_D \left(\frac{\rho}{t_{relax}} \right) \begin{bmatrix} u \\ v \\ 0 \end{bmatrix} \tag{6.28}$$

Where t_{relax} is the time over which the drag is applied. This can be put into Fluidity as momentum absorption terms, which means t_{relax} becomes an automatically calculated relaxation period.

Adaptive mesh settings

The bulk of the settings were carried over from the previous rigid lid case for $u = 4.0\,\mathrm{m/s}$ in section 6.4.3, however several of the adaptive settings had to be modified to cope with the altered flow conditions due to the bottom drag: these alterations are listed below in table 6.12.

The most significant changes are the increased minimum element dimensions, and the z-component of the error tolerance in the velocity gradient. These were made to ensure that whilst the simulation would not concentrate an undue number of nodes near the sea floor, it would maintain enough to resolve the non-linear velocity gradient accurately.

Name	**Symbol**	**Value**
minimum x length of element	$x_{ele,min}$	2.0 m
minimum y length of element	$y_{ele,min}$	2.5 m
minimum z length of element	$z_{ele,min}$	2.5 m
maximum x length of element	$x_{ele,max}$	100.0 m
maximum y length of element	$y_{ele,max}$	50.0 m
maximum z length of element	$z_{ele,max}$	30.0 m
initial divisions along x-axis	N_x	100
initial divisions along y-axis	N_y	20
initial divisions along z-axis	N_z	20
error metric in x-direction	Δe_u	0.08
error metric in y-direction	Δe_u	0.08
error metric in z-direction	Δe_u	0.09
simulation duration (s)	$LTIME$	6000

Table 6.12: List of invariant settings for bottom drag simulation adaptive mesh

Chapter 7

Results and analysis

7.1. Data formats

7.1.1. Unstructured mesh files

Fluidity makes periodic dumps of the unstructured finite element mesh representing the flow, after an interval specified by the $TIMDUM$ parameter. This contains information on the fluid field at each mesh node; the fluid properties are listed in table 7.1

Name
Node coordinate $\underline{x}$
Velocity $\underline{u}$
Pressure P
Turbine presence
CPU ID

Table 7.1: List of mesh properties

These dump files are in written in Fluidity's propietary format, on a per-processor basis. Fortunately, Imperial College provides tools to convert and merge these into a VTU file, a data format for unstructured meshes and compatible with the Visualisation Toolkit (VTK) [4]. The VTU format allows a variety of VTK-aware software to load these files for data analysis and visualisation purposes.

7.1.2. Time-dependent turbine performance data

In addition to the fluid state, the turbine module would write to file various turbine performance-related statistics for each turbine at each simulation time-step, ie. every Δt seconds. Separating out data for each turbine would allow the performance of each individual turbine to be evaluated in wind farm or marine farm simulations. The data was in the common spreadsheet CSV format, essentially a text file with comma-separated values; this would allow convenient loading and graphing in spreadsheets applications such as Microsoft's Excel or OpenOffice's Calc program. The list of variables is shown in table 7.2.

Name
t, current simulation time
k, turbine number
N_k, number of nodes in turbine k
$\overline{u}_k$, relaxed form
$\overline{u}_{H,k}$, relaxed form
$max(u)_k$, maximum value of u, relaxed form
α_k
$\overline{\omega}_{FL,k}$
$\overline{\omega}_{T,k}$
$\overline{\omega}_{T,max,k}$
B'_k
$\overline{\rho}$ mean fluid density within turbine (future compressible use)
$PW_{ex,k}$
$\overline{u}_k$, instantaneous form
$\max(u)_k$, instantaneous form

Table 7.2: List of turbine performance variables

7.2. Analysis software

Bespoke Numerical Python (NumPy)

One of the main problems with unstructured mesh data, is manipulating into a regular form whereupon analysis can be performed. This is compounded when the meshes are time-dependent, since comparison between two time-steps necessarily involves interpolating both individually to identical meshes: usually these are regular grids (more on these techniques in 7.3.1).

Python is an easy-to-use scripting language with a freely available interpreter. It has support for VTK , and extensive mathematical tools provided through its NumPy library. This made it an ideal choice as a processing tool for the fluid mesh. Specifically, NumPy programs written formed the basis of all the grid-interpolation and complex time-dependent analysis in this section.

Matlab

Since Matlab's duties for data interpolation had been taken over by the Python programs, it was relegated to doing batch processing of:

1. Contour plots for the Python programs' output
2. Statistical and time-based plots from global variables

OpenOffice Calc

While this spreadsheet program was not in itself used to generate plots, it was invaluable as a tool to quickly inspect turbine performance variables, ie. 'test' plots. The CSV file is a universal format generated by the model is readily understood by Calc and a variety spreadsheet packages.

Paraview

Paraview is free visualisation software from Kitware, which allows slices through 3D sets of data to be generated in real time, and viewed in a variety of ways. Paraview will be used to demonstrate instantaneous properties of the flow in the forthcoming sections.

7.3. Techniques for analysis

7.3.1. Preface

Mesh interpolation

As mentioned before, Fluidity stores fluid field information in an unstructured finite element mesh, and this presents several problems, which require mesh interpolation to answer them. For fluid properties that required full-volume analysis, a 3D Cartesian grid would be specified as

$$D_x \times D_y \times D_z \tag{7.1}$$

Where D_x, D_y and D_z would be the number of subdivisions along the x, y, and z axes respectively.

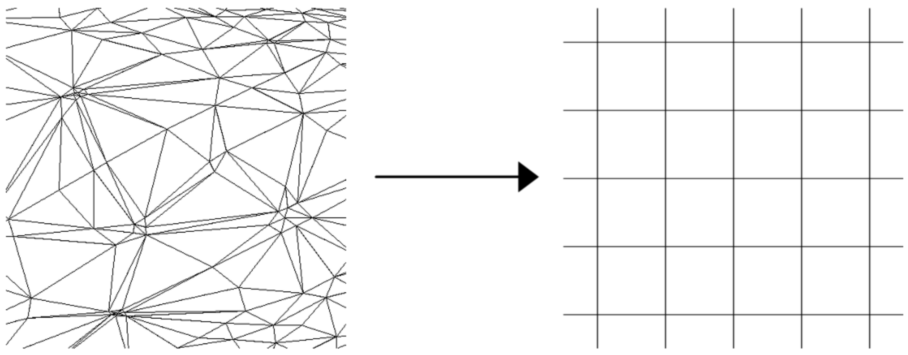


Figure 7.1: Interpolation from a 2D irregular mesh to a regular 2D Cartesian grid

The VTK extensions for Python allow for rapid linear interpolation of VTU meshes within Python programs, and so regular gridded datasets could be produced very quickly. Using Python and VTK to interpolate the data instead of outputting sparse data to Matlab had two benefits:

1. **Speed**. While more complex to use, Python is several orders of magnitude faster than Matlab at data interpolation.
2. **Accuracy**. Python used with VTK allows linear interpolation within elements (see figure 7.2), rather than sparse data interpolation in Matlab, which does not support VTU files.

Essentially, once $D_x \times D_y \times D_z$ was specified, this would produced a file of interpolated values for a particular VTU file.

Standard deviations and means

Since all the flows dealt with are turbulent and thus transient, it is more meaningful to look at statistical properties of the flow, rather than those at any one point in time. We can do this once a series gridded data files had been produced; statistics about the data can be produced using custom-written Python programs.

Suppose we have gridded data files $\{G_{t_n}, G_{t_{n+1}}, ..., G_{t_m}\}$ where t_n is the n^{th} time-step, and so for t_m. If there are M time-steps, then we can write $n = m - (M - 1)$. If we think of G containing a set of properties for each of N points, denoted $\underline{q}_i$ for point i where $1 \leq i \leq N$, then we can write the temporal average of an individual property as

$$\overline{q}_i = \frac{1}{M} \sum_{j=n}^{j=m} q_{i,j} \tag{7.2}$$

For each point i, where $q_{i,j}$ is property q_i at the j^{th} time-step. If we then define the mean of the square as

$$\overline{q_i^2} = \frac{1}{M} \sum_{j=n}^{j=m} q_{i,j}^2 \tag{7.3}$$

Then we can define $\sigma_{q,i}$ the standard deviation of q_i as

$$\sigma_{q,i} = \sqrt{(\overline{q}_i^2 - \overline{q_i}^2)} \tag{7.4}$$

Typically, we only look at the flow when it is fully developed, so this would be when the wake behind the turbine has become relatively stable. Moreover, M has to be quite large for the calculated values to be reliable.

7.3.2. Velocity deficit

This is essentially a measure of flow recovery downstream of the turbine in the wake. If we have our regular gridded data of N points, we can define this for point i as

$$d_i = 1 - \left(\frac{\overline{u}_i}{u_0}\right) \tag{7.5}$$

Where u_0 is the final value of u at the inlet boundary.

7.3.3. Turbulence intensity

As turbulence intensity is essentially the normalised standard deviation of the speed of the flow, this is written as

$$Ti_i = \frac{\sigma_{u,i}}{u_0} \tag{7.6}$$

Which is the turbulence intensity at the i^{th} point.

7.3.4. Circulation

Circulation, as used here, can be thought of as a measure of the rotation of the wake. This is defined as the path integral over a closed curve C

$$\Gamma = \oint_C \underline{u} \cdot dS \tag{7.7}$$

Where dS is the unit vector pointing along the loop. For the purposes of measuring circulation downstream of turbine, the loop shall be considered as a circle centered on the turbine's axis, positioned at $\underline{x}_C = (x_C, y_T, z_T)$ with a radius of R_C – see figure 7.3.

Numerically, this breaks down into the N points around the circumference, and can be represented as

$$\Gamma = \frac{2\pi}{N} \sum_{i=0}^{N-1} v_i \left(\frac{z_i - z_T}{R_C}\right) - w_i \left(\frac{y_i - y_T}{R_C}\right) \tag{7.8}$$

where $\underline{x}_i$ and $\underline{u}_i$ represent the position of points and the fluid velocity at points on the loop. $\underline{u}_i$ is found via mesh interpolation, while the point position is specified thus

$$x_i = x_C \tag{7.9}$$

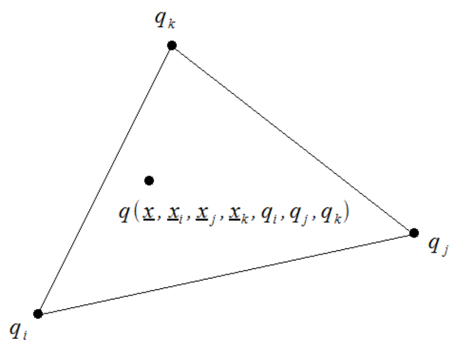

q_k
$q(\underline{x}, \underline{x}_i, \underline{x}_j, \underline{x}_k, q_i, q_j, q_k)$
q_j
q_i

Figure 7.2: Linear interpolation of property q in a 2D triangular element constructed from elements i, j and k.

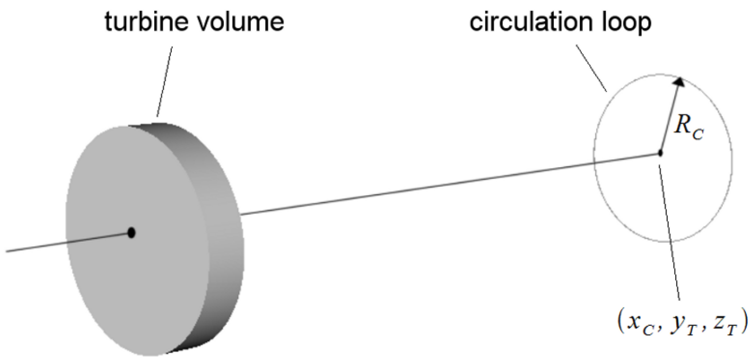


Figure 7.3: Circulation loop downstream of turbine

$$y_i = sin(\theta_i) + y_T \tag{7.10}$$

$$z_i = cos(\theta_i) + z_T \tag{7.11}$$

where $\theta_i = 2\pi\left(\frac{i}{N}\right)$.

7.3.5. Turbine performance

There are four variables that are of interest for each turbine

- Blade parameter α
- Turbine angular velocity ω_T
- Extracted power PW_{ex}
- Effective solidity B'

As the flow in all simulation cases is turbulent, these variables will almost always be unsteady, since they are a function of the fluid velocity field. For power efficiency analysis and the like, this means long-term averages must be used.

If we consider time-step n as when the wake has fully developed and maximum power output has been achieved, and m the final time-step, then for turbine property q_i:

$$\overline{q}_i = \frac{1}{M}\sum_{j=n}^{j=m} q_{i,j} \tag{7.12}$$

where $M = (m+1) - n$ is as before.

Moreover, for each different inlet value of u_0 the power efficiency shall be calculated as

$$c_P = \frac{\frac{1}{2}\rho\pi R_T^2 u_0^3}{\overline{PW}_{ex}} \tag{7.13}$$

7.4. Results for Vestas V52 wind turbine

7.4.1. Vestas V52 with inlet turbulence at $u_0 = 12\,\mathrm{m/s}$

For any time-averaged values, the interval over which they are averaged must be chosen with care. The simulation has a spin-up period in which the turbine reaches its relatively stable operating conditions, and the wake has fully developed. This was estimated through inspection of the turbine variables over time, at which point parameters such as PW_{ex} have largely stablised, and by visual inspection of planar velocity slices within applications such as Paraview.

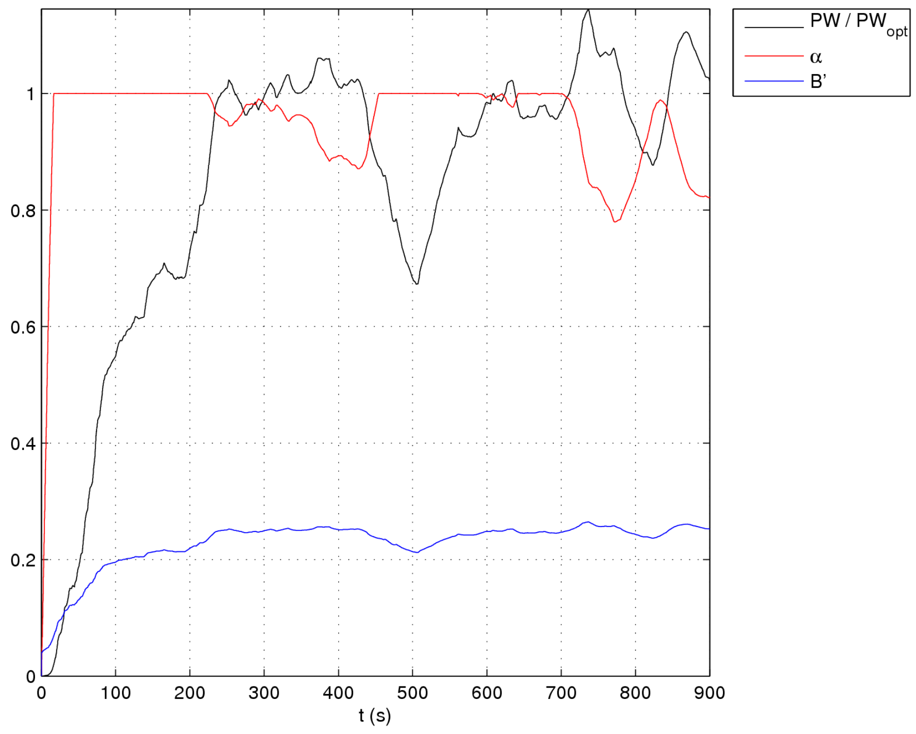


Figure 7.4: Evolution of α, B' and normalised PW_{ex} over time

Figure 7.4 shows a significant drop in power at $t = 500\,s$ after the initial turbine spin-up, which then stablises at approximately $PW_{ex} = PW_{ex,opt}$. This was perhaps due to a recirculation bubble – a vortex – that occured at approximately 2-3 turbine diameters downwind (denoted 'D') of the turbine, see fig. 7.5. This was later almost completely dispersed, and after 13-14 minutes was just over half of what it was, as can been in fig. 7.6.

The counteracting action of α can be seen; of particular note is not only

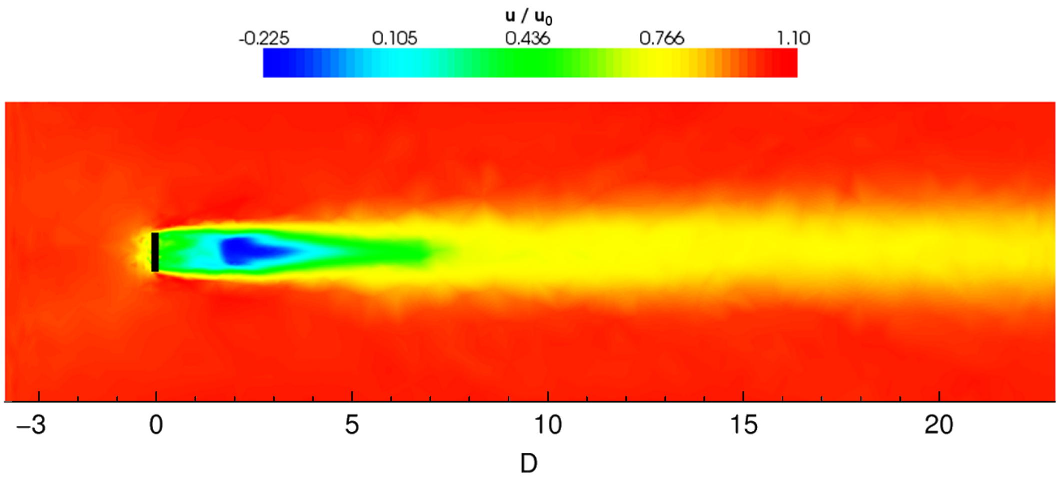


Figure 7.5: Planar slice showing u/u_0 at $t = 500\,\text{s}$. The turbine volume is represented by the black rectangle. The strongest recirculation is 22%, 2 to 3 diameters downstream.

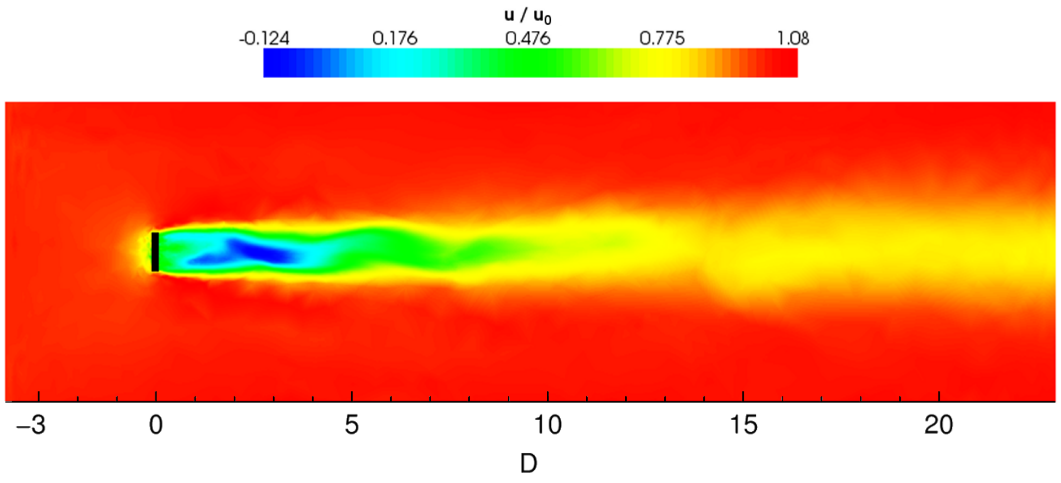


Figure 7.6: Planar slice showing u/u_0 at $t = 815\,\text{s}$. The recirculation has now signficantly weakened, at 12%. Note the wake is more turbulent.

that α decrease when PW_{ex} exceeds $PW_{ex,opt}$, but also when $\frac{d(PW_{ex})}{dt}$ becomes too large: this is a consequence of equation (4.63) specifying $(\dot{\omega}_T)_{max}$ in section 4.3.5.

It can also be seen that there is a corresponding drop in effective solidity with power; conversely, this increases the flow through the turbine, which leads to a power increase. From figure 7.4 we can see that the period after 600 s is dynamically stable, so all time-averaged values were calculated over the period $600\,\mathrm{s} < t < 900\,\mathrm{s}$; an interval of 5 minutes. We shall denote this interval t_{av}.

Horizontal slices were then taken through domain from the inlet to the outlet, over the period t_{av}, at $z = 200\,\mathrm{m}$ so they would pass through the turbine. A time-averaged velocity deficit was calculated; fig. 7.7 shows the contour plot. From this, we can see the $u = 0$ contours that sit either side of the turbine, indicating that the external layers of fluid are moving *faster* than u_0; it also shows a stagnant region about $2 - 3D$ (diameters) downwind of the turbine. There is a non-linear wake recovery (ie. $\frac{du}{dx}$ is not constant), as seen through the non-linear separation of the contours in the wake, with near total wake recovery ($u > 0.80\,u_0$) occuring at approximately $19D$ downstream. Furthermore, the wake expansion is clearly limited at $x \approx 8D$; this is suspected to be due to the restrictive effect of the boundary walls.

Another view of this would the normalised cross-sectional velocity profile, as seen in fig. 7.8, which gives greater detail in radial directions. This profile has been averaged over both t_{av} as well as orbitally – $-R_T < r < 0$ represents the average over an arc a angle of π where $y < 0$, whereas $0 > r > R_T$ is the average for a similar arc where $y > 0$.

In this figure we can see that upstream at $x' = -5D$ (where $x' = x - x_T$) there is little disturbance to the flow, however at $x' = -1D$, approximately 50 metres upstream, the pressure wave created by the turbine is already slowing the fluid. At $x' = 0$, immediately downwind of the turbine, the peak moment extraction occurs at $r \approx \pm\frac{3}{4}R_T$; at larger r, the tip-loss effect is clearly evident. For $R_T \leq |r| \leq 2R_T$ at $1D \leq x' \leq 3D$ acceleration occurs just past the tips of the turbine, which matches the bulge in the vertical profiles for flat plate laminar flow in figure 2.4 (b). At greater values of x' this effect diminshes, and the profile smooths out into a 'U' then shallow 'V' shape, showing the minor recirculation ($\approx$ 10%) occuring at $x' = 2 - 3D$; by the time the wake is more than $15D$ long, it has returned to almost 80% of its freestream value.

The wake turbulence structure in figure 7.9, clearly shows an outer layer of high turbulence which decays sharply along two parallel ridges, and a less

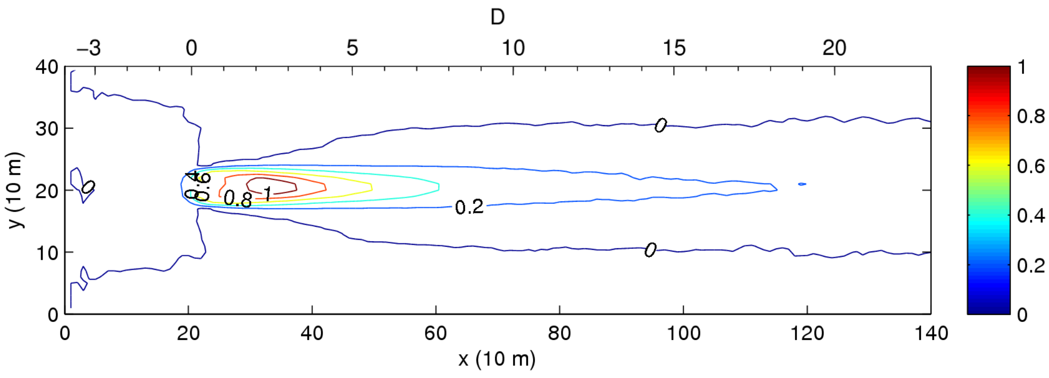


Figure 7.7: Planar contour plot of time-averaged velocity deficit. The contour line ‘0’ denotes full wake recovery.

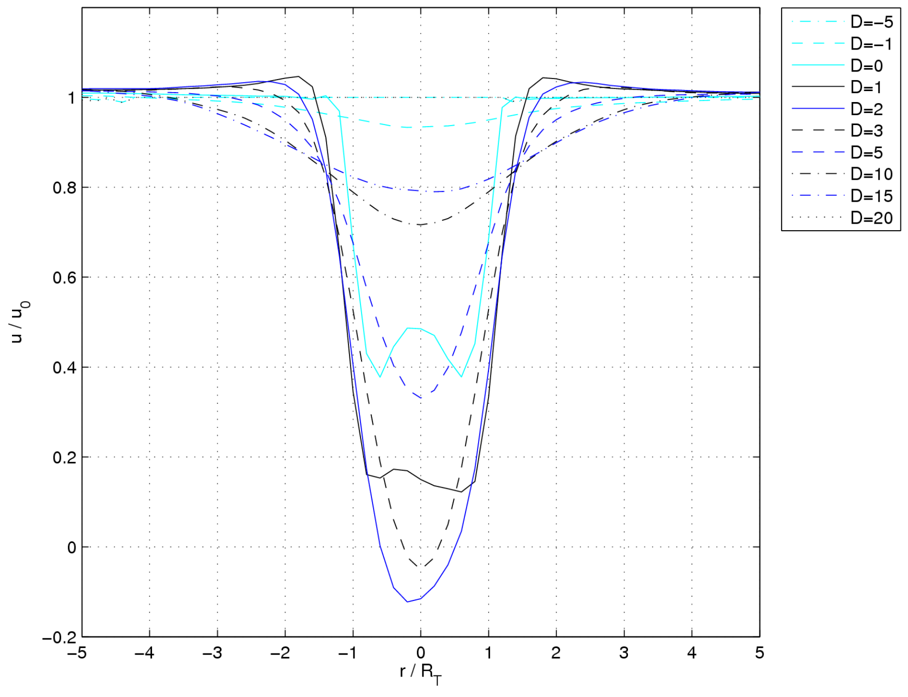


Figure 7.8: Spatially and time-averaged u profiles for varying distances upstream and downstream. The slight asymmetry at D=1 suggests t_{av} may need to be longer to give a symmetric profile.

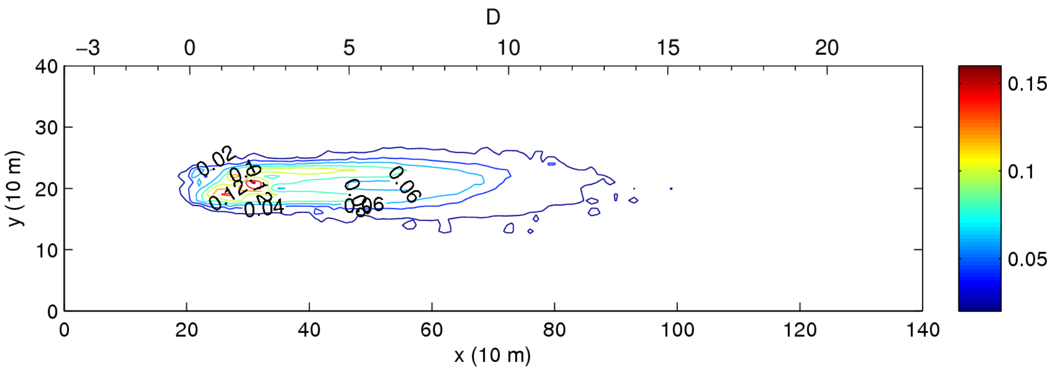


Figure 7.9: Planar contour plot of turbulence intensity

turbulent zone centred around $r = 0$ which terminates in a highly turbulent patch at about $x' = 2D$. This is approximately where the wake recirculation is. It should be noted, however, that while the maximum turbulence intensity Ti, is almost the same as that specified for the turbine, with $Ti_{opt} = 0.16$, this does not include sub-grid turbulence. The sub-grid turbulence of the LES model is being generated, but is not available as output within Fluidity, although its effect can still be seen within the flow structure.

7.4.2. Vestas V52 with inlet turbulence: overview of performance

As u_0 was varied for each simulation run, t_{av} was respecified as

$$t_{av} = \left[\frac{LTIME}{2}, LTIME\right] \tag{7.14}$$

For inlet conditions other than $u_0 = \{7.5\,\mathrm{m/s}, 10\,\mathrm{m/s}, 12\,\mathrm{m/s}\}$. This was sufficient criteria to ensure any mean values taken over t_{av} were representative.

The performance plot in figure 7.10 shows a sharp jump to nearly the quoted maximum efficiency of $c_P = 0.4$, with a near quadratic drop-off for higher wind speeds, until a complete cut-out at $u_0 = 25\,\mathrm{m/s}$. The drop-off is due to the hard-limiting of the power output of the turbine to 850 kW through the blade factor α. Figure 7.11 shows the relationship between α and u_0 through this power throttling: time-averaged power output remains close to $PW_{ex,opt}$ even at higher windspeeds.

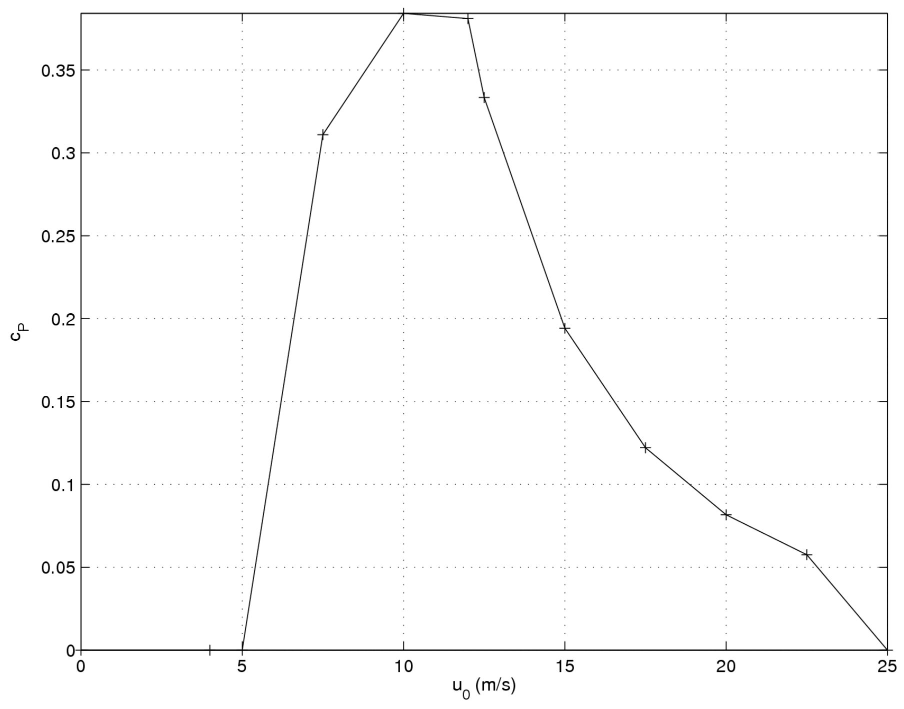


Figure 7.10: Performance curve for V52 with inlet turbulence, peaking at $u_0 = 12\,\mathrm{m/s}$. Official figures give $c_P = 0.4$ at $u_0 = 11.78\,\mathrm{m/s}$.

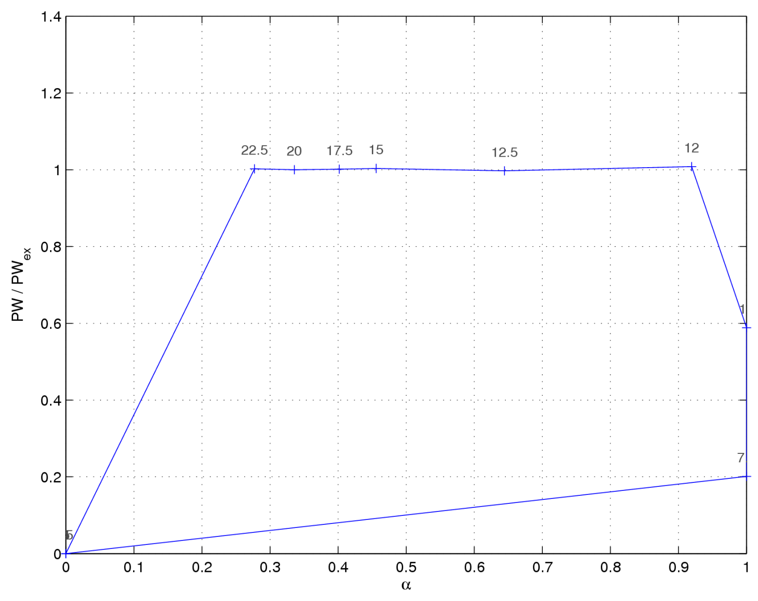


Figure 7.11: Plot of blade factor α versus PW_{ex}/PW_{opt}. The numbers next to the points represent u_0 in m/s.

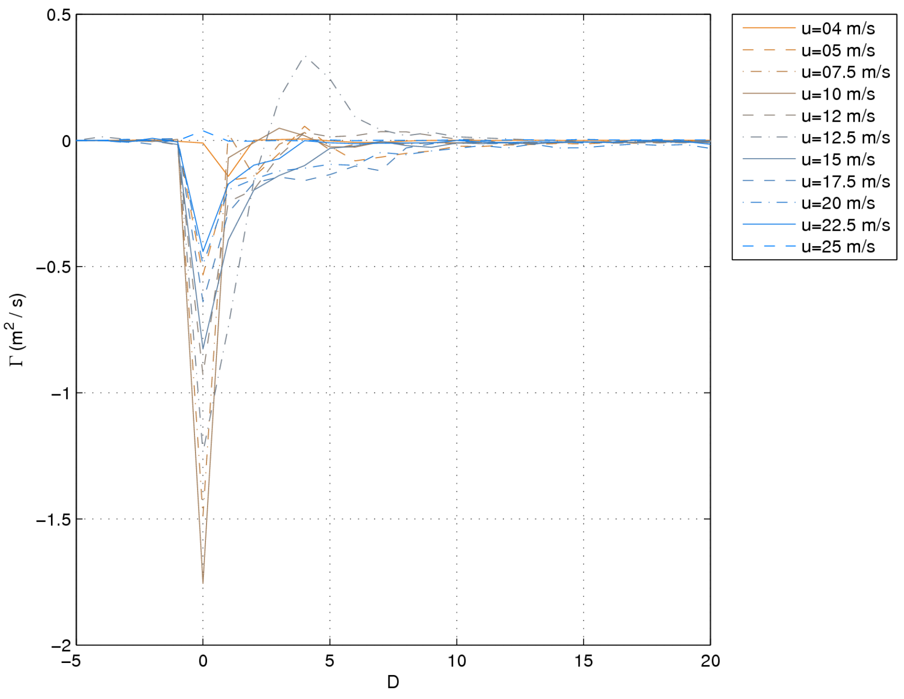


Figure 7.12: Plot of D versus circulation Γ for varying u_0

Looking at the circulation in figure 7.12, we can see that for all wind speeds Γ peaks immediately or almost immediately down wind of the turbine, and decreases almost quadratically with distance downstream. We can also see that Γ decreases with increasing u_0, after peaking at $u_0 = 10 - 12\,\mathrm{m/s}$. Conversely, circulation is very small at $u_0 = 4\,\mathrm{m/s}$ and $u_0 = 25\,\mathrm{m/s}$.

7.4.3. Vestas V52 with inlet turbulence: comparisions between wind speeds

Inlet windspeed of $u_0 = 7.5\,\mathrm{m/s}$

From the fig. 7.13, we can see that the turbine power output is about $\frac{1}{5}PW_{ex,opt}$ whilst the dynamic solidity B' quickly progresses to $B' \approx 0.20$. We can also see that $\alpha = 1$ for the most part, since no limiting is required at such low wind speeds. t_{av} was taken as between 600 and 900 s.

The velocity deficit contour plot in figure 7.14 shows that a similar amount of recirculation occurs to $u_0 = 12\,\mathrm{m/s}$, with $\max(d) \approx 1.25$, and that the wake shows a similar length for recovery, which can be more clearly seen in the

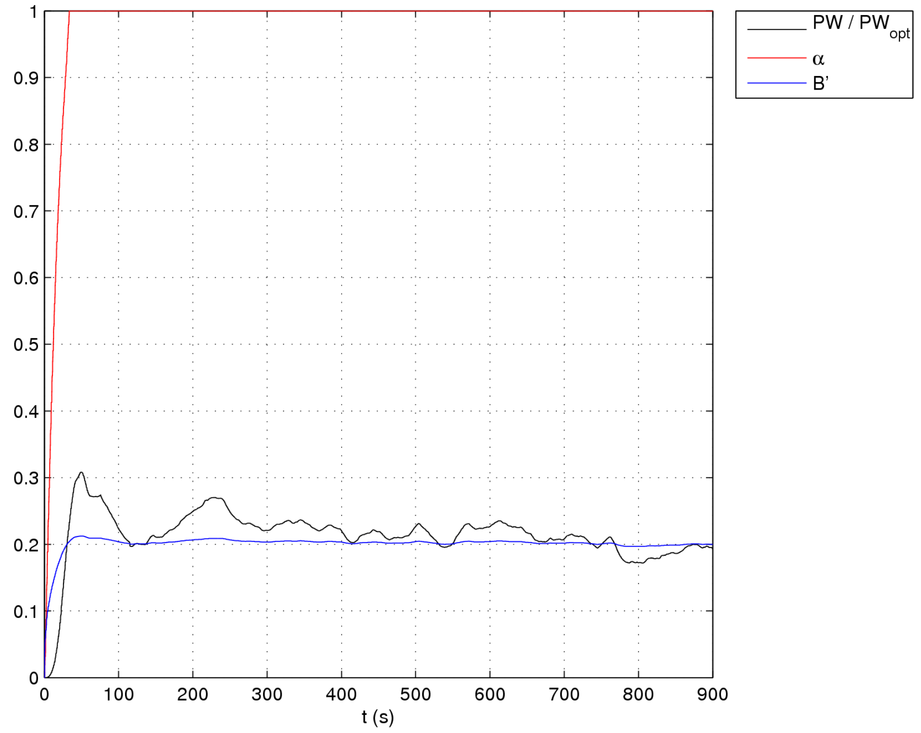


Figure 7.13: $u_0 = 7.5\,\mathrm{m/s}$: evolution of α, B' and normalised PW_{ex} over time

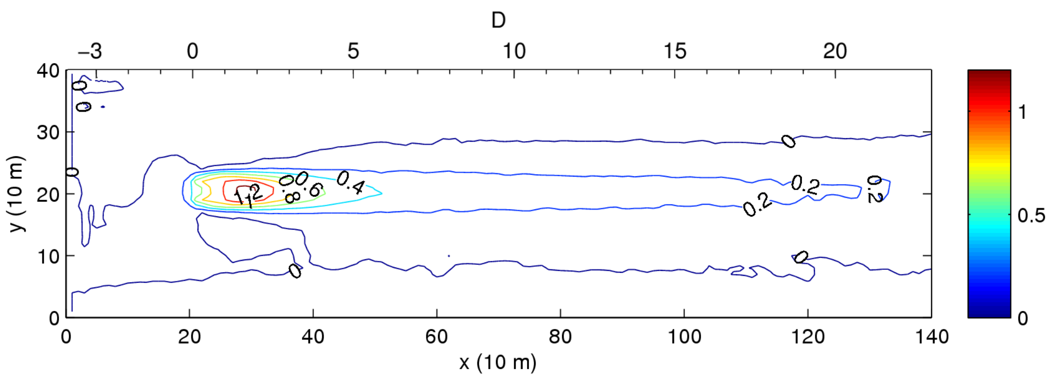


Figure 7.14: $u_0 = 7.5\,\mathrm{m/s}$: planar contour plot of time-averaged velocity deficit. The contour line '0' denotes full wake recovery.

velocity profile plots in figure 7.15 – for $D \approx 15$ for 80% wake recovery. The visible turbulence is at similar levels to $u_0 = 12\,\mathrm{m/s}$, as can been seen in figure 7.16, approximately $Ti_{x,opt} = 0.15$.

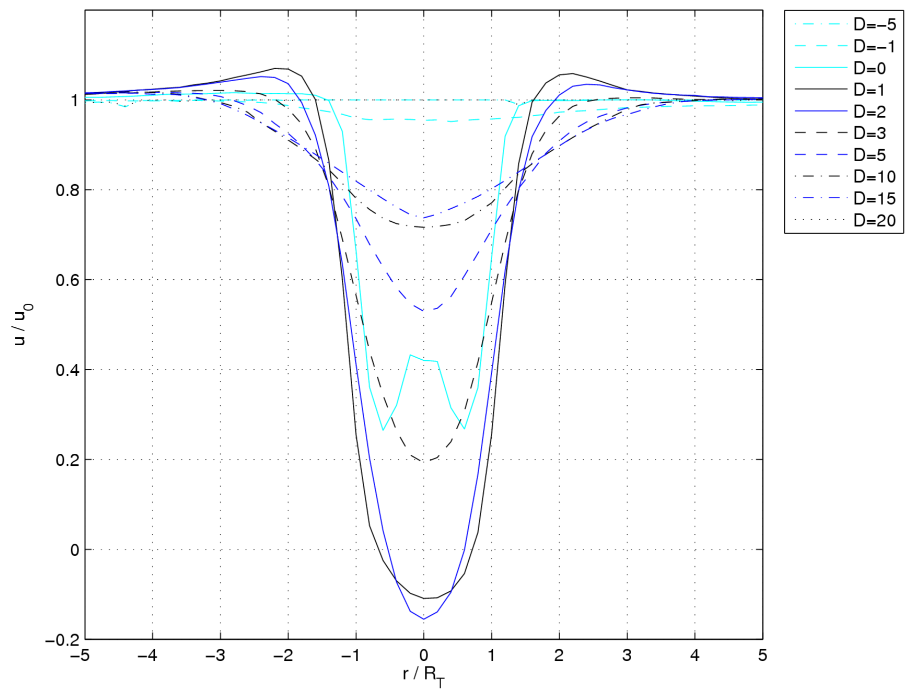


Figure 7.15: $u_0 = 7.5\,\mathrm{m/s}$: spatially and time-averaged u profiles for varying distances upstream and downstream

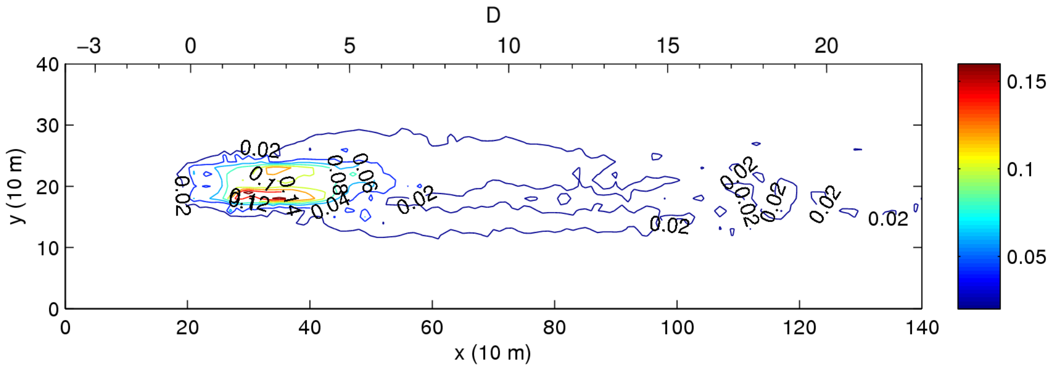


Figure 7.16: $u_0 = 7.5\,\mathrm{m/s}$: planar contour plot of turbulence intensity

Inlet windspeed of $u_0 = 20\,\mathrm{m/s}$

In figure 7.17, the turbine progresses to a stable maximum power output much more quickly than for $u_0 = 12\,\mathrm{m/s}$ – $100\,\mathrm{s}$ as opposed to $600\,\mathrm{s}$ – and that the blade factor α starts to reduce *prior* to $PW_{ex} > PW_{ex,opt}$, which, as in the $12\,\mathrm{m/s}$ case ensures that $\max(PW_{ex}) \leq 1.2\,PW_{ex,opt}$. α is now continually varying, and manages to keep PW_{ex} within 10% of the optimum power, despite the turbulent flow.

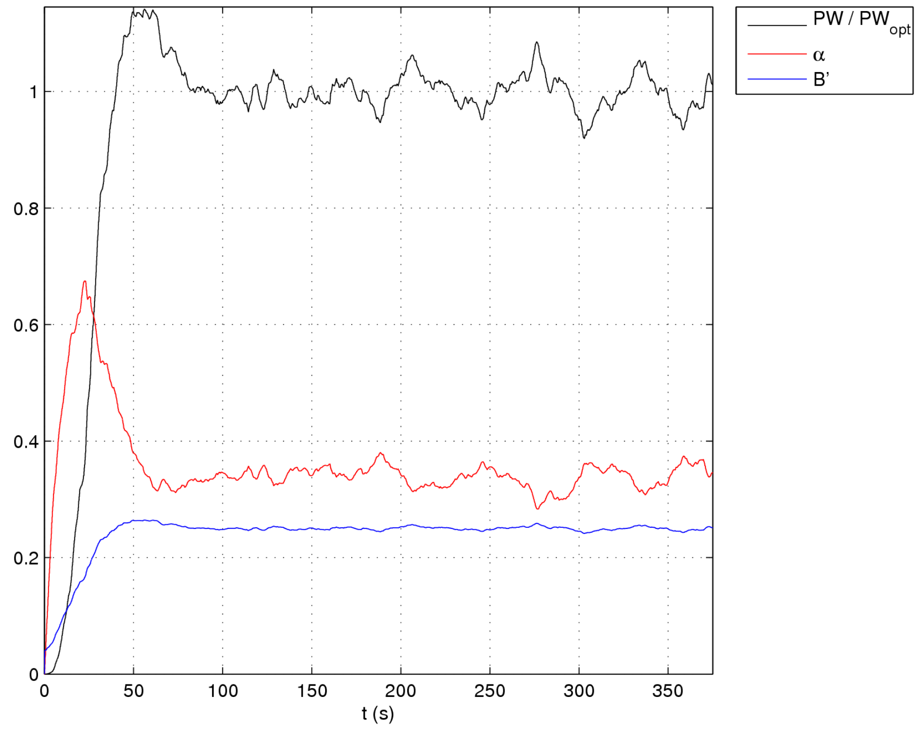


Figure 7.17: $u_0 = 20\,\mathrm{m/s}$: evolution of α, B' and normalised PW_{ex} over time

The velocity deficit plot in fig. 7.18 shows that $\max(d) \approx 0.6$, while the wake is longer than at either $7.5\,\mathrm{m/s}$ or $12\,\mathrm{m/s}$ – reaching approximately $0.75\,u_0$ at $x = 1400\,\mathrm{m}$.

The velocity profiles in fig. 7.19 shows this a little more clearly. What can also be noticed is the more pronounced effect of the hub at $x' = 0$ (just downwind of the turbine). Furthermore, the profiles seem to be more symmetrical than at $u_0 = 12\,\mathrm{m/s}$, which could be down to the increased viscous action due to turbulence.

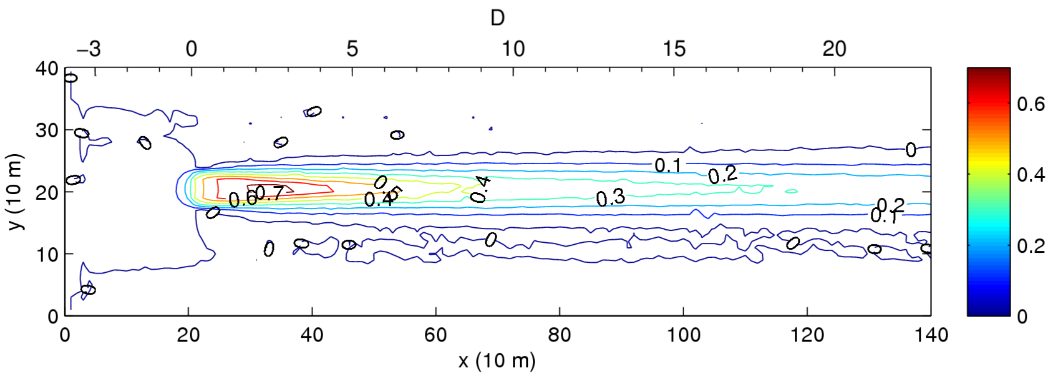


Figure 7.18: $u_0 = 20\,\mathrm{m/s}$: planar contour plot of time-averaged velocity deficit. The contour line '0' denotes full wake recovery.

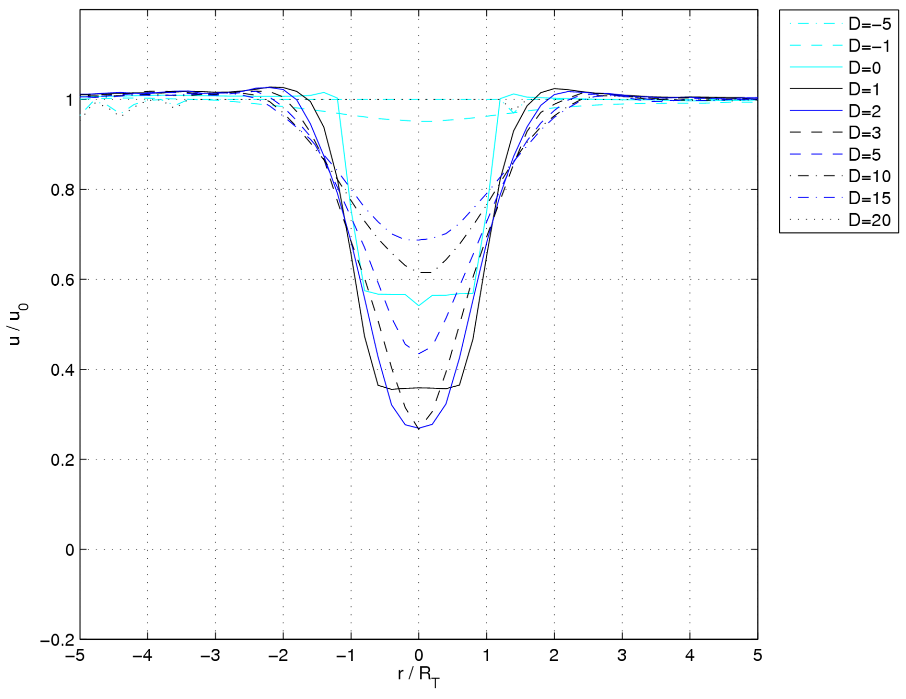


Figure 7.19: $u_0 = 20\,\mathrm{m/s}$: spatially and time-averaged u profiles for varying distances upstream and downstream

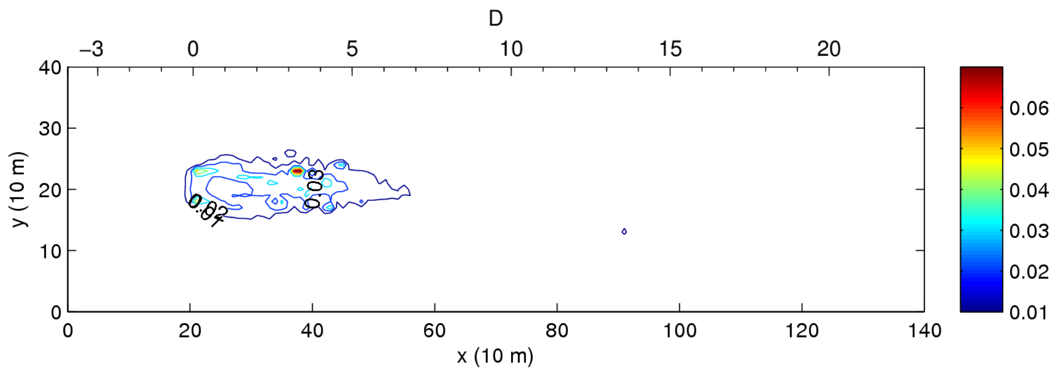


Figure 7.20: $u_0 = 20\,\mathrm{m/s}$: planar contour plot of turbulence intensity

The turbulent contours plot in fig. 7.20 however sees the turbulence intensity peak at *half* the 12 m/s case, with $\max(Ti) \approx 0.07$. This could be down to one of several things:

1. Most of the turbulence within the model is sub-grid, and so cannot be directly measured.
2. The turbulence generating algorithm is ineffective at higher windspeeds.
3. The sampling period of $t_{av} = \left[\frac{LTIME}{2}, LTIME\right]$ is only adequate for lower values of u_0.

These points shall be discussed in chapter 8.

7.4.4. Vestas V52: without turbulent inlet conditions

Overall performance

From fig. 7.21 we can see that c_P at $u_{0,opt}$ peaks slightly lower than the turbulent inlet case at $c_P \approx 0.365$. Below this, the power output is roughly similar to the turbulent inlet simulations. This is also the case above $u_{0,opt}$ – to be expected, due to the hard limiting of the power output to $PW_{ex,opt}$. via α. At lower wind speeds, excepting the peak at $u_0 = 10\,\mathrm{m/s}$, the overall efficiency is less than the turbulent case as shown in fig. 7.10.

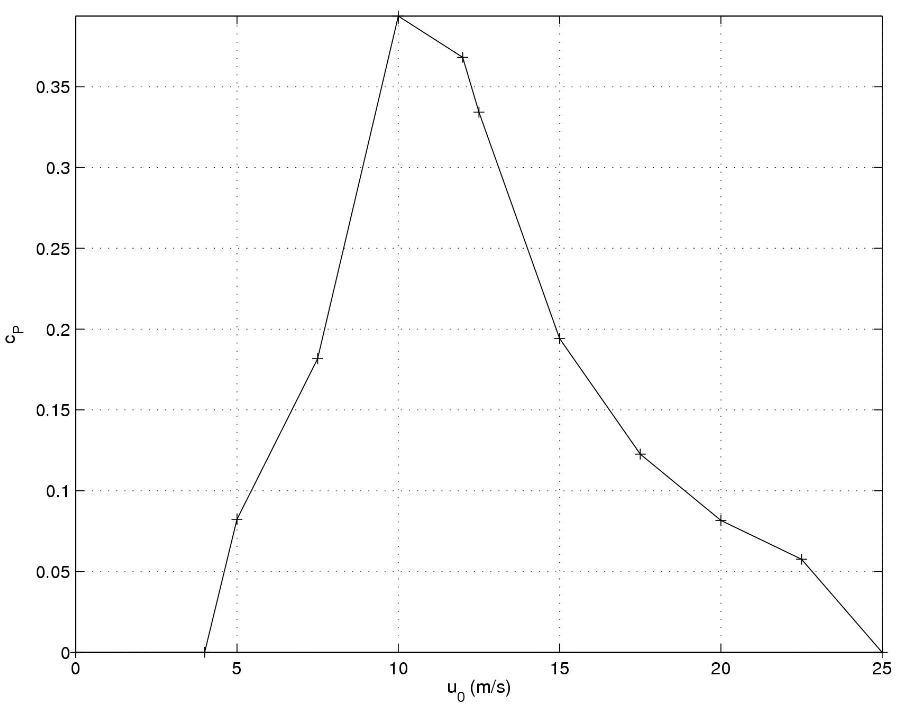


Figure 7.21: Performance curve for V52 without inlet turbulence, peaking at $u_0 = 12\,\mathrm{m/s}$ with $c_P \approx 0.36$; lower than the turbulent inlet case

Inlet condition $u_0 = 12, \mathbf{m/s}$

From fig. 7.22 it is clear that the power fluctuates less than the turbulent inlet case. The velocity profile and velocity deficit plots in figs. 7.23 and 7.25 respectively show that despite c_P being higher, the peak velocity deficit and wake recovery remain almost the same.

Inlet condition $u_0 = 20, \mathbf{m/s}$

Comparing the non-turbulent inlet power output fig. 7.26 with fig. 7.17 shows a smaller variation in PW_{ex}. The velocity deficit in fig. 7.27 is almost identical to the turbulent inlet case shown in fig. 7.18; contrasting the two velocity profile cases (fig. 7.19 and fig. 7.28) also shows little change.

Finally, the planar turbulence intensity slice shows in fig. 7.29 shows slightly less turbulence than the turbulent inlet case, fig. 7.20.

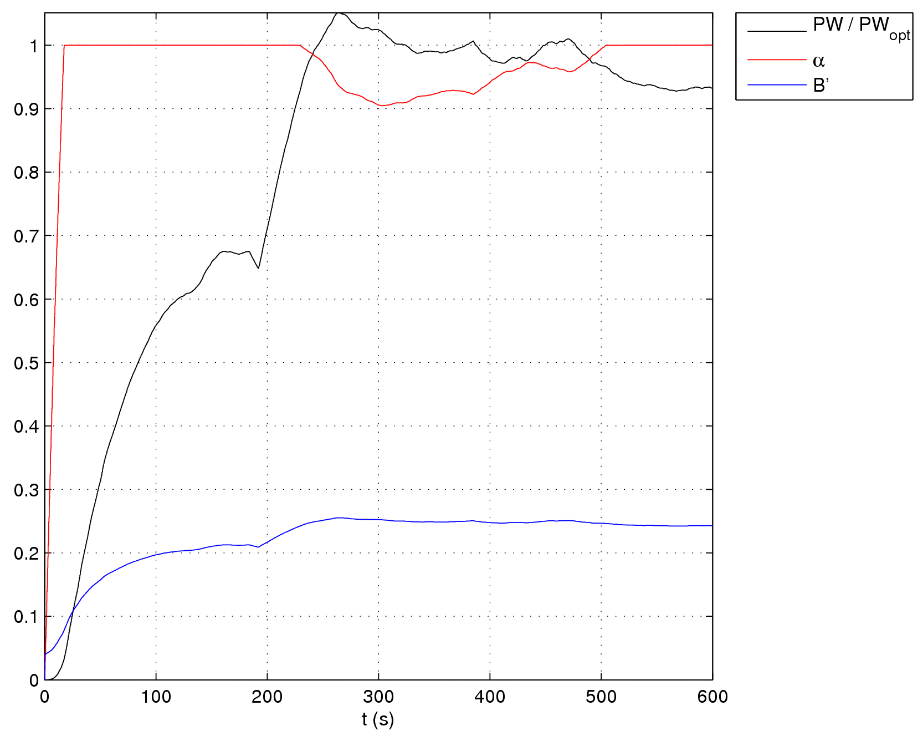


Figure 7.22: $u_0 = 12\,\mathrm{m/s}$: evolution of α, B' and normalised PW_{ex} over time

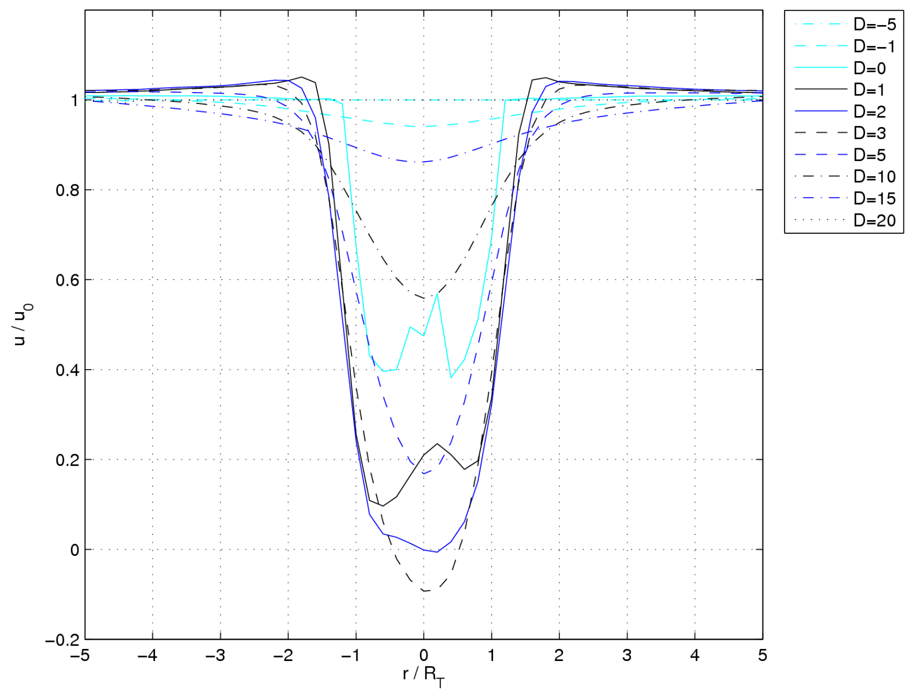


Figure 7.23: $u_0 = 12\,\mathrm{m/s}$: spatially and time-averaged u profiles for varying distances upstream and downstream

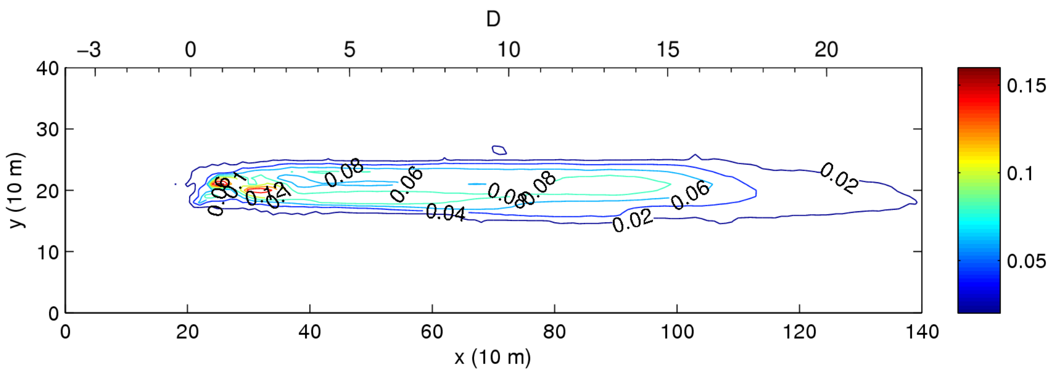


Figure 7.24: $u_0 = 12\,\mathrm{m/s}$: planar contour plot of turbulence intensity

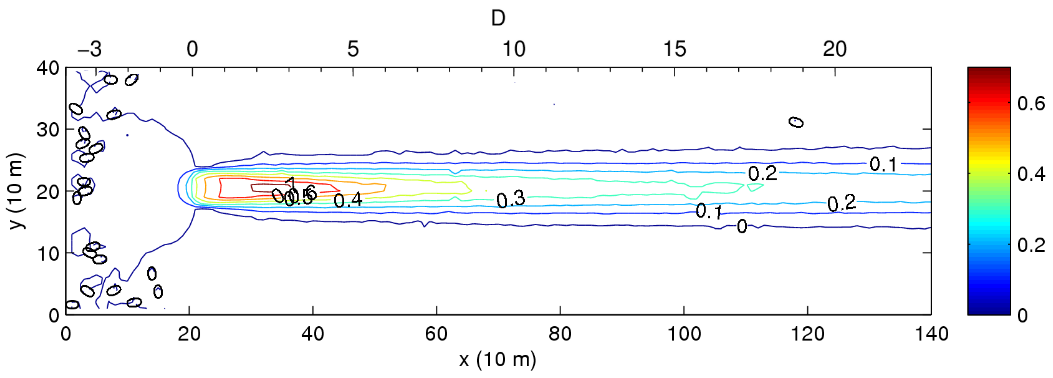


Figure 7.25: $u_0 = 12\,\mathrm{m/s}$: planar contour plot of time-averaged velocity deficit. The contour line ‘0’denotes full wake recovery.

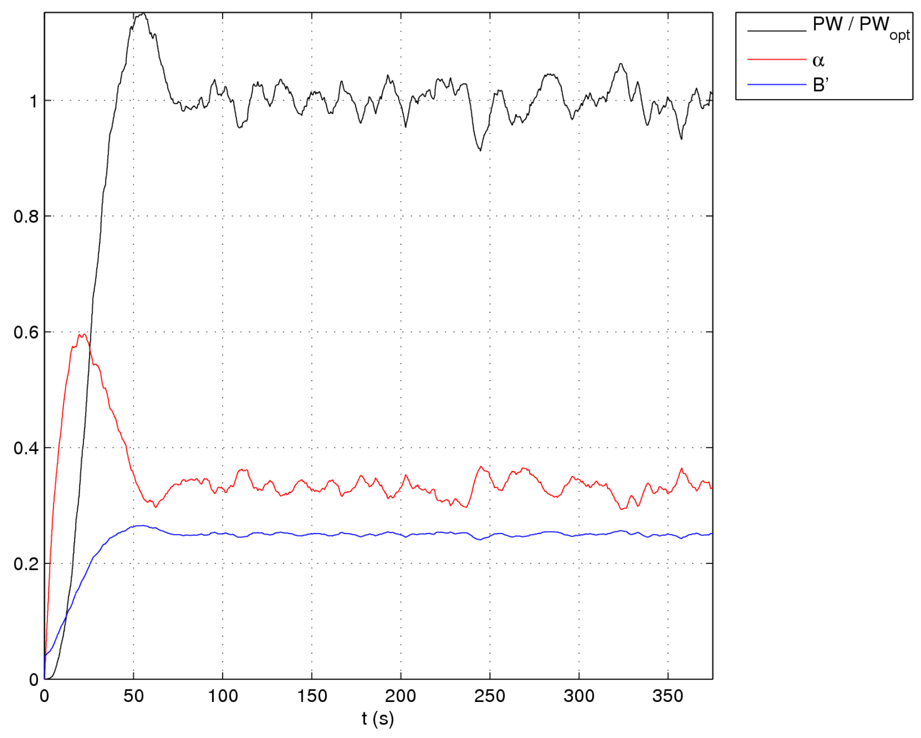


Figure 7.26: $u_0 = 20\,\mathrm{m/s}$: evolution of α, B' and normalised PW_{ex} over time

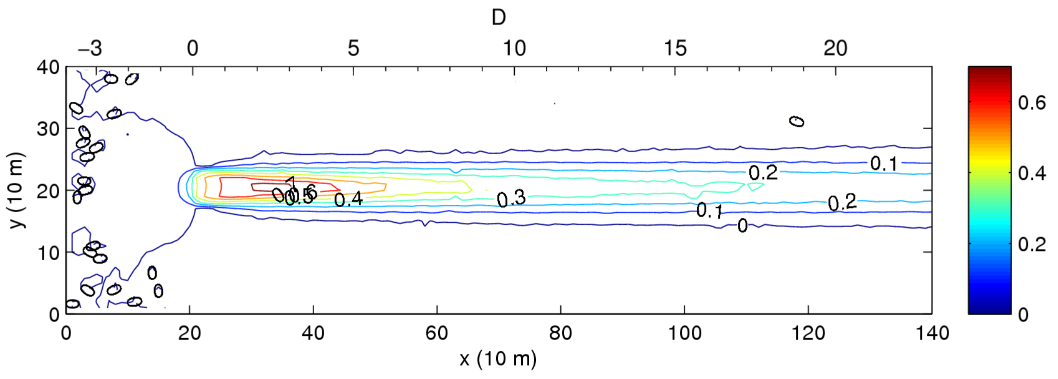


Figure 7.27: $u_0 = 20\,\mathrm{m/s}$: planar contour plot of time-averaged velocity deficit. The contour line '0' denotes full wake recovery.

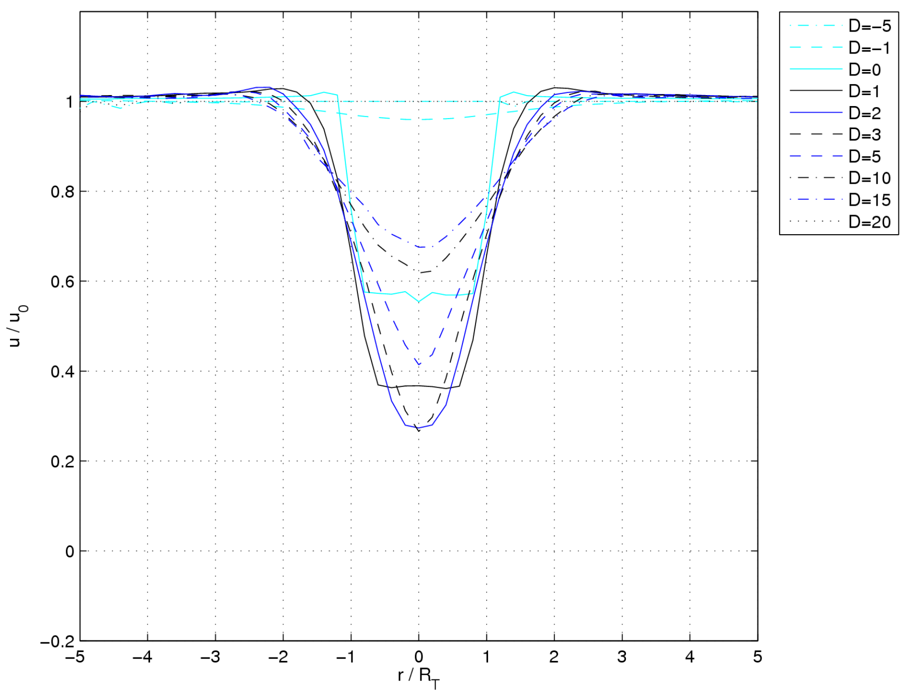


Figure 7.28: $u_0 = 20\,\mathrm{m/s}$: spatially and time-averaged u profiles for varying distances upstream and downstream

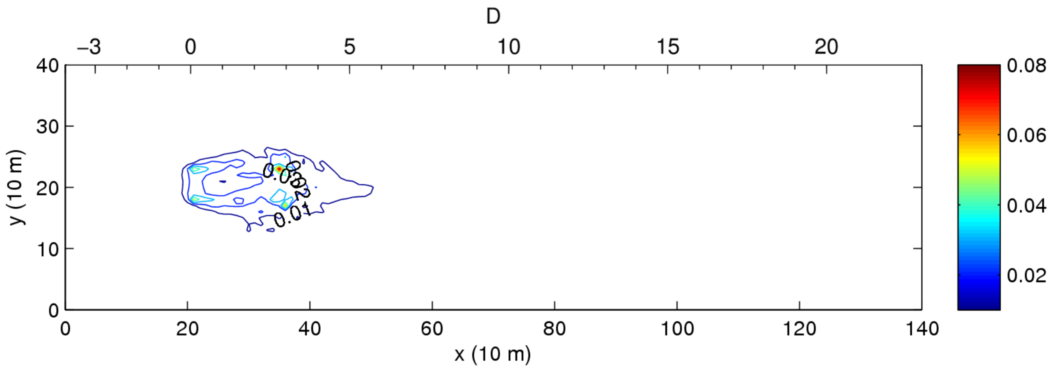


Figure 7.29: $u_0 = 20\,\mathrm{m/s}$: planar contour plot of turbulence intensity

7.5. Results for Seaflow marine turbine

7.5.1. Channel with rigid lid: inlet condition $u_0 = 2.70\,\mathrm{m/s}$

As with the Vestas V52 simulations, there was a spin-up period during which experimental data was considered unreliable. This was further complicated by the the ramped boundary conditions, with the maximum u_0 occuring at the boundary after $t_{ramp} = 3000\,\mathrm{s}$. This mean that any period of stability had to be considered only for $t > 3000\,\mathrm{s}$. Beyond that critera, this was ascertained through inspection of global turbine properties such as PW_{ex} and planar velocity slices as before.

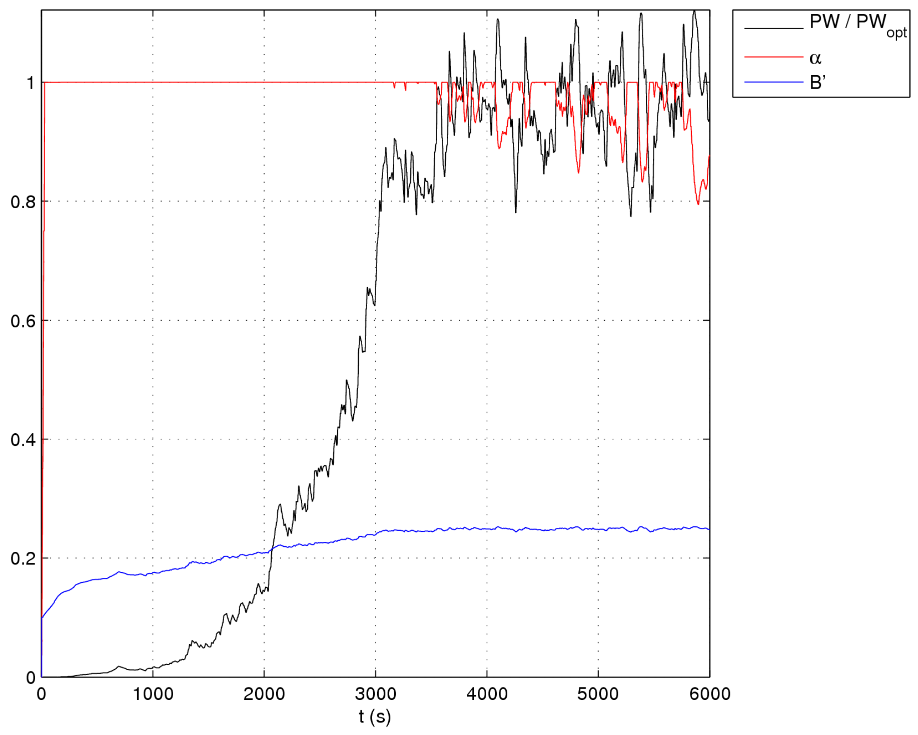


Figure 7.30: Evolution of α, B' and normalised PW_{ex} over time

From fig. 7.30 we can see that, as expected the power output reaches its maximum after t_{ramp} at $t \approx 3500\,\mathrm{s}$. We can also see the limiting action of α, 'kicking in' as it does when PW_{ex} occasionally exceeds $PW_{ex,max}$. Unlike the Vestas turbine however, the wake behind Seaflow exhibits no recirculation whatsoever, and significant dips in the power as per the Vestas in fig. 7.4.

Figure 7.33 shows the planar slice of $\frac{u}{u_0}$ averaged over a period $t_{av} = 1500\,\mathrm{s}$,

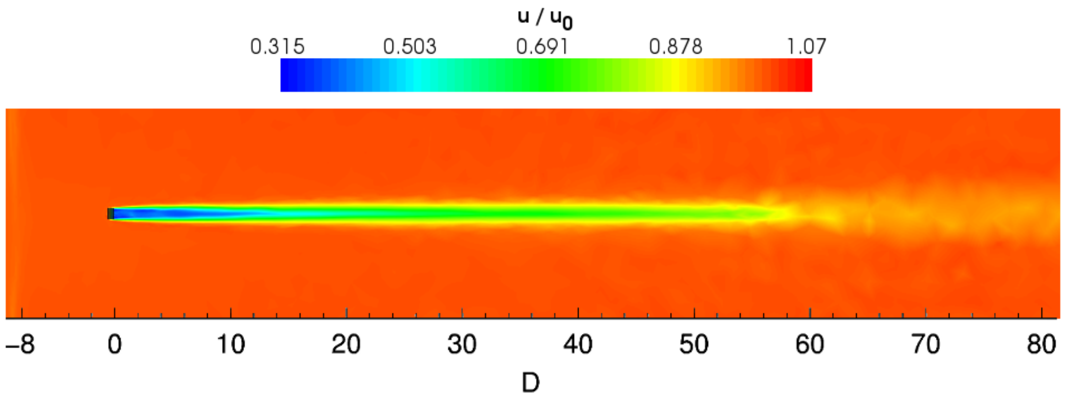


Figure 7.31: Planar slice showing u/u_0 at $t = 4000\,\mathrm{s}$. The turbine volume is represented by the black rectangle.

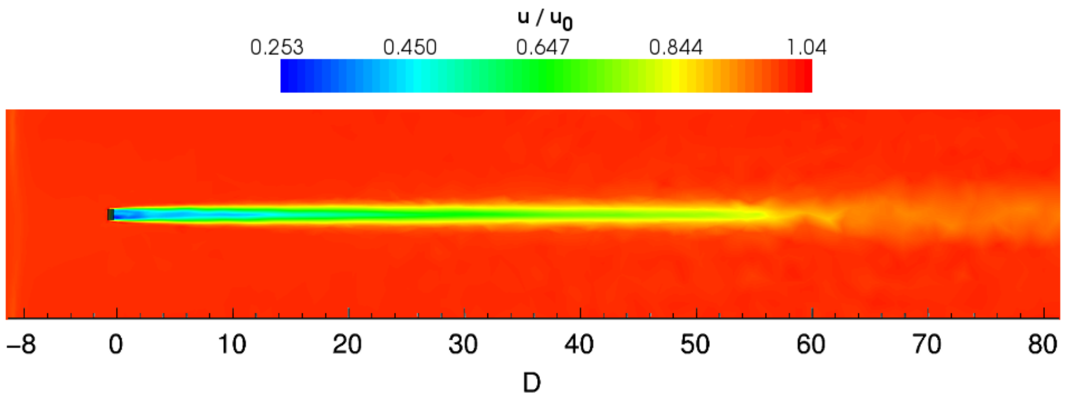


Figure 7.32: Planar slice showing u/u_0 at $t = 6000\,\mathrm{s}$.

taken from 4500 s to 6000 s. We can see that the velocity deficit reaches a peak of $0.4\,u_0$ between $1 - 9D$ downstream of the turbine, and that the '0' contour lines demonstrate that the water accelerates around the the turbine: in contrast with the Vestas example in fig. 7.7, the upstream '0' contour forms a straight rather than curved line. Lastly, the wake behind the turbine shows a 80% recovery ($u \approx 0.8\,u_0$) at $x \approx 500\,\mathrm{m}$, almost $36D$ downstream.

The cross-sectional profile of u/u_0 in fig. 7.34 shows that upstream the flow is relatively undisturbed at $x' = -5D \ldots -1D$. whereas at $x' = 0$ develops into a shallow 'W' shape, with minima of $u = 0.36\,u_0$ occuring at $r \approx \pm\frac{3}{4}R_T$, with u returning to at $r = \pm 1\frac{1}{2}\,R_T$. Further downstream, the wake retains this 'W' profile until $x' = 5D$, whereupon the far wake develops into a 'U' shape.

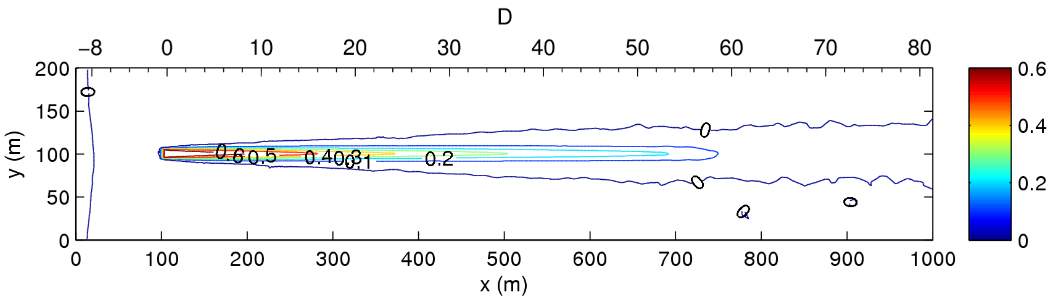


Figure 7.33: Planar contour plot of time-averaged velocity deficit. The contour line '0' denotes full wake recovery.

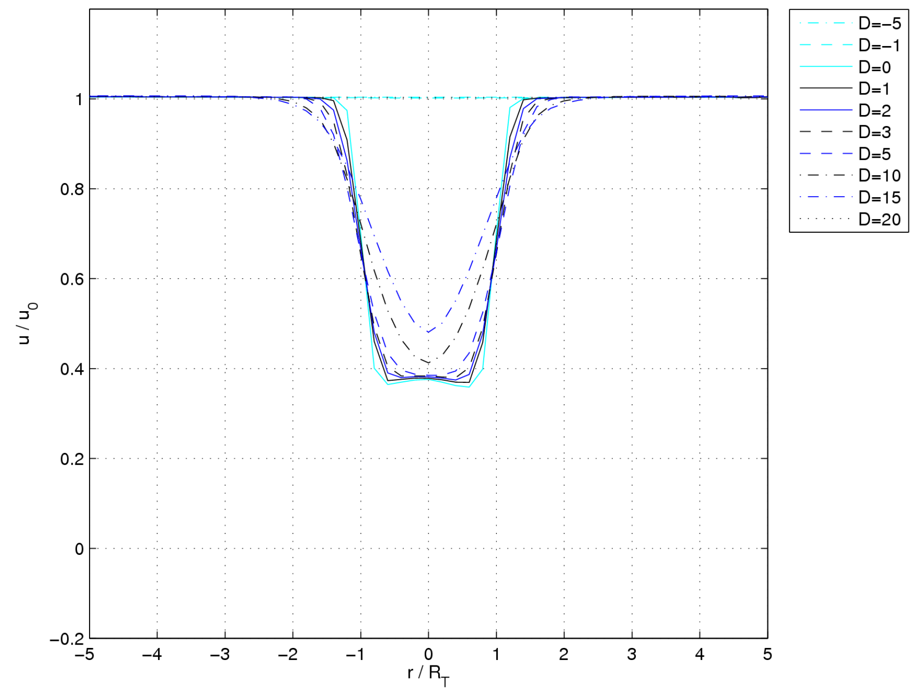


Figure 7.34: Spatially and time-averaged u profiles for varying distances upstream and downstream

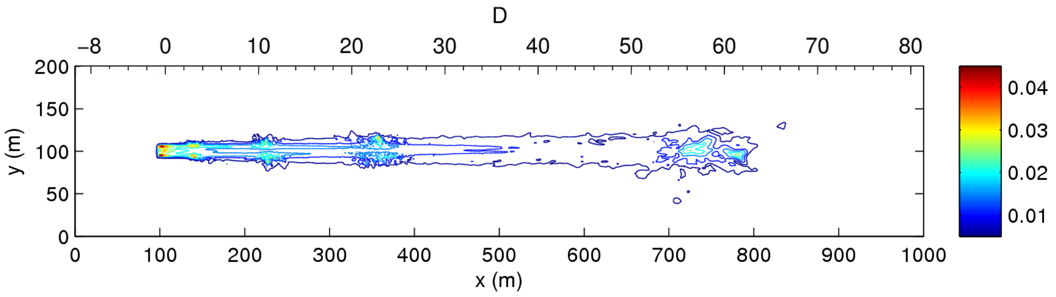


Figure 7.35: Planar contour plot of turbulence intensity

Also noteworthy at this stage is the gradual fanning out of the wake, until at $x' = 15D$ it is clear 'V' with a minimum of $u(r = 0) = 0.44\,u_0$; $u \to u_0$ at $r = \pm 3\frac{1}{2}\,R_T$. There is a very slight accelerative bulge for $-3R_T < r > 3R_T$, but this is much less pronounced than that of the Vestas V52 turbine as shown in fig. 7.8.

The planar slice of turbulence intensity calculated over t_{av} in fig. 7.35 shows a peak of $Ti = 0.04$ at $x' = 1D$, which decays to $Ti = 0.015$ at $x' = 15D$, and finally to negligible levels at $x' = 60D$. The exceptions to this wake are two peaks of $Ti = 0.02$ and $Ti = 0.015$, at $x' = 24D$ and $x' = 58D$ respectively.

7.5.2. Channel with rigid lid: overview of performance

The time over which the averages were calculated, t_{av}, was defined as

$$t_{av} = [\,0.75\,LTIME,\; LTIME\,] \tag{7.15}$$

which gave results that were representative. Fig. 7.36 shows the power co-efficient as a function of u_0: starting at 0 for $u_0 = 0\,\text{m/s}$, it sharply jumps to $c_p = 0.24$ at $u = 0.5\,\text{m/s}$ (there is no lower cut-in flowspeed). c_p increases step-wise to 0.37 at $u_0 = 1.5\,\text{m/s}$, decreasing slightly to 0.35 at $u_{0,opt}$. Beyond this point, c_p drops sharply due to PW_ex being limited to $PW_{ex,max}$.

The plot in fig. 7.37 shows how the average $\overline{\alpha}$ decreases with increasing u_0 whilst keeping $\overline{PW}_{ex}$ constant, dropping to $\overline{\alpha} = 0.4$ at $u_0 = 5.00\,\text{m/s}$. At $u_{0,opt}$ we can see that both $\overline{\alpha} < 1$ and $\overline{PW}_{ex} < PW_{ex,max}$. At lower flowspeeds, $\overline{\alpha} = 1$ since no limiting occurs. The circulation plot (fig. 7.38) shows that the

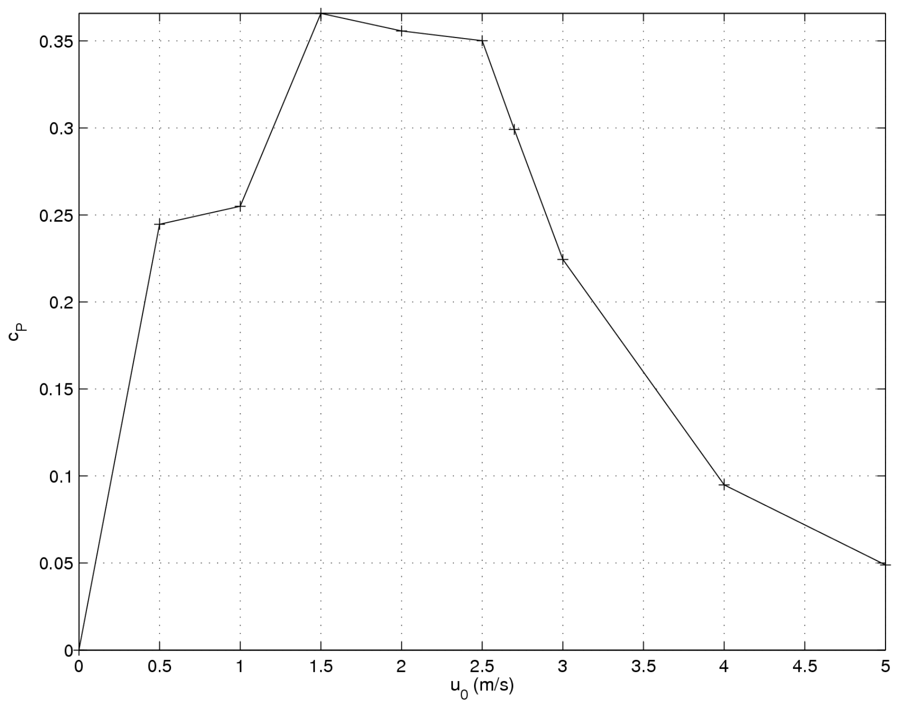


Figure 7.36: Performance curve for Seaflow, power output peaking at $u_0 = 2.70\,\mathrm{m/s}$.

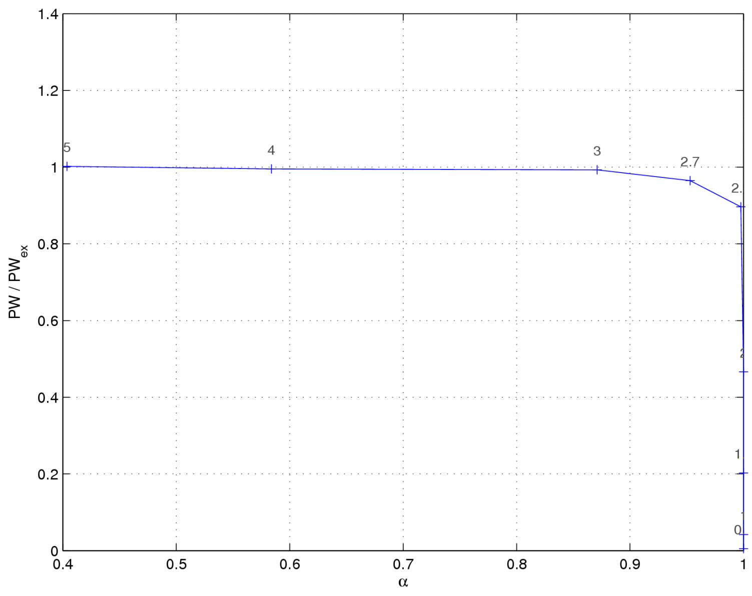


Figure 7.37: Plot of blade factor α versus PW_{ex}/PW_{opt}. The numbers next to the points represent u_0 in m/s.

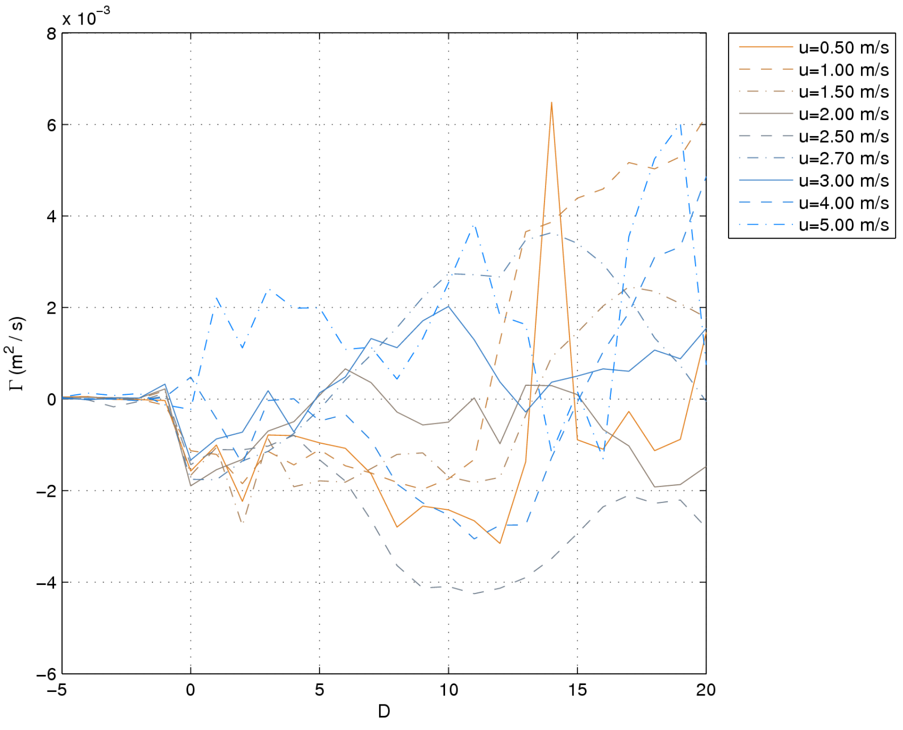


Figure 7.38: Plot of D versus circulation Γ for varying u_0

turbine at most flow speeds does produce a negative circulation at $x' = 0$, the outlet of the turbine volume, however further downstream the circulation varies randomly, which suggests that the turbine itself may be generating turbulence of a larger magnitude than the net orbital motion of the water due to the turbine blades.

7.5.3. Channel with rigid lid: comparison between flow speeds

Inlet flow speed of $u_0 = 1.00\,\mathrm{m/s}$

At this low speed, the effecitive solidity takes until $t = 4500\,\mathrm{s}$ (1 hour 15 minutes) to reach 0.19, at the same time the mean of $PW_{ex} \rightarrow 0.04\,PW_{ex,opt}$ – see fig. 7.39; both show considerably less variation than at $u_0 = 2.70\,\mathrm{m/s}$. Moreover, α quickly reaches its maximum of 1 and remains there, since no limiting is required in slow-moving flow.

In fig. 7.40 the velocity deficit contour plot shows a $d = 0$ contour at $x' = 7D$. The deficit peaks with $\max(d) = 0.35$ where $D \leq x' \leq 11D$, in an area the width of the turbine. The wake has about 90% recovery at 62

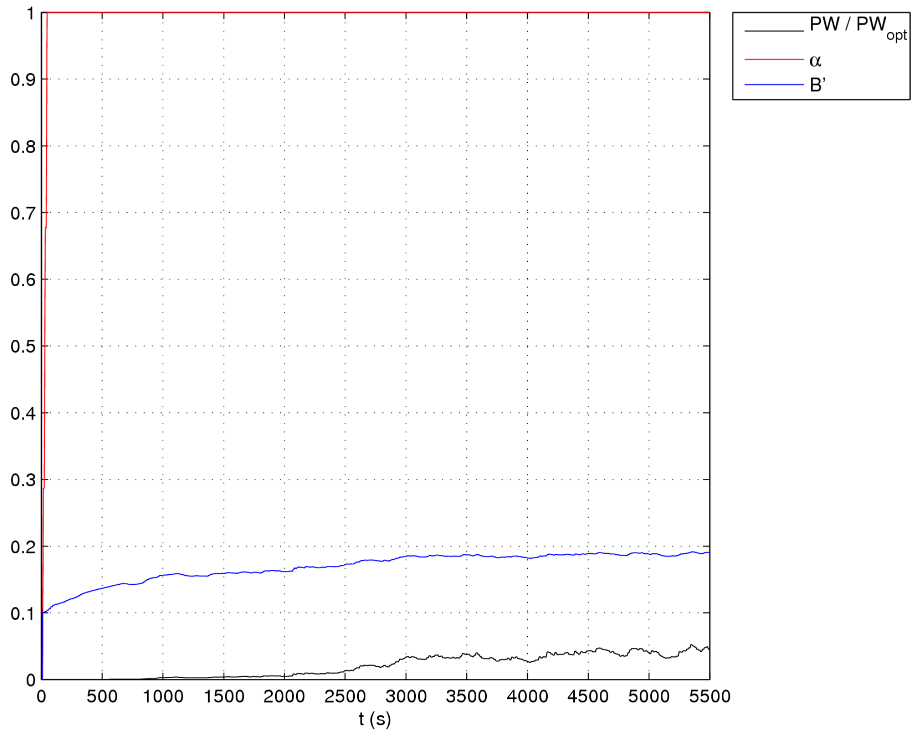


Figure 7.39: $u_0 = 1.00\,\mathrm{m/s}$: α, B' and normalised PW_{ex} over time

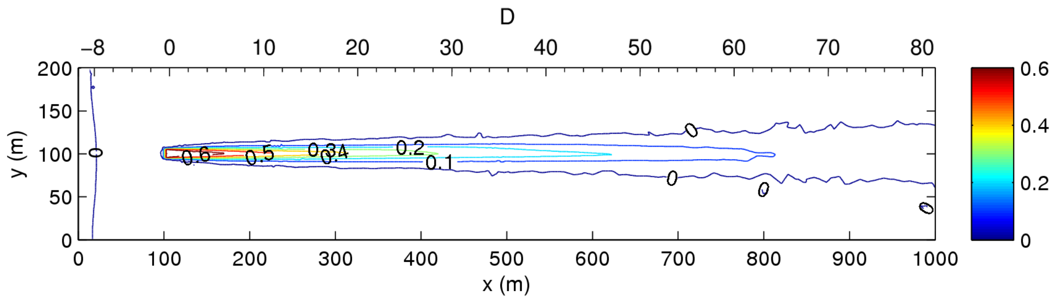


Figure 7.40: $u_0 = 1.00\,\mathrm{m/s}$: planar contour plot of time-averaged velocity deficit. The contour line '0' marks the recovery to free-stream velocity.

diameters downstream. The velocity profile plot (fig. 7.41) shows the cross-sectional shape of the wake, transforming from a shallow 'W' at $x' = 0$ to a 'U', and then to a shallow 'V' shape, with wake broadening as it does so. The visible turbulence contours in fig. 7.42 display a peak of $Ti = 0.055$ at $0 \leq x' \leq 1D$, and a couple of lesser peaks in turbulence occur at $x' = \{14D,\, 35D\}$; these could be either due to t_{av} being too short, or the error tolerances $\Delta\epsilon_u$, $\Delta\epsilon_v$ and $\Delta\epsilon_w$ being too high and so not adequately resolving the flow.

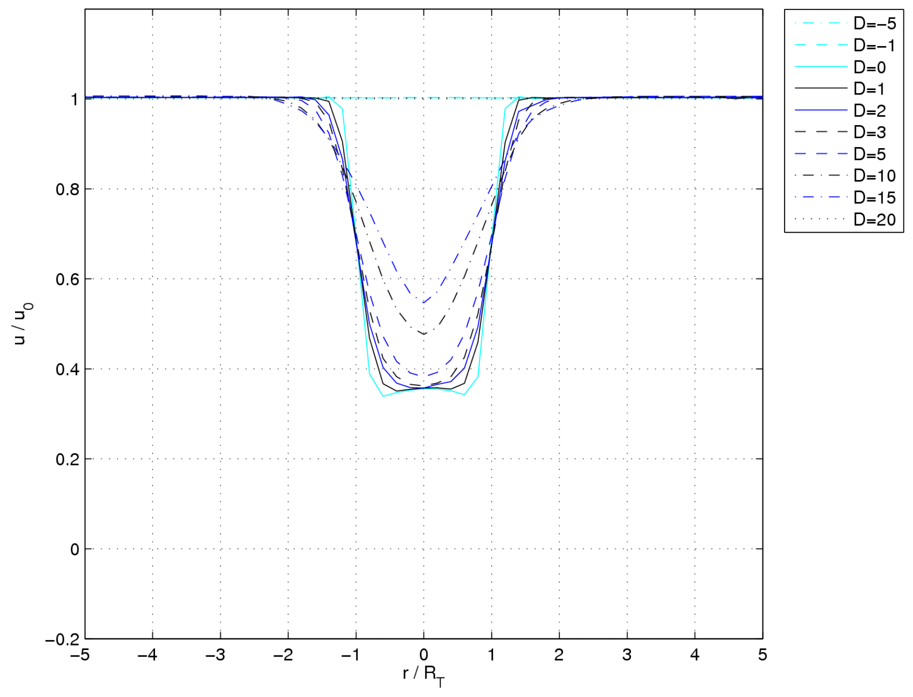


Figure 7.41: $u_0 = 1.00\,\mathrm{m/s}$: spatially and time-averaged u profiles for varying distances upstream and downstream

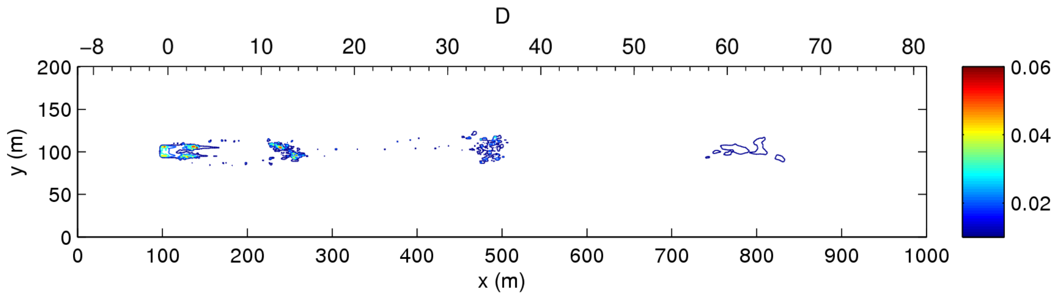


Figure 7.42: $u_0 = 1.00\,\mathrm{m/s}$: planar contour plot of turbulence intensity

Inlet flow speed of $u_0 = 5.00\,\mathrm{m/s}$

The evolution over time of the turbine performance in fig. 7.43 shows PW_{ex} reaching its maximum by $t = 1800\,\mathrm{s}$, as $B' \to 0.24$, in less than half the time of the Seaflow case for $u_0 = 2.70\,\mathrm{m/s}$. The action of hard-limiting by α can be seen, as it also drops rapidly at $= 1800\,\mathrm{s}$, reaching a minimum of $\alpha \approx 0.4$, at $2500\,\mathrm{s}$ fluctations excepted. It will be noted that α to a degree varies conversely with PW_{ex}, albeit with a time delay of approximately a minute.

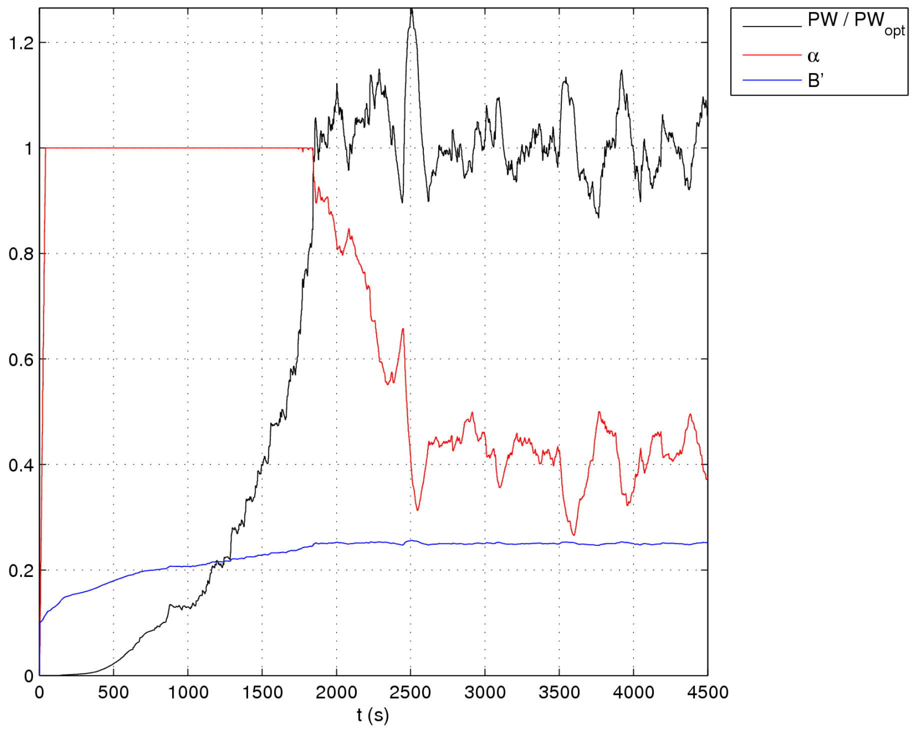


Figure 7.43: $u_0 = 5.00\,\mathrm{m/s}$: evolution of α, B' and normalised PW_{ex} over time

In fig. 7.44 we see a familiar wake structure, however with one main difference. Wake recovery occurs much earlier than before; u regainining 90% of u_0 at $x' \approx 25D$; approximately half the wake recovery distance for the $u_0 = 2.70\,\mathrm{m/s}$ case. The finer-grain detail is revealed in fig. 7.45 – surprisingly, $\min(u) \approx 0.4\,u_0$, a similar velocity deficit to before. The turbulence contours in fig. 7.46 show a peak of $Ti = 0.085$ immediately downstream of the turbine, dropping down to below 1% beyond $x' = 30D$.

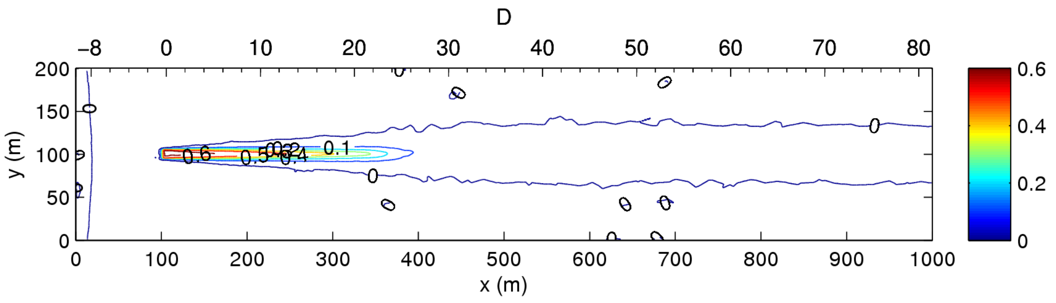


Figure 7.44: $u_0 = 5.00\,\mathrm{m/s}$: planar contour plot of time-averaged velocity deficit.

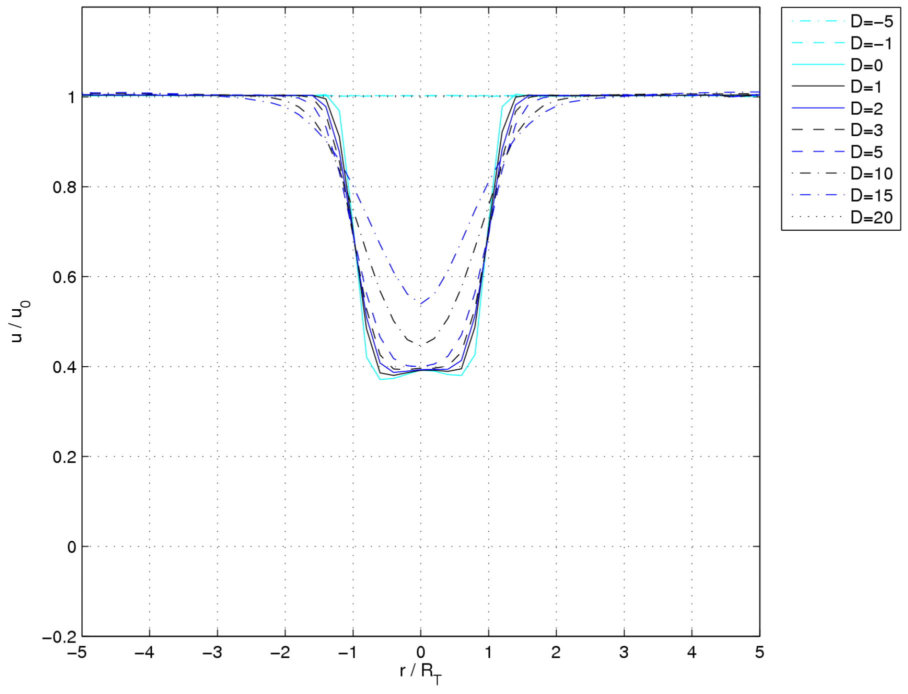


Figure 7.45: $u_0 = 5.00\,\mathrm{m/s}$: spatially and time-averaged u profiles for varying distances upstream and downstream

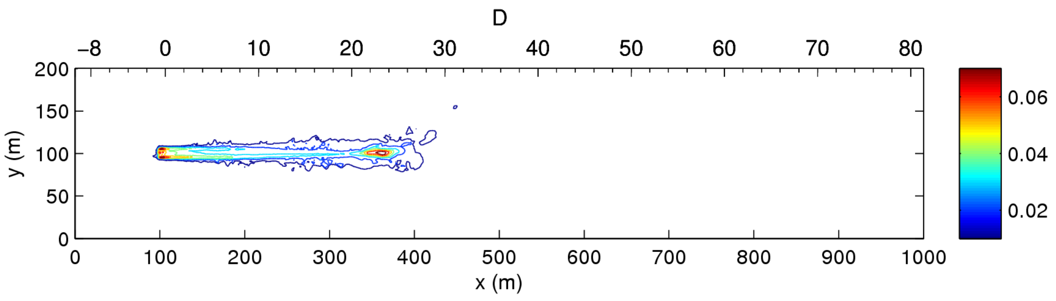


Figure 7.46: $u_0 = 5.00\,\mathrm{m/s}$: planar contour plot of turbulence intensity

7.5.4. Channel with vertical velocity gradient and bottom drag

As in previous cases, we can see the spin-up period in fig. 7.47; once again, the limiting effect of α kicks in, with $\alpha \approx 0.7$ after 3000 seconds.

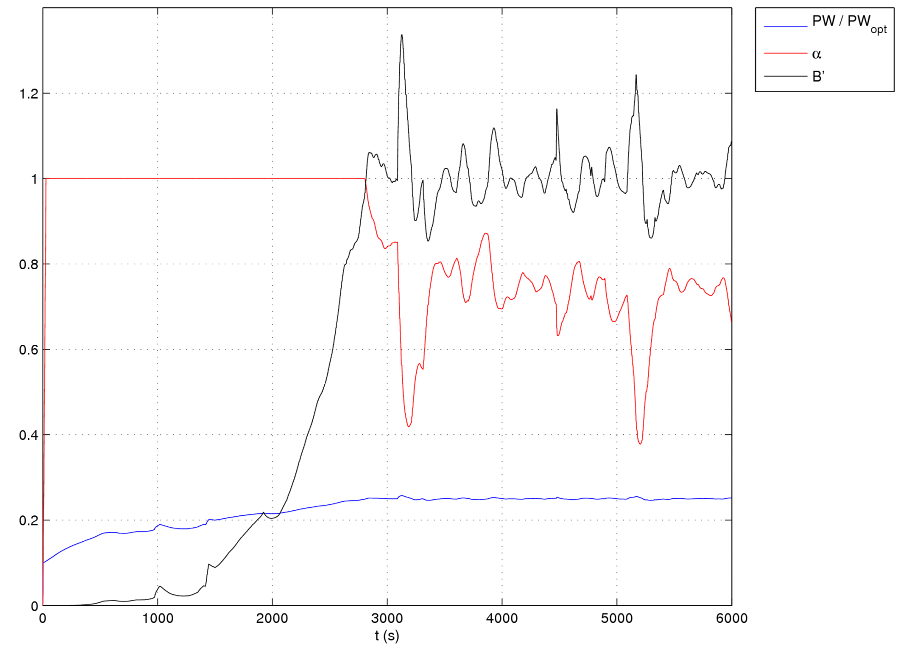


Figure 7.47: Evolution of α, B' and normalised PW_{ex} over time

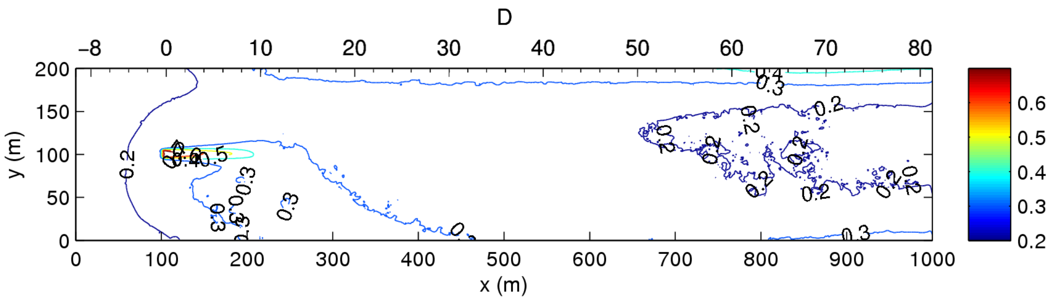


Figure 7.48: Horizontal plane contour plot of time-averaged velocity deficit.

Figure 7.48 shows the velocity deficit reaching $d = 0.7$ at $x' = 1D$ downstream of the turbine. Bearing in mind that $d = 0.2$ represents full wake recovery at $z = 20\,\mathrm{m}$ due to the bottom drag-induced vertical velocity profile, we can see that wake recovery in the horizontal plane at this height occurs at approximately $17D$. There is also a slight asymmetry to the flow; an instantaneous slice of u/u_0 at $y = 4000\,\mathrm{s}$ in fig. 7.49 shows a tongue of fast-moving

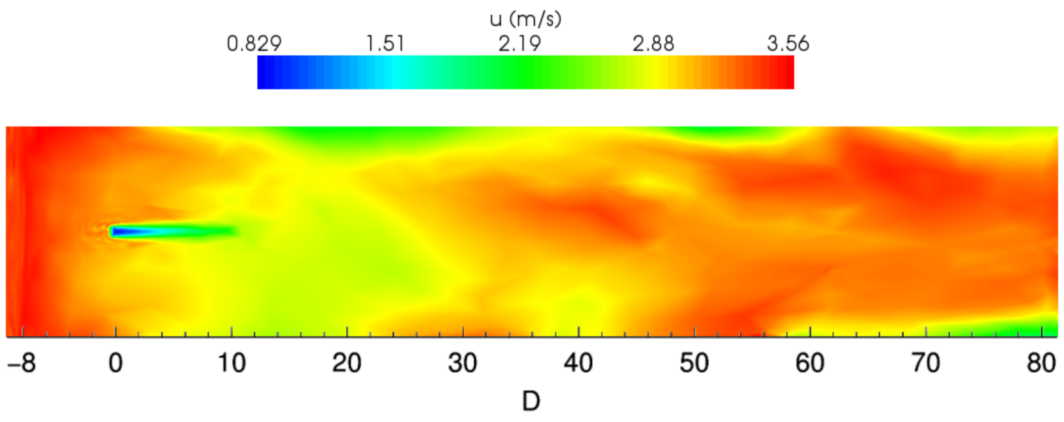


Figure 7.49: Horizontal planar slice showing u/u_0 at $t = 4000\,\mathrm{s}$. The turbine volume is represented by the black rectangle.

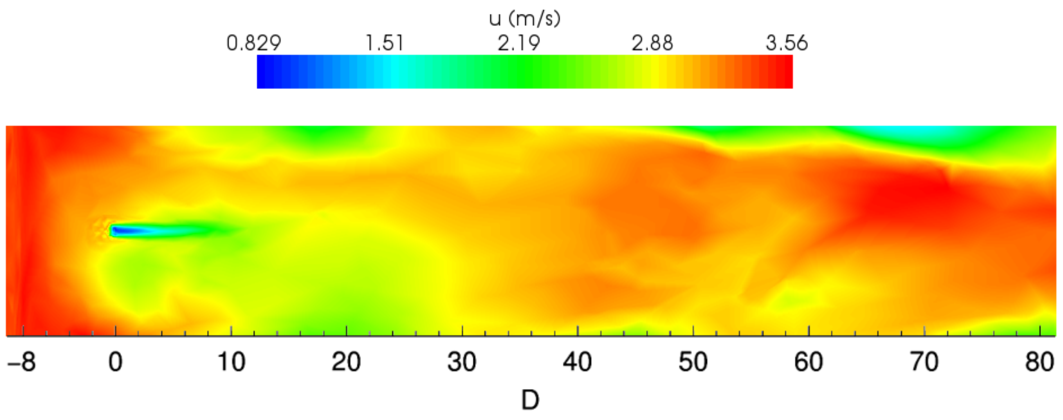


Figure 7.50: Horizontal planar slice showing u/u_0 at $t = 6000\,\mathrm{s}$.

fluid at $y > 100\,\mathrm{m}$, which by $t = 6000\,\mathrm{s}$ has begun to dissipate and be balanced by a similar tongue $y < 100\,\mathrm{m}$. Looking at the vertical slice in fig. 7.51 we can see the strong vertical gradient in the velocity, but we can also see that the disturbance to the flow occurs over a longer distance. Looking at the vertical component of the velocity in fig. 7.51, we can see that there is an upwelling before the turbine of $w \approx 0.2\,\mathrm{m/s}$, and a downwelling of $0 > w > -0.05\,\mathrm{m/s}$ downstream.

In fig. 7.53 we see the horizontal profile of the turbulence intensity, which peaks at $Ti = 0.12$. It appears to be largely uniform for $x > 200\,\mathrm{m}$, with some strong turbulence near the side of the tidal channel for $y = 200\,\mathrm{m}$ and $x > 500\,\mathrm{m}$. There is also a narrow trail of turbulence behind the turbine of

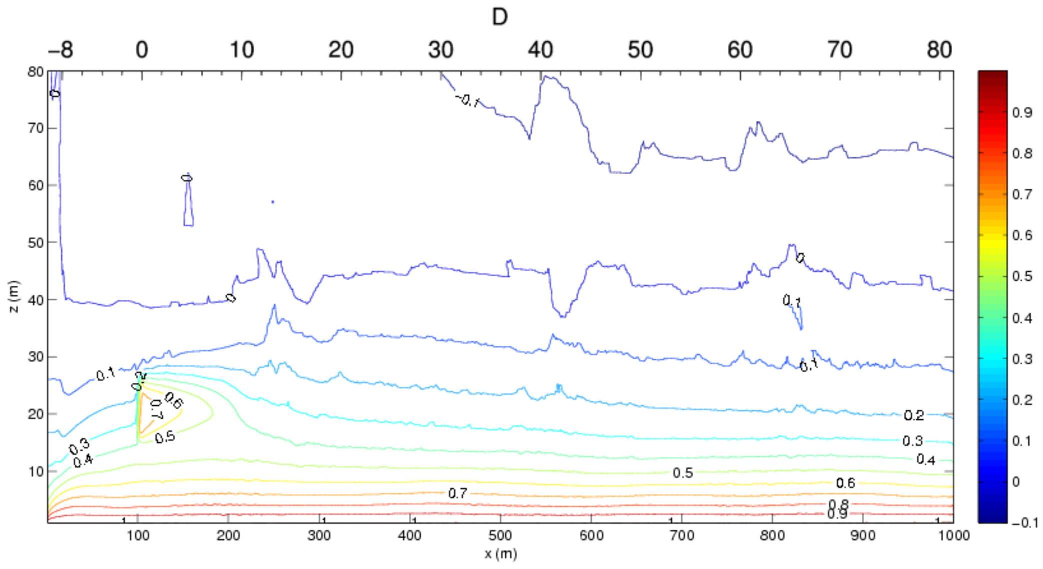


Figure 7.51: Vertical plane contour plot of time-averaged velocity deficit.

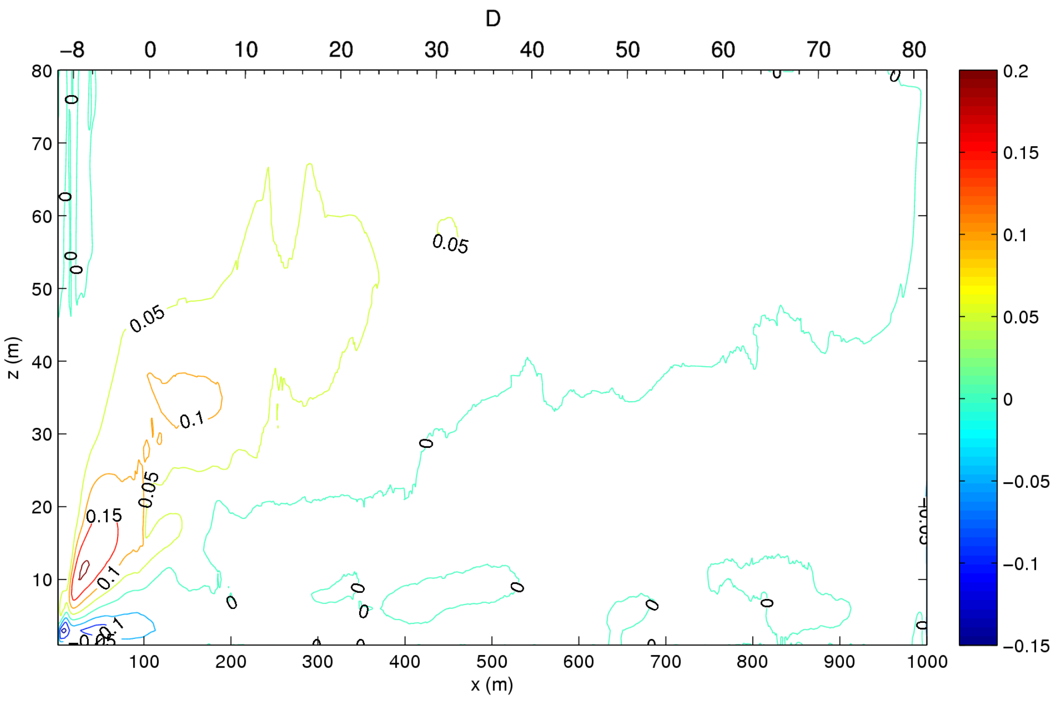


Figure 7.52: Vertical plane contour plot of time-averaged w in m/s.

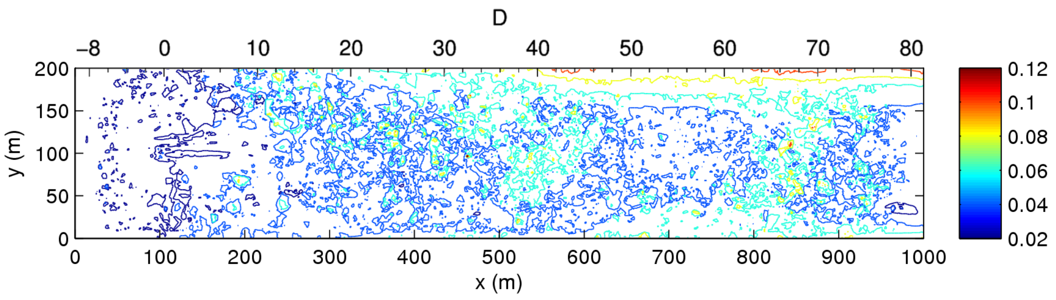


Figure 7.53: Horizontal plane contour plot of turbulence intensity

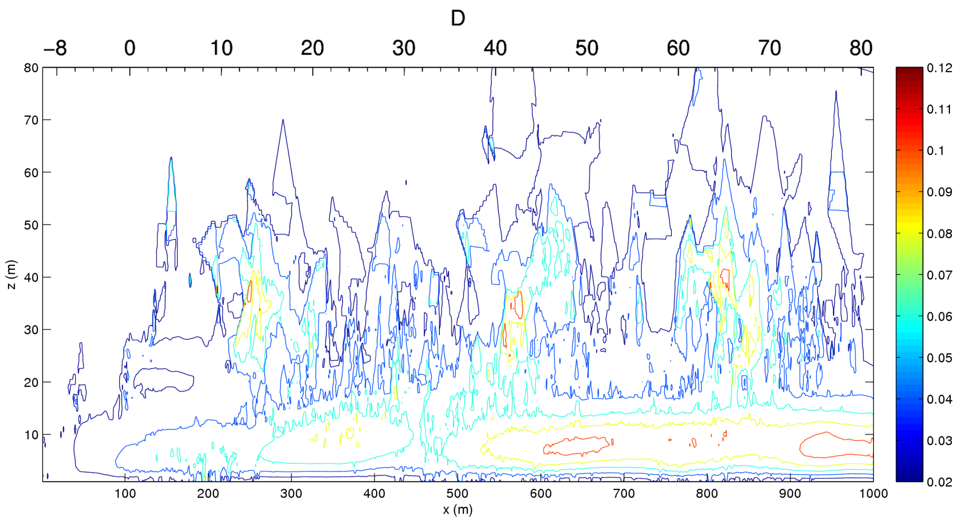


Figure 7.54: Vertical plane contour plot of turbulence intensity

$Ti = 0.4$ which extends for approximately $7D$ downstream. The vertical slice of the turbulence intensity in fig. 7.54 shows a similar level of turbulence, with relatively little for $z > 60\,\text{m}$, but as the vertical velocity gradient increases so does the turbulence, with three large peaks at $z \approx 40\,\text{m}$ and two large patches of turbulence near the sea floor at $x = 500\,\text{m}$ and $x = 900\,\text{m}$. A small trail of turbulence of $Ti = 0.04$ can be seen in the turbine, extending downstream by about $7D$.

Chapter 8

Discussion

8.1. Comparisons between simulations

8.1.1. Vestas V52: turbulent versus non-turbulent inlet conditions

There are several key comparison points between the turbulent and non-turbulent simulations. The inlet turbulence resulted in greater efficiency of the wind turbine, and less recirculation. This becomes apparent when figures 7.10 and 7.21 are superimposed upon each other, as seen in fig. 8.1.

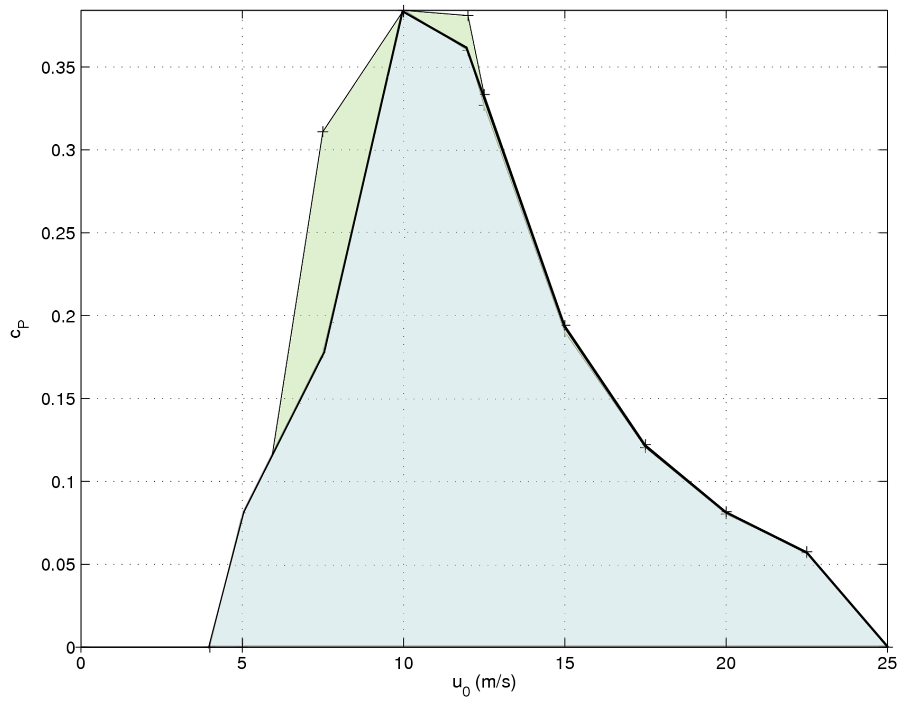


Figure 8.1: Performance curve for V52 with and without inlet turbulence superimposed upon each other. The green area represents the turbulent inlet simulations; the blue, the non-turbulent inlet simulations.

Two things are clear:

1. That a significant amount of additional power is being produced at lower wind speeds in the presence of upstream turbulence.
2. That limiting of the turbine's angular velocity, ω_T, via α is effective at limiting the performance at wind speeds greater than $u_{0,opt}$, irrespective of the presence or absence of upstream turbulence.

A second characteristic difference is the wake recovery. At higher wind speeds, the turbulent inlet case has a shorter wake recovery distance than that of the non-turbulent. Referring to the velocity profiles for $u = 20\,\mathrm{m/s}$ in figures 7.8 and 7.28 we can see that at $x' \geq 3D$, u/u_0 is consistently about 5% higher in the turbulent case.

Clearly, the turbulence generated at the inlet has a dissipative effect on the wake, which has reduced the velocity deficit therein. From this, we can conclude that recirculation has a negative impact on the performance on the turbine, and that turbulence plays a key role in reducing or removing it. That it does not show on the turbulence intensity contour plots suggests that much of the inlet turbulence generated is sub-grid, ie. of a shorter length scale than the minimum length of the finite elements – in this case, 1 metre. This should not be surprising, since the algorithm generating the velocity fluctuations is neither temporally nor spatially coherent, instead only limited by the minimum element length, the simulation time-step size Δt, and the inlet velocity u_0.

Applying coherent turbulence as a boundary condition in LES turbulence models described by Lesieur [45] and Meneveau [54], such as those Fluidity uses, would be a step in the right direction. It hoped that the techniques discussed by Jarrin et al [40] can be applied to the turbine model in future.

8.1.2. Vestas V52 versus Seaflow

Whilst the principles of the horizonal axis wind turbine and the horizontal axis marine turbine are effectively the same, the conditions under which they operate are wholly different. As an attempt to quantify the differences, the

Reynolds number of the V52 wind turbine at a wind speed of $u_{0,opt}$ can be defined as

$$Re_{air} = \frac{\rho_{air} u_{0,opt} L_p}{\mu_{air}} \tag{8.1}$$

Where L_p is the approximate length scale of the problem: in this case the sum of the blade chord lengths, calculated as $L_p = B\pi R_T$.

Simulation	$u_{0,opt}$ (m/s)	ρ (kg m^{-3})	μ (Pa s)	L(m)
Vestas V52	12	1.226	1.8×10^{-5}	3.3
Seaflow	2.7	1027	1.5×10^{-3}	1.7

Table 8.1: List of physical values for Reynolds number calculations

Plugging the values in from table 8.1 we have

$$Re_{air} \approx 2.6 \times 10^6 \tag{8.2}$$

Which is safely within the turbulent regime. Similarly, for the marine turbine simulation in the water channel without bottom drag, we have

$$Re_{sea} \approx 3.2 \times 10^6 \tag{8.3}$$

Given that Re_{sea} is 23% larger than Re_{air}, we can expect some differences in flow characteristics; this is indeed borne out by examination of the velocity deficit contour plots for the V52 turbulent inlet and the Seaflow, at their respective values of $u_{0,opt}$ (figs. 7.7 and 7.33).

If we superimpose the 80% recovery contours for both the V52 and Seaflow in optimum flow conditions as in figure 8.2, we can see that while the V52 produces a significant wake 620 m longer than that of the Seaflow at approximately 380 m, and however measured in turbine diameters (D), it still obeys the 20 diameter rule-of-thumb used by wind farm engineers at a recovery distance $\approx 19D$; this contrasts with the Seaflow turbine wake recovery, which stands at $\approx 35D$. At an initial, cursory glance, this would seem to suggest that a hypothetical marine turbine farm woule be capable of less dense energy

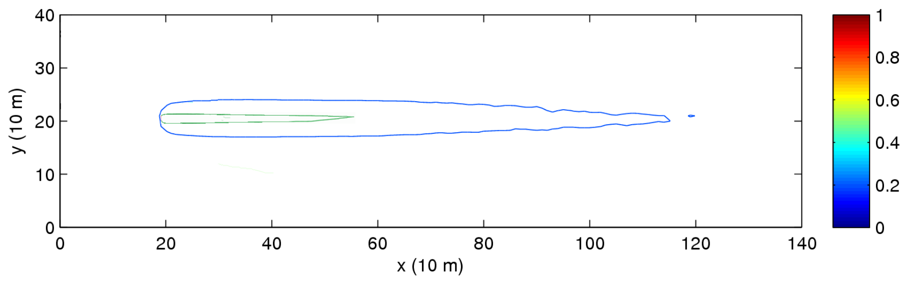


Figure 8.2: Contour plots at 80% wake recovery for the Vestas V52 turbine at $u_0 = 12\,\mathrm{m/s}$ and Seaflow turbine at $u_0 = 2.7\,\mathrm{m/s}$. The Vestas wake is in blue, and the Seaflow wake is in green.

extraction than wind farms of similar rated power output – this does however, neglect the impact that the density of flow medium, ie. air and seawater. Would a marine turbine farm need a similar surface area allocated to that of a wind farm with the same rated power output? An attempt to answer this will be made below.

Due to the seawater's comparatively high density, 838 times that of air under standard conditions, although Seaflow has a diameter of $0.21\,D_{vestas}$ it still manages to produce 0.38 times the power of the V52 under optimum conditions. Therefore, a more representative rule-of-thumb for effective energy extraction density would be

$$\frac{PW_{seaflow}}{PW_{vestas}} = c_{ed}\,\frac{W_{seaflow}}{W_{vestas}} \tag{8.4}$$

Where $PW_{seaflow}$ equals the power extracted under optimum conditions (similarly for the Vestas V52), and $W_{seaflow}$ is the effective wake recovery distance for the Seaflow, beyond which another turbine downstream can operate effectively. c_{ed} is our extraction density co-efficient: if $c_{ed} > 1$, then a farm of Seaflow turbines are more effective at extracting energy, per unit area, than a farm of Vestas turbines; if $c_{ed} < 1$ then it is less effective. Feeding the numbers above in, we find that

$$c_{ed} = 0.93 \tag{8.5}$$

Which puts them almost at equal par, ie. almost the same power output per unit area.

This conclusion does though ignore several important facts. Firstly, the highly variable and irregular nature of wind resource means that wind turbines often will not be operating at peak output; calculations based upon mean wind speeds may not be accurate due to the non-linear relationship between wind speed and power output of wind turbines, as shown in figure 7.10. This contrasts with the flow regime of tidal straits, which can produce approximately regular, sinusodial flows peaking four times daily. Secondly, whilst the V52 is a commercial turbine, Seaflow is but a prototype of which one can assume that it is not the most efficient design. Changing the diameter of the turbine will alter the cross-sectional Reynolds number, which may affect the performance of the turbine: it remains to be seen whether such a change will be beneficial or detrimental. That no recirculation occurs in the Seaflow simulations, as seen in fig. 7.34 suggests that higher values of c_P are possible, however the danger as previously mentioned is of cavitation, a risk naturally not present with wind turbines. Lastly, aside from bathymetric-driven features, the water tunnel experiments neglect aspects of tidal flow which have an important effect on turbine efficiency and wake development, which are discussed in the following section.

8.1.3. Realistic velocity profiles and bottom drag

The characteristic differences introduced along with bottom drag is that of the velocity gradient, and of the consequent turbulence. The velocity gradient induced by the bottom drag does several things:

1. Firstly, it reduces the incoming flow of approximately u_0 by 12%, which means that although the peak flow may be quoted as $u_0 = 4.0\,\mathrm{m/s}$, the turbine is effectively in a tidal stream of $u_0 = 3.5\,\mathrm{m/s}$.

2. Secondly, it generates a great deal of turbulence as seen in figs. 7.53 and 7.54, causing a great deal of mixing between the upper and lower strata

of the flow; this dissipates the wake more effectively.

3. Thirdly, wake recovery is facilitated by the transport of kinetic energy from faster fluid moving around the turbine: as the fluid moving over the turbine moves faster than that under it, the top edge of the wake is accelerated to greater degree than that below. This has the effect of turning the wake down towards the sea floor.

Another feature of the flow is a upward flow before and over the turbine, then the descending flow behind it, as in fig. 7.52. This strongly suggests that, should a free-surface be implemented in a future model channel, the surface would rise above the turbine and fall behind it creating a hydraulic jump, which is not a phenomena that exists in wind turbine wake dynamics. It could be speculated that this conversion of pressure energy to gravitational energy, then finally to kinetic energy downstream, *could* aid recovery of the wake; shortening the distance between turbines in a marine turbine farm and ensuring that $c_{ed} > 1$.

8.2. Comparison with existing literature

8.2.1. With wind turbine theory and experiment

In approach

The turbine model in this thesis was first aimed at accurately modelling wind turbines and far-wake behaviour, since these are relatively well understood in terms of performance and behaviour, and as such will allow for cross-validation. However, in terms of modelling it ignores convential stream-tube, blade-element momentum theory and notions of lift [16] [22], instead favouring a higher level approach. For instance, the effective solidity and flow factor parameterise the complex changes in lift due to blade characteristics and flow conditions to a simple function of the orbital velocity of the turbine; it certainly does not attempt the level of detail that Mikkelsen aims for with actuator line modelling [55]. In these terms, it can be thought of as half-way between single blade blade-element analysis [24] [41] and the linear wake model of Risoe's WAsP, mentioned in Barthelmie [9], although it remains a full fluid dynamic simulation – with just enough detail to give a second-by-second report of perfomance of the turbine the flow field around it, but not enough that modelling a wind farm becomes unfeasible.

Primarily, it was felt that with its origins in analytic methodology [34], current actuator disc theory in computational models such as Sørensen's vorticity-velocity equations [68] [69] were too abstract (as opposed to simply high level), and that by exploiting the iterative, discrete and finite nature of computational methodology, a simpler – or rather, more intuitive– approach could be taken. As an example of this would be the infinitely thin disc in actuator theory, which uses pressure as a boundary since force cannot be applied over infintesemal distance; by turning this into a cylinder, a force can be applied, which can be implemented as a body force in the Navier-Stokes momentum equation. This then makes the technique, indeed the modelling code itself, straightforward to apply or implement in finite-element, finite-difference or finite volume CFD modelling programs.

In results

As for experimental comparisons, Hossain [37] provides an excellent source of velocity and turbulence profiles for both the near and far wakes of microturbines.

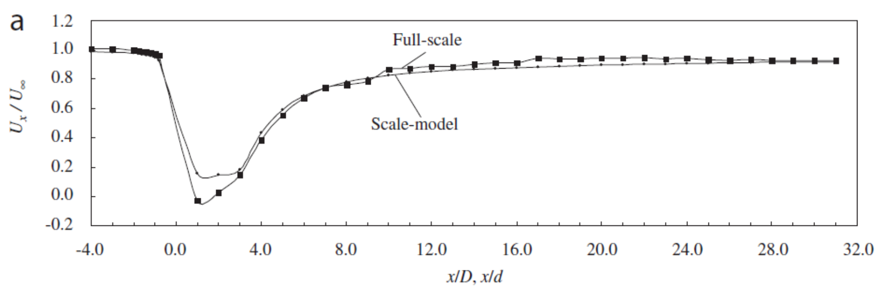


Figure 8.3: Normalised velocity upwind and downwind of micro wind turbine (courtesy of Hossain)

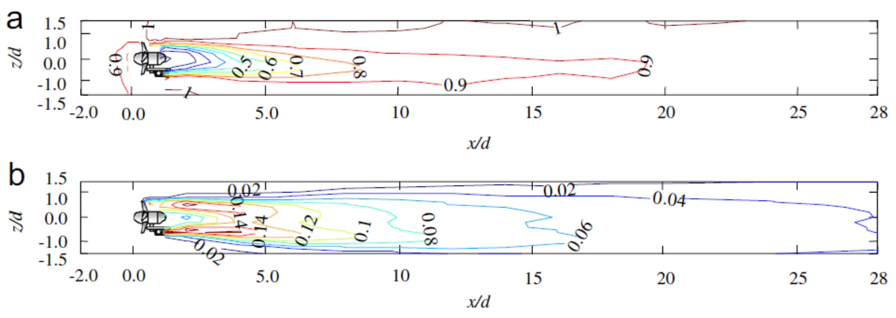


Figure 8.4: Contour plots from micro wind turbine measurements. (a) of u/u_0. (b) of turbulence intensity (courtesy of Hossain)

The velocity profile in fig. 8.3 demonstrates that, by and large, the Vestas V52 simulation follows the same pattern of wake recovery. Moreover, figures 8.4(a) and 8.4(b) show that it also resembles the same wake *structure* in terms of velocity and turbulence. The same twin ridges of turbulence near the blade tips are also shown in Gomez-Elvira [35]; these are present in the results from the thesis model, eg. in fig. 7.9.

Further validating the wake recovery, Magnusson's measurements of the wakes behind horizontal axis turbines at Alsvik [49] demonstrated 80% recovery for at $x' \approx 16D$ for 23 m diameter turbines at hub height (see fig. 8.5). This is close to the distance of $x' = 17D$ for the Vestas model for with $u_0 = 12\,\mathrm{m/s}$ in figure 7.7. Högstrom's wake measurements behind a 2 MW turbine at Näsudden, Sweden [36] in fig. 8.6 show a similar wake structure, but with sharper decay in velocity deficit at distances of 10-15D. This may be partially due to the difference in scale: the Näsudden turbine is 75 m in diameter, over three times the size of the ones at Alsvik. Also, the Näsudden turbine also demonstrates levels of turbulence over twice that of the thesis model (see fig. 8.7); this we know affects wake recovery.

Another experimental validation must be that of the power output. In fig. 8.8, the thesis model without inlet turbulence is superimposed with official figures for low-turbulence conditions, from the Danish Wind Industry Association. Three aspects are immediately noticeable: the slight underperformance of the model at $u_0 < 10\,\mathrm{m/s}$, the overperformance at $u_0 = 12\,\mathrm{m/s}$, and the

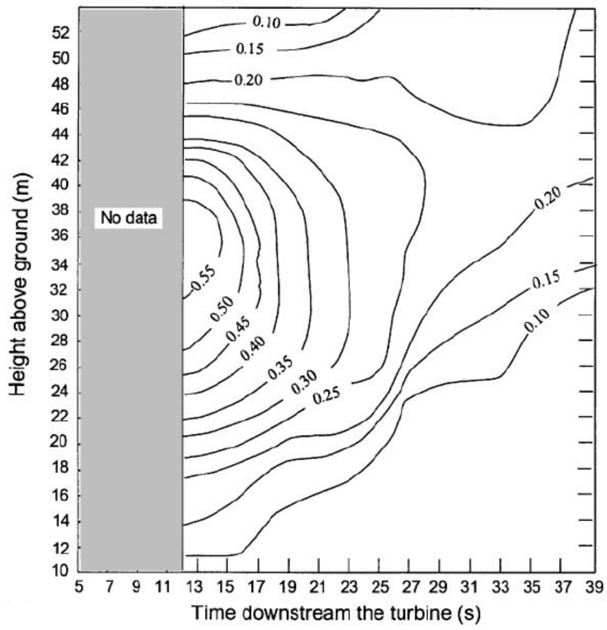


Figure 8.5: Vertical profile of the velocity deficit downwind of a 23 m turbine at Alsvik (courtesy of Magnusson)

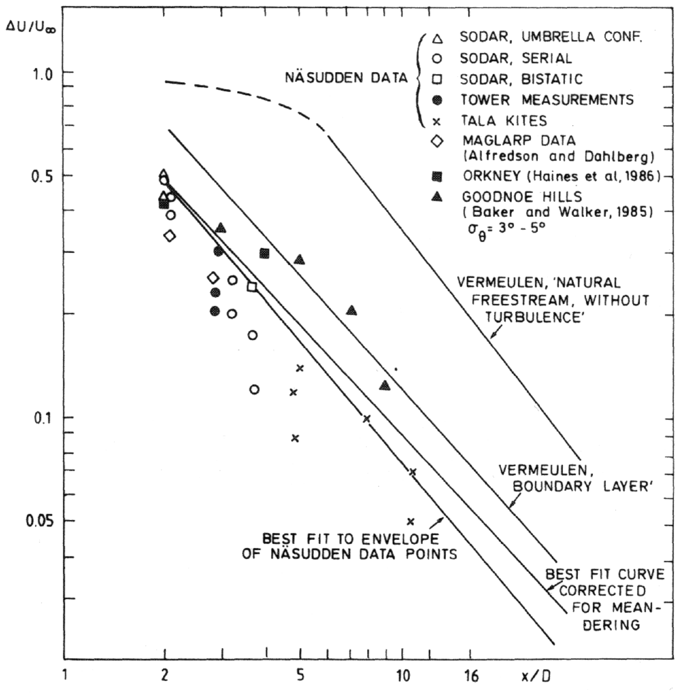


Figure 8.6: Mean relative velocity deficit at hub height behind a 2 MW turbine in Näsudden, Sweden, as a function of distance downwind; other sources are for comparison (courtesy of Högström et al)

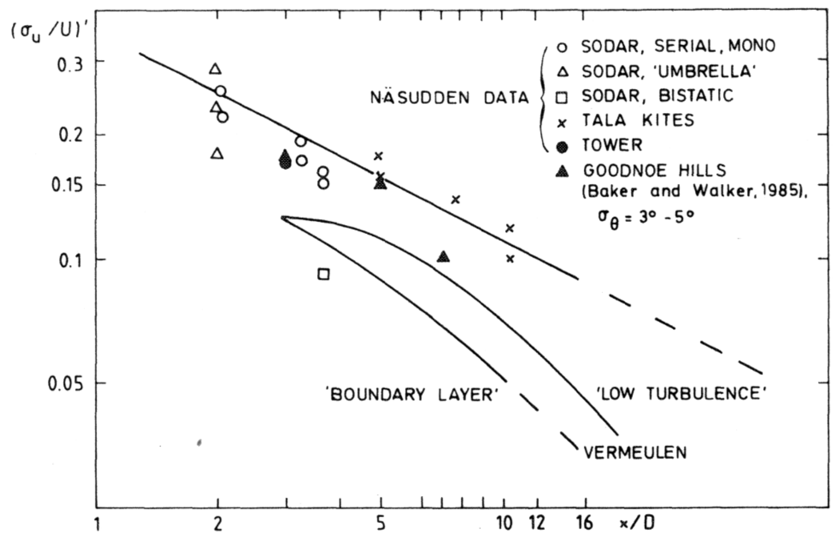


Figure 8.7: Turbulence intensity behind a 2 MW turbine in Näsudden, Sweden, as a function of distance downwind; other sources are for comparison (courtesy of Högström et al)

slanting cutoff in the model for $u_0 > 22.5\,\text{m/s}$ (due to the discrete jumps in u_0). As far as the performance at $u_0 \leq 12\,\text{m/s}$ is concerned, a revisal of the definition of $f_B(\omega_T)$ (see fig. 4.45) is in order for more accurate results; that said, the model's power output does represent the performance of the real turbine with a fair degree of accuracy, over a range of wind speeds.

Comparing the thesis model to the actuator disc model of Thomsen [70], in fig. 8.9 we can see the Thomsen model retains the effect of peak power extraction at $r = \frac{3}{4}\,R_T$ even at $x' = 6D$, whereas the thesis model has smoothed to 'U' shape $x = 2D$. Furthermore, in the former $\min(u) = 0.2\,u_0$, whereas the latter shows some recirculation with $\min(u) = -0.12\,u_0$.

This recirculation can be explained partly by the restriction in the FEM resolution. With the minimum dimension of an element set to 5 m that would mean that the virtual Vestas V52, with a diameter of 52 m, is represented by only 10 elements across, which coupled with Fluidity's smoothing algorithms, may smear out any fine detail. Decreasing the minimum element size would resolve the flow more accurately, allowing tip-induced vorticity to be accurately modelled and thus the turbulent mixing in the wake that it causes. This would

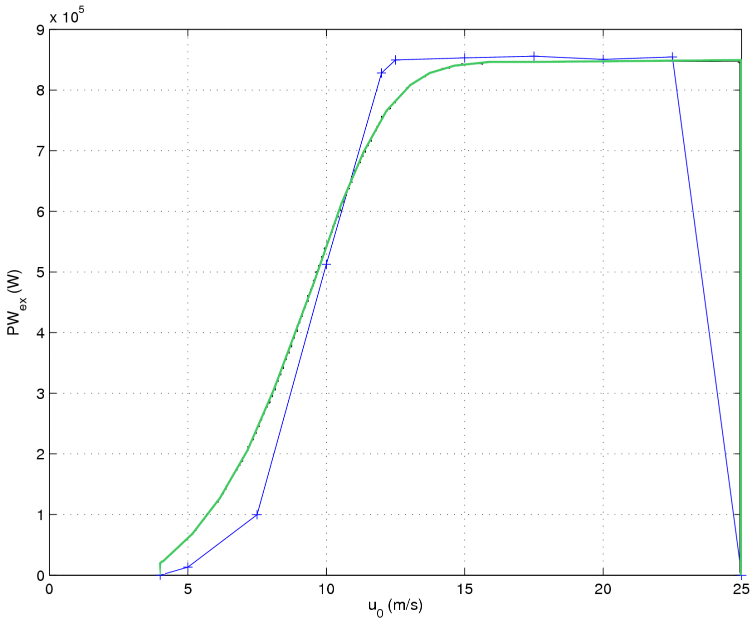


Figure 8.8: Power output as a function of the wind speed for the Vesta V52. The model output is in blue, the experimental measurements in green (courtesy of the Danish Wind Industry Association).

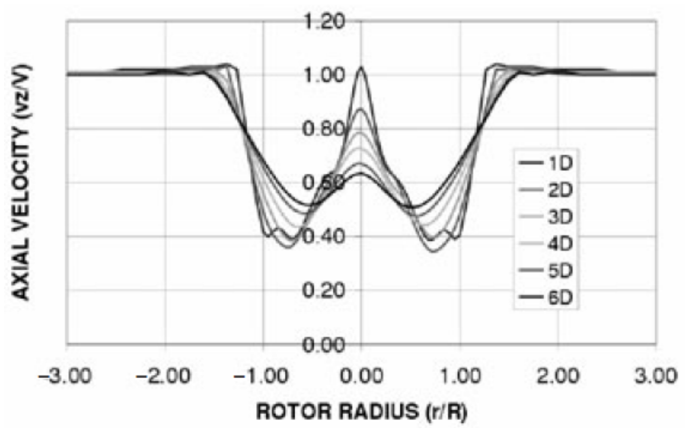


Figure 8.9: Down-wind velocity profiles as a function of r/R (courtesy of Thomsen)

however increase the computational demands and effectively decrease the size of the domain being simulated: note that the Thomsen model simulates to $x' = 6D$ downstream a quarter the length of the wind tunnel domain in this thesis.

The second significant cause of this may be the treatment of the hub. Ideally, the hub should be treated as a solid obstacle with a slip condition, rather than just a momentum sink. This however would require enforcing boundary conditions at the surface of the hub:

$$\underline{u}_{\perp} = 0 \tag{8.6}$$

$$\frac{\partial u_{//,i}}{\partial x_i} = 0 \tag{8.7}$$

Which represent the velocity components of the fluid that are both perpendicular and parallel to the surface of the hub respectively. The problem is that the thesis model by definition does not have any boundary conditions, and so the hub is modelled as a partial momentum sink. Because of this, the wind speed directly downwind of the hub is reduced.

The third suspect is the solidity distribution of the blades. On modern wind turbines, the solidity actually *drops* towards the hub (see photograph in fig. 6.2), whereas in the model it increases until it reaches the hub. If the model followed solidity distribution of the real Vestas, the kinetic energy of the fluid near the hub would be higher, which would feed momentum into the wake, thus reducing or dispersing recirculation eddies. There is nothing preventing an extension to the model to deal with this, since the local solidity is defined as a function of radius, ie. $\beta(r)$.

Regardless of the near-wake recirculation issues, the model closely emulates the actual performance of single turbines and the wake recovery behind them; these were the main concerns in building a model capable of simulating air flow through and around turbines over distances greater than 1 km.

A worthwhile future study may be of how the model's wake structures

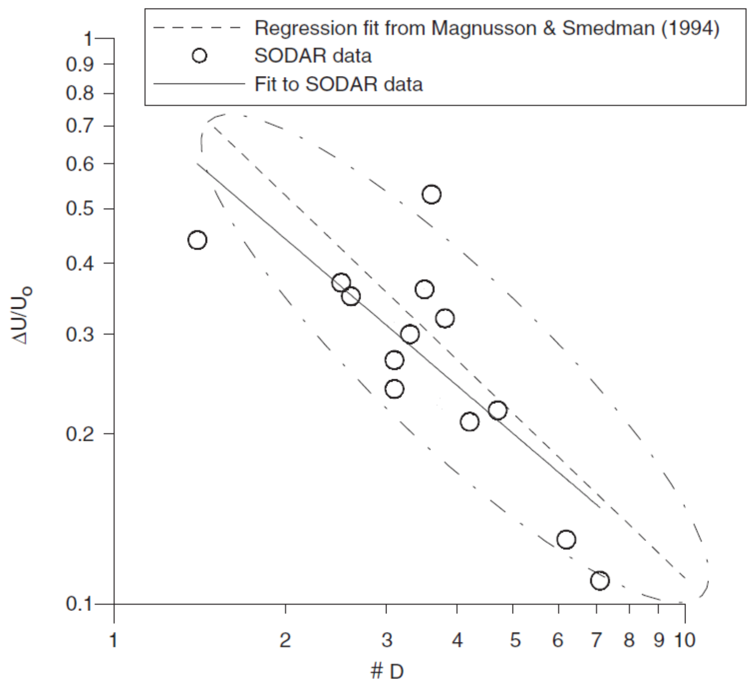


Figure 8.10: SODAR measurements showing distance downwind versus relative velocity deficit at hub height (courtesy of Barthelmie et al)

change in multiple turbine arrangements and more realistic environments. In their paper on wind farm wakes, Barthelmie et al [10] show SODAR measurements of the velocity deficit behind a single wind turbine within the Vindeby offshore wind farm in Denmark (see fig. 8.10). This clearly points to faster wake recovery than in the model: for comparison, see figs. 7.14 and 7.18. To understand why turbulence is so important, we must consider the role that it plays in wake development. Turbulence has a diffusive effect on the wake, aiding the transport of kinetic energy from the outer, faster wake flow to the more stagnant inner layers; therefore, we can expect higher levels of turbulence to equate to shorter wakes. This may be of key importance to understand wake recovery, and thus turbine performance, within large wind farms.

In real wind farms such as Vindeby, there are additional significant contributors to turbulence such as upwind turbine wakes and the sea surface. From the marine turbine model results in section 7.5, we can see that imposing a vertical velocity gradient – necessary for modelling sea surface friction –

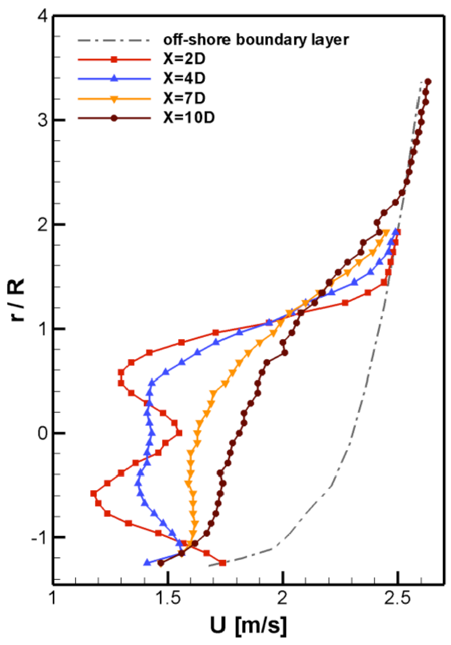


Figure 8.11: Vertical profiles of the horizontal wind velocity component in the wake of a scaled turbine. D is diameters downwind (courtesy of Pascheke and Hancock)

also contributes to wake shortening, and indeed increases levels of turbulence. Pascheke and Hancock [61] show in their scale model that the reduced surface roughness in off-shore environments (in comparison to on-shore) still has a profound effect on the vertical structure of the wake – see fig. 8.11. The scale model was verified with Högstrom's 1988 field data for the wake behind a 2MW wind turbine. If the Vestas V52 simulations were extended to include multiple turbines and friction on the bottom of the wind tunnel, something which the thesis model software is capable of doing, we should expect down-wind turbine wakes to be shortened as a consequence. More recently, Barthelmie has initiated a preliminary investigation along these lines [11].

8.2.2. With marine turbine theory and experiment

Information on the performance of horizontal axis marine current turbines and the wakes that such turbines produce is harder to come by than that of wind turbines. Certainly, the Seaflow model performed to the specifications laid out

[30] [48], however real performance data proved difficult to obtain, perhaps understandably given the sensitive commercial nature of such information.

What we can talk about is aspects of the flow in the marine environment that differentiate it from that of wind turbines. Bryden et al in their discussion of marine power site criteria [13] touch upon the issues raised by flow with a strong vertical velocity gradient, namely restrictions on turbine diameter and depth of the channel (see figure 8.12). It can be seen that the Seaflow model used in section 8.1.3 fits both the depth and diameter criteria.

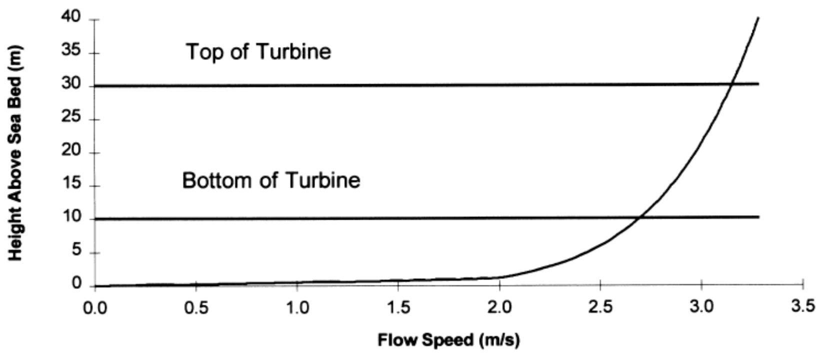


Figure 8.12: Restriction on turbine dimensions in a idealised tidal channel (courtesy of Bryden et al).

Moving on to free surfaces and whether they will have any significant impact on marine turbine performance, it was speculated in the section above that the upwelling before the Seaflow turbine would lead to a hydraulic jump with a free surface. Myers [56] proved that this is can occur in experiment with a $\frac{1}{30}$ scale turbine with a free surface; see figure 8.13.

But what kind of surface disturbance could be expected? The Froude number indicates whether supercritical or subcritical flow can be expected. This is defined [71] as

$$Fr = \frac{u_0}{\sqrt{gL}} \tag{8.8}$$

Where g is the acceleration due to gravity, and L here is the critical length. From Myers, for a rectangular channel this is

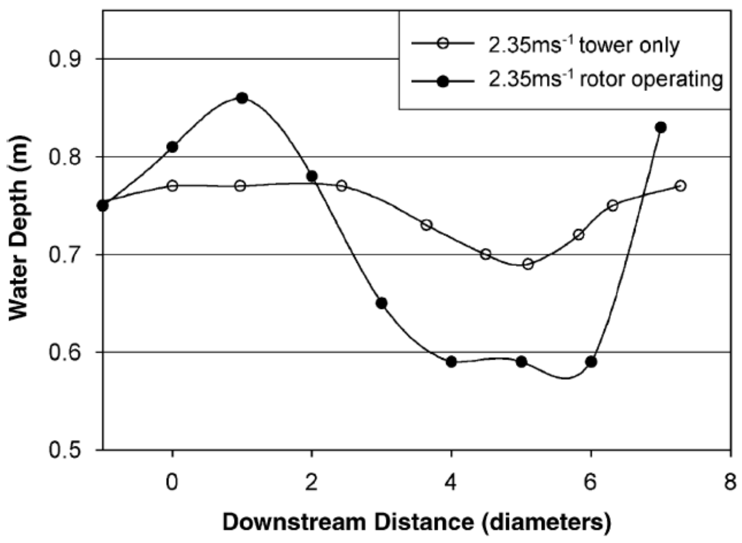


Figure 8.13: Surface elevation downstream of a scaled marine turbine in experiment for $u_0 = 2.35\,\mathrm{m/s}$ (courtesy of Myers et al).

$$L = \left(\frac{Q^2}{gB^2}\right) \tag{8.9}$$

where Q is the mass flow rate, and B the width of the channel. For a 40 m deep channel, over a cross-section 10 m wide in a current of $2.00\,\mathrm{m/s}$ this gives

$$Fr \approx 0.03 \tag{8.10}$$

This suggests that the surface displacement over full-scale marine turbines in tidal straits are likely to be small.

Chapter 9

Conclusions

The main aim of the model in this thesis was to produce an intuitive, high-level model of a horizontal axis turbine, that would be both applicable to wind turbines and marine current turbines. It was designed to behave as a real turbine, so that even though heavily parameterised, once the parameters for a particular model of turbine had been fed in, it would behave as a real turbine would in a variety of flow conditions. It was also designed so that it could, with modest computing resources such as a desktop workstation, simulate at least one wind or marine turbine over a large enough domain to study the far wake. Moreover through its distributed, parallel design, given adequate processing power there is no reason why the model should not scale to the simulation of turbine farms.

9.1. Achievements

From chapter 7, the model has produced similar c_P curves to that of a real wind turbine, the Vestas V52. It has also produced velocity deficit contour plots and velocity profiles that closely resemble that of experimental data. Turbulence plots have mostly agreed with available data, however two problems remain. Firstly, that only the *visible* turbulence is being measured, so that the sub-grid turbulence remains unquantified. Secondly, realistic turbulence needs to be coherent turbulence, which is as yet untested waters in unstructured finite-element CFD modelling; the current algorithm generates deviations in the fluid velocity that are Gaussian white (uncorrelated) noise. This has the unphysical side-effect of being dependent on the simulation time-step size Δt: higher values of u_0 require smaller Δt to maintain a Courant number $Cr \approx 1$ and thus stability. Thus, at higher windspeeds the higher frequences will start to dominate the turbulence spectra. Yet, section 7.4 demonstrated that through comparing turbulent and non-turbulent inlet conditions, sub-grid turbulence was being generated and having an effect – it was clear that the dissipative effective of the additional turbulence viscosity was improving the performance of the Vestas especially at lower wind speeds.

The simulation of the Seaflow marine turbine raised some interesting questions. While an increased wake length was not wholly unexpected due to higher Reynolds numbers, a wake recovery twice that of the Vestas was a surprise, and suggests profound implications for designers of marine turbine farms in the future. As the basic analysis in section 8.1.2 has shown, despite the energy-density of seawater flowing through tidal straits there are practical limits to the extraction density; the number of turbines per square kilometre, if you prefer.

Lastly, this thesis has hopefully vindicated Fluidity's approach to CFD. By using an hr-adaptive, finite-element mesh, Fluidity has allowed the simulation of the complete wake behind both turbines, on a computer no more powerful than a modern standard PC. It has done this by adding and moving element nodes to areas where the velocity gradients are largest, such as the wake - so satisifying the specified error tolerances - and by removing and moving nodes away from areas where the velocity gradients are small. Thus if the adaptive mesh error tolerances and element dimension ranges correctly set, a simulation of 10 minutes in a wind tunnel can take less than a day to run, and yet give useful results.

9.2. Future work

The author's ambitions in this area can be placed into two categories: enhancement of the model, and enhancing the simulation domain of the model.

The foremost candidate for enhancement of the model would be is the turbulence generation algorithm within the turbine. Coherent turbulence requries a memory of the past flow field, which in an unstructured mesh proves difficult. That said, by interpolating to a regular mesh within the turbine volume and storing the points on that mesh for several time steps, spatially and temporally coherent turbulence within the turbine should be possible. Currently, simulation runs spend less than 0.01% of their time within the turbine model routines, so an linear interpolation routine would be feasible.

As for simulation domains – the deployment of the model in this thesis has been for the most part in idealised conditions, necessary for validation purposes. Now that that stage is complete, more realistic domains are desirable to gauge the reaction of the model to such environments. For wind turbines, this would be the introduction of geographic features such as hills and terrain, as well as buildings, trees and obstacles typical of urban environments. A key point to remember is that Fluidity with its oceanographic roots is capable of this already, since the bottom of the model domain is already specified as a depth, albeit a constant one in the simulations in this thesis. However, the depth can can be varied if to represent any convex surface and thus the features listed above. As for marine turbines, the introduction of a free surface *and* bathymetric features such as seamounts, ridges, depressions and coastlines would represent an exciting step forward in study of marine turbines in their operating environment.

Lastly, concerning both marine and wind turbines, a study of the downstream effects on a secondary turbines seems an avenue worth exploring. The model has been implemented in a wholly parallel fashion, and is capable of supporting multiple turbines running simultaneously – prelimary test runs have been made with this in mind.

Appendix A

Test CFD solvers

A.1. An implementation of a SIMPLE algorithm

To investigate the SIMPLE algorithm, a C program was built based upon it. The following is a listing of the main parts of the program, with explanations of the mathematics behind it throughout the listing.

It did not work completely correctly, perhaps because of the explicit solution of δp, which despite an application of a variety of relaxation parameter values proved rather unstable; or the diffusion terms for y axis-based contributions were incorrect. At this point, it was becoming plain that there were certain shortcomings in finite difference models which would greatly affect their applicability to tidal modelling over the scales and at level of detail that would be required in future.

A.1.1. Model overview

The model was two dimensional, modelling what should be evolve to become horizontal steady flow, but with extensions could model unsteady flow. Initially, a staggered grid approach was implemented, then a centred-difference scheme as it was more straightforward to implement; Fletcher [27] discusses the advantages of using these.

The top boundary is a solid, moving boundary, which has a fixed velocity along the x axis, while the bottom boundary is solid and stationary; ie. both Dirichlet. The left and right boundaries are Von Neumann.

A.1.2. Program structure

The program essentially follows the steps below:

1. **Initialisation** – set the initial values for the main global variables such as $\underline{u}_n$, $\underline{u}^*$, p, etc.
2. **Display results** – after N time-steps, print $\underline{u}_n$ and diagnostic information
3. **Boundary conditions** – set Dirichlet and Boundary conditions for $\underline{u}_n$.
4. **Calculation** – ∇p, $\underline{u}^*$, δp, etc.
5. **Loop from second step** – if the maximum number of iterations has not been reached.

A.1.3. Parameters

Model parameters that could be set were:

- Reynolds number Re
- Grid spacing via Δx and Δy
- The number of points on the grid in both x and y-axis directions
- Time-step size Δt

A.1.4. The main routine and entry point

This is the first routine the program calls when run. It also contains all the major variables, including:

- **Vector **u Vector **u** – a 2D array of C structures containing x and y components of the velocity at time step n; eg. $\underline{u}_n$
- **Vector **ug** – $\underline{u}^*$
- **Vector **uc** – $\underline{u}^c$
- **Vector **gradP** – ∇p
- **Vector **a_u** – contains $a^u_{j,k}$ and $a^v_{j,k}$ $\forall(j,k)$

- **FlowCell **fc** – a 2D array of structures, the most important component of which is the pressure p^n.

```
#include <stdio.h>
#include {\textquotedbl}structs.h>
#include {\textquotedbl}externs.h>
#include {\textquotedbl}defines.h>

GlobalVariables globalVars;

int main(int argc, char **argv)
{
 int error=0;
 IntVector size;
 Vector **u, **ug, **uc, **a_u;
 Boundary *boundary;
 FlowCell **fc;
 Vector **gradP;

 double **deltaP;
 double t;
 int n=0, nflag;

 initGlobalVariables();
 size=globalVars.size;
```

This part allocates memory to the main arrays. *createArray()*, *createVectorArray()* and *createFlowCellArray()* are routines contained in an external C module.

```
 u = createVectorArray(size);
 ug = createVectorArray(size);
 boundary = createBoundaryArray(size, NUM_DIMENSIONS);


 ug = createVectorArray(globalVars.size);
 uc = createVectorArray(size);
 a_u = createVectorArray(size);


 fc = createFlowCellArray(size);


 gradP = createVectorArray(size);
 deltaP = (double **)createArray(size, sizeof(double));


 error=initialiseEverythingElse(u, boundary, fc);


 /* If there was no bother with the initialisation, go on right ahead.
*/


 if(!error)
 {
```

This is the start of the main loop. t is incremented in steps of *globalVars.delta_t* until it is larger than *globalVars.maxTime*.

The condition if(n%1==0) will output results every iteration; changing the 1 to 100, say, would output every 100 iterations.

calculateEverything() calls the real meat of the program; the calculation module. It passes all variables to the central routine in the SIMPLE algorithm module, which are then passed to further subroutines for specific calculations. Note – only u and fc will be carried forward from the last iteration; the other variables are here for program structuring purposes, and to allow the output of diagnostic information in the main routine.

```
 /* Run simulation until time limit reached, or error occurs */
 for(t=0.0; t<globalVars.maxTime; t+=globalVars.delta_t, n++)
 {
   /* Every ten loops, output some data */
   if(n\%1==0)
   {
   printf("outputResults()\n");
   outputResults(u, size, t, n);
   }
   printf({"time: \%.6f\n", t);

   error=calculateEverything(u, ug, uc, a_u,
  boundary,
      fc, gradP, deltaP);

   if(error) break;
 }

}

  /* Exit stuff */

if(error)
{
fprintf(stderr, "< Error caught! > \n");
exit(1);
} else {
printf("*** Completed ok\n");
exit(0);
}
 }
```

A.1.5. Initialisation

This module contains the routines which set the initial values for all the important variables. initialiseEverything() sets up the initial $\underline{u}^n$ values, as well as calling routines to set the boundary conditions and the pressure values.

initGlobalVariables() initialises the parameters, such as Δx, Δy and Δt, etc.

```
#include <structs.h>
#include <externs.h>
#include <defines.h>

/* Entry point for initialise.c .
 Calls other dependant functions --
 except for initGlobalVariables, which is called seperately from main.c
*/

int initialiseEverythingElse(Vector **vel,
    Boundary *boundary,
    FlowCell **fc)
{
 initVelocitiesAndPressure(vel, fc);
 initBoundaries(boundary);
 return 0;
}

int initGlobalVariables()
{
 globalVars.delta.x=globalVars.delta.y
=globalVars.delta.z=0.5;

 globalVars.size.x=12;
 globalVars.size.y=12;
 globalVars.size.z=0;

 globalVars.fcSize.x=globalVars.size.x;
 globalVars.fcSize.y=globalVars.size.y;
 globalVars.fcSize.z=globalVars.size.z;

 globalVars.Re=0.5;
 globalVars.delta_t=1;
 globalVars.maxTime=100;

 return 0;
}
```

This routine fixes the initial velocities and pressure: both pressure and velocity are initially set to zero. Boundary conditions will diffuse flow into the model as the iterations progress.

```
/* Set-up the velocity and pressure fields */
int initVelocitiesAndPressure(Vector **vel, FlowCell **fc)
{
 int i, j;
 IntVector size=globalVars.size;
 IntVector fcSize;

 /* First the velocity field */
 vel=(Vector **)createVectorArray(size);

 for(i=0; i<size.x; i++)
 for(j=0; j<size.y; j++)
 vel[i][j].x=vel[i][j].y=vel[i][j].z=0;
 /* Now the flow cells */
 fcSize.x=size.x-1;
 fcSize.y=size.y-1;
 fcSize.z=0;

 fc=(FlowCell **)createFlowCellArray(fcSize);

 for(i=0; i<fcSize.x; i++)
for(j=0; j<fcSize.y; j++)
  {
fc[i][j].pressure=0;
fc[i][j].cellType=CELL_EMPTY;
  }

 return 0;
}
```

This routine sets the boundary conditions. BD_DEPENDENT indicates a Von Neumann boundary; BD_CONSTANT, Dirichlet.

```
/* Routine for setting boundary-specific values */

int initBoundaries(Boundary *boundary)
{
 int i, j;
 IntVector size=globalVars.size;

 /* set X boundaries
 Bottom of model is solid and so fixed, top assumes no vel gradient. */
 boundary[LEFT_BD].boundaryType=BD_DEPENDENT;
 /* commented out for just now, Poiseuille flow is just wrong...
 setVelocityProfile(&boundary[LEFT_BD], FL_POISEUILLE, size, 1.0); */
 boundary[RIGHT_BD].boundaryType=BD_DEPENDENT;

 /* set Y boundaries.
 Left of model is constant, right of model is calculated from internal
values */

 boundary[BOTTOM_BD].boundaryType=BD_CONSTANT;

 for(i=0; i<size.x; i++)
 {
 boundary[BOTTOM_BD].u[i].x
 =boundary[BOTTOM_BD].u[i].y
 =boundary[BOTTOM\_BD].u[i].z
 =0;

 }

 boundary[TOP_BD].boundaryType=BD_CONSTANT;

 /* set fluid moving horizontally along top */

 for(i=0; i<size.x; i++)
 {
 boundary[TOP_BD].u[i].x=0.5;

 /* dirty hack, will make configurable later */
 boundary[TOP_BD].u[i].y = boundary[TOP_BD].u[i].z = 0;
 }
 return 0;

}
```

This routine is commented out; it was never used.

```
/* Set up Poiseuille flow at left boundary */
/*

int setVelocityProfile(Boundary *leftBoundary,
    int flowType, double flowParam)
{
 int i, j;
 double Re=globalVars.Re;
 Vector delta=globalVars.delta;
 Vector size=globalVars.size;

 switch(flowType)
 {
 case FL_POISEUILLE:
 default:
 break;
 }

 return 0;

}

*/
```

A.1.6. The calculation routines

```
#include <stdio.h>
#include <math.h>
#include <structs.h>
#include <externs.h>
#include <defines.h>

int calculateEverything(Vector **u_n,
   Vector **ug,
   Vector **uc,
   Vector **a_u,
   Boundary *boundary,
   FlowCell **fc,
   Vector **gradP,
   double **deltaP)
{
 int i, j, error=0;
 IntVector size=globalVars.size;

 enforceBoundaryConditions(u_n, boundary, size);
 printVectorArray(u_n, size);
 calcPressureGradient(fc, gradP, size);

 printf({"calcDeltaP()\n");
 calcDeltaP(deltaP, uc, a_u, size);

 copyVectorArray(u_n, ug, size);

 printf({"calcGuessVelocity()\n");
 calcGuessVelocity(ug, u_n, boundary, gradP, a_u, fc, size);

 printf("calcVelocityCorrection()\n");
 calcVelocityCorrection(uc, deltaP, a_u, size);

 printf("updatePressure()\n");
 updatePressure(fc, deltaP, size);

 printf("updateVelocities()\n");
 updateVelocities(u_n, ug, uc,size);

 /* If an error has occurred, return error code */
 if(error)
 return 1;
 else
 return 0;

}
```

enforceBoundaryConditions() copies values to or from the boundary in the $\underline{u}^n$ array, depending on the boundary type:

1. BD_DEPENDENT – values will be copied outward from the outermost-but-one line of velocity values to the boundary, thus enforcing Von Neuman boundary conditions.
2. BD_CONSTANT – fixed velocity values will be written the boundary and adjacent inner rows or columns, for Dirichlet boundary conditions.

```
int enforceBoundaryConditions(Vector **vel, Boundary *boundary,
IntVector size)
{
 int i, j, n;
```

```
/* X boundaries first */

for(n=0; n<2; n++)
for(j=0; j<size.y; j++)
{
switch(boundary[n].boundaryType)
{
case BD_DEPENDENT:
  if(n==LEFT_BD)
  {
  /* left boundary */
  vel[0][j]=vel[2][j];
  vel[1][j]=vel[2][j];

  } else {
  /* right boundary */
  vel[size.x-1][j]=vel[size.x-3][j];
  vel[size.x-2][j]=vel[size.x-3][j];
  }
  break;

case BD_CONSTANT:
  if(n==LEFT_BD)
  {
  /* left boundary */
  vel[0][j]=boundary[n].u[j];
  vel[1][j]=boundary[n].u[j];

  } else {
  /* right boundary */
  vel[size.x-1][j]=boundary[n].u[j];
  vel[size.x-2][j]=boundary[n].u[j];

  }

  break;

default:
  break;

}
}

/* Now Y boundaries ... */

for(n=2; n<4; n++)
for(i=0; i<size.x; i++)
{
switch(boundary[n].boundaryType)
{
case BD_DEPENDENT:
  if(n==BOTTOM_BD)
  {
  /* bottom boundary */

  vel[i][0]=vel[i][2];
  vel[i][1]=vel[i][2];

  } else {
  /* top boundary */
  vel[i][size.y-1]=vel[i][size.y-3];
  vel[i][size.y-2]=vel[i][size.y-3];

  }
  break;

 case BD_CONSTANT:
  if(n==BOTTOM_BD)
  {
  /* bottom boundary */

  vel[i][0]=boundary[n].u[i];
  vel[i][1]=boundary[n].u[i];
  } else {
  /* top boundary */

  vel[i][size.y-1]=boundary[n].u[i];
  vel[i][size.y-2]=boundary[n].u[i];
  }
  break;

 default:
  break;

 }

 }

 return 0;
}
```

This function calculates the coefficient matricies for a^u and a^v. This is called from within *calcGuessVelocity(),* twice.

```
int calcVelocityCoeffs(Vector **a_u, Vector **u, IntVector size)
{
 int i, j;

 double twoDyRe=globalVars.delta.y/(globalVars.Re*globalVars.delta.x);
 double twoDxRe=globalVars.delta.x/(globalVars.Re*globalVars.delta.y);
 double quarterDx=globalVars.delta.x/4.0;
 double quarterDy=globalVars.delta.y/4.0;

 for(i=1; i<size.x-1; i++)
 for(j=1; j<size.x-1; j++)
 {

 a_u[i][j].x =
   + quarterDy * ( u[i][j].x + u[i+1][j].x - u[i-1][j].x - u[i][j].x )
   + twoDyRe
   + quarterDx * ( u[i][j].y + u[i+1][j].y - u[i-1][j].y - u[i+1][j-1].y)
   + twoDxRe;

 a_u[i][j].y =
   + quarterDy * ( u[i][j].x + u[i+1][j].x - u[i-1][j].x - u[i-1][j+1].x)
   + twoDyRe
   + quarterDx * ( u[i][j].y + u[i][j+1].y - u[i][j-1].y - u[i][j].y )
   + twoDxRe;
 }

 return 0;
}
```

Using a four-point first-order star difference scheme, calculate the pressure gradient ∇p:

```
int calcPressureGradient(FlowCell **fc, Vector **gradP, IntVector size)
{
 int i, j;
 Vector delta=globalVars.delta;
 for(i=0; i<size.x; i++)
 {
 for(j=0; j<size.y; j++)
 {
 /* Get x-component of pressure gradient */

 if(i==0)
   gradP[i][j].x=(fc[i+1][j].pressure - fc[i][j].pressure)/delta.x;
   else {
   if(i==size.x-1)
   {
  gradP[i][j].x=(fc[i][j].pressure - fc[i-1][j].pressure)/delta.x;
   } else {
  gradP[i][j].x=(fc[i+1][j].pressure - fc[i][j].pressure)/delta.x;
   }
   }

   /* Get y-component of pressure gradient */
   if(j==0)
   gradP[i][j].y=(fc[i][j+1].pressure - fc[i][j].pressure)/delta.y;
   else {
   if(j==size.y-1)
   gradP[i][j].y=(fc[i][j].pressure - fc[i][j-1].pressure)/delta.y;
   else
   gradP[i][j].y=(fc[i][j+1].pressure - fc[i][j].pressure)/delta.y;
   }
}
 }


 return 0;


}
```

calcGuessVelocity() calculates $\underline{u}^*$. Essentially, this solves

$$(\frac{\Delta x \Delta y}{\Delta t} + a^u_{j,k})u^*_{j,k} + \sum a^u_{nb}u^*_{nb} = -b^u - \Delta y(p^n_{j+1,k} - p^n_{j,k}) \tag{A.1}$$

and the similar equation for $v^*_{j,k}$, for $u^*_{j,k}$ and $v^*_{j,k}$ respectively by treating them as a simultaneous set of equations. To make this easier, the equations are split up into x and y contributions using the ADI (alternating direction implicit) method in Fletcher to turn these equations into a sparse tridiagonal matrix, which can then be solved by the Thomas algorithm. The ADI method essentially treats the first part, the x sweep, as a half-time step, and so all values of Δt are changed to $0.5\Delta t$; the velocities are then solved along the x-axis as the tridagonal matrix via Thomas. The solutions for this intermediate equation, $\underline{u}'_{j,k}$, are then solved along the y-axis via Thomas again to produce $\underline{u}^*_{j,k}$.

```
/* calcGuessVelocity():
 Calculates new 'guess velocity' (ie.
 not corrected for continuity law), using
 a combination of the Thomas algorithm
 and ADI (alternating direction implicit) */


int calcGuessVelocity(Vector **ug,
 Vector **u_n,
 Boundary *boundary,
 Vector **gradP,
 Vector **a_u,
 FlowCell **fc,
 IntVector size)
{
 int i, j;

 Vector delta=globalVars.delta;

 double delta_t=globalVars.delta_t;
 double Re=globalVars.Re;
 double twoDxDyOverDt=2.0*delta.x*delta.y/delta_t;

 int maxDimen = size.x>size.y?size.x:size.y;

 Vector **ugDash = createVectorArray(size);

 double *uxTemp = (double *)malloc(sizeof(double) * maxDimen);
 double *uyTemp = (double *)malloc(sizeof(double) * maxDimen);

 double *ax = (double *)malloc(sizeof(double) * maxDimen);
 double *ay = (double *)malloc(sizeof(double) * maxDimen);

 double *bx = (double *)malloc(sizeof(double) * maxDimen);
 double *by = (double *)malloc(sizeof(double) * maxDimen);

 double *cx = (double *)malloc(sizeof(double) * maxDimen);
 double *cy = (double *)malloc(sizeof(double) * maxDimen);
 double *dx = (double *)malloc(sizeof(double) * maxDimen);
 double *dy = (double *)malloc(sizeof(double) * maxDimen);

 calcVelocityCoeffs(a_u, u_n, size);

 /* X sweeps first (ADI algorithm) */
 for(j=2; j<size.x-2; j++)
 {
 /* prepare variables for Thomas algorithm */
 for(i=2; i<size.x-2; i++)
 {
   uxTemp[i-2] = u_n[i][j].x;
   uyTemp[i-2] = u_n[i][j].y;

   ax[i-2] = ( delta.y / 4.0)*( u_n[i][j].x + u_n[i+1][j].x ) -
delta.y/(Re * delta.x);

   bx[i-2] = twoDxDyOverDt + a_u[i][j].x;

   cx[i-2] = (-delta.y / 4.0) * ( u_n[i-1][j].x + u_n[i][j].x ) -
delta.y/(Re * delta.x);

   dx[i-2] = - twoDxDyOverDt * u_n[i][j].x - gradP[i+1][j].x * delta.y;

   ay[i-2] = ( delta.y / 4.0)*( u_n[i][j].x + u_n[i][j+1].x ) -
delta.y/(Re * delta.x);

   by[i-2] = twoDxDyOverDt + a_u[i][j].y;

   cy[i-2] = (-delta.y / 4.0) * ( u_n[i-1][j].x + u_n[i-1][j+1].x ) -
delta.y/(Re * delta.x);
```

```
  dy[i-2] = - twoDxDyOverDt * u_n[i][j].y - gradP[i][j+1].y * delta.x;

}

/* Thomas algorithm now */
thomasAlgorithm(uxTemp, ax, bx, cx, dx, size.x-4);
thomasAlgorithm(uyTemp, ay, by, cy, dy, size.y-4);

for(i=2; i<size.x-2; i++)
{
  ugDash[i][j].x = uxTemp[i-2];
  ugDash[i][j].y = uyTemp[i-2];
}

}

enforceBoundaryConditions(ugDash, boundary, size);
calcVelocityCoeffs(a_u, ug, size);

/* Y sweep now */

for(i=2; i<size.x-2; i++)
{
/* Prepare for Thomas */
for(j=2; j<size.y-2; j++)
{
  uxTemp[j-2]=ugDash[i][j].x;
  uyTemp[j-2]=ugDash[i][j].y;

  ax[j-2] = ( delta.x / 4.0)
```

```
   * ( ugDash[i][j].y + ugDash[i+1][j].y ) - delta.x/(Re * delta.y);

   bx[j-2] = twoDxDyOverDt + a_u[i][j].x;

   cx[j-2] = (-delta.x / 4.0) *
   ( ugDash[i][j-1].y + ugDash[i+1][j-1].y ) - delta.x/(Re * delta.y);

   dx[j-2] = - twoDxDyOverDt * ugDash[i][j].x - gradP[i+1][j].x *
delta.y;

   ay[j-2] = ( delta.x / 4.0)
   * ( ugDash[i][j].y + ugDash[i][j+1].y ) - delta.x/(Re * delta.y);

   by[j-2] = twoDxDyOverDt + a_u[i][j].y;

   cy[j-2] = (-delta.x / 4.0)
   * ( ugDash[i][j-1].y + ugDash[i][j].y ) - delta.x/(Re * delta.y);

   dy[j-2] = - twoDxDyOverDt * ugDash[i][j].y - gradP[i][j+1].y *
delta.x;

 }

 /* Thomas algorithm now */

 thomasAlgorithm(uxTemp, ax, bx, cx, dx, size.x-4);
 thomasAlgorithm(uyTemp, ay, by, cy, dy, size.y-4);

 for(=2; j<size.y-2; j++)
 {
   ug[i][j].x = uxTemp[j-2];
   ug[i][j].y = uyTemp[j-2];
 }
 }

 deleteArray((void **)ugDash, size);

 free(dy);
 free(dx);

 free(cy);
 free(cx);

 free(by);
 free(bx);

 free(ay);
 free(ax);

 free(uyTemp);
 free(uxTemp);

 return 0;
}
```

thomasAlgorithm() employs the Thomas Algorithm to solve sparse tridiagonal matrices, such as those created from a series of points governed by three-point finite difference equations.

```
/* thomasAlgorithm():
 Solves sparse matrices. */

int thomasAlgorithm(double *u, double *a, double *b, double *c, double
*d, int maxN)
{
 int n;

 double *c_dash = (double *)malloc(sizeof(double) * maxN);
 double *d_dash = (double *)malloc(sizeof(double) * maxN);

 /* sweep forward */

 c_dash[0] = c[0]/b[0];
 d_dash[0] = d[0]/b[0];

 for(n=1; n<maxN; n++)
 {
 c_dash[n] = c[n] /
 ( b[n] - a[0]*c_dash[n-1] );
 d_dash[n] = ( d[n] - a[n]*d_dash[n-1] ) /
 ( b[n] - a[n]*c_dash[n-1] );
 }

 /* now sweep back, solving for u */

 u[maxN-1] = d_dash[maxN-1];

 for(n=maxN-2; n>=0; n--)
 u[n] = d_dash[n] - u[n+1]*c_dash[n];

 free(c_dash);
 free(d_dash);

 return 0;
```

```
}
```

calcDeltaP() uses an explicit FTCS scheme to solve for δp. This was, on reflection, perhaps a bad choice: even with relaxation parameters explicit methods such as this can prove unstable, and may need many iterations to converge for lower values of those relaxation parameters. Perhaps an implicit scheme would have been better, employing the combination of the ADI method and Thomas algorithm for quasi-simultaneous solution in a similar fashion to that of $\underline{u}^{*}_{j,k}$: this would have required the construction of a Δt-like parameter for the intermediate x and y sweeps.

```
int calcDeltaP(double **dp,
   Vector **ug,
   Vector **a_u,
   IntVector size)
{
 int i, j;
 int n=0;
 int convergence=0;
 Vector E;

 double **oldDp=(double **)createArray(size, sizeof(double));
 double **bp=(double **)createArray(size, sizeof(double));

 Vector **d=createVectorArray(size);
 Vector delta=globalVars.delta;

 double delta_t=globalVars.delta_t;

 /* Keep the coefficents up to date */
 calcVelocityCoeffs(a_u, ug, size);
```

```
for(i=1; i<size.x-1; i++)
for(j=1; j<size.y-1; j++)
{
bp[i][j] =
  ( ug[i-1][j].x-ug[i][j].x )*delta.y
  + ( ug[i][j-1].y-ug[i][j].y )*delta.x;

E.x = delta_t * a_u[i][j].x / (delta.x*delta.y);
E.y = delta_t * a_u[i][j].y / (delta.x*delta.y);

d[i][j].x = E.x * delta.y/( (1+E.x)*a_u[i][j].x );
d[i][j].y = E.y * delta.x/( (1+E.y)*a_u[i][j].y );
}

/* I went for an iterative and explicit solution here. */

/*
while(n++<MAX_CONVERGENCE_LOOPS
| | convergence==0)
*/

while(n++<100) // Quick hack until I write a convergence testing routine
{

/* make copy of old dp */
copyArray((void **)dp, (void **)oldDp, size, sizeof(double));

for(i=1; i<size.x-1; i++)
for(j=1; j<size.y-1; j++)
{
  dp[i][j] =
   (
   ( oldDp[i+1][j] * d[i][j].x * delta.y )
   + ( oldDp[i-1][j] * d[i-1][j].x * delta.y )
   + ( oldDp[i][j+1] * d[i][j].y * delta.x )
   + ( oldDp[i][j-1] * d[i][j-1].y * delta.x )
   + bp[i][j]
   ) / (
    (d[i][j].x+d[i-1][j].x)*delta.y
    + (d[i][j].y+d[i][j-1].y)*delta.x
    );
 }

 convergence=checkForConvergence(dp, oldDp, size);
 }

 deleteArray((void **)oldDp, size);
 deleteArray((void **)d, size);
 deleteArray((void **)bp, size);

 return 0;

}
```

This stub function was never used.

```
/* This doesn't do anything yet, but converge... */

int checkForConvergence(double **dp, double **oldDp, IntVector size)
{
 return 1;
}
```

calcVelocityCorrection() calculates $\underline{u}^c_{j,k}$ from equations

$$u^c_{j,k} = d_{j,k}(\delta p_{j,k} - \delta p_{j+1,k}) \tag{A.2}$$

and

$$v^c_{j,k} = d_{j,k}(\delta p_{j,k} - \delta p_{j,k+1}) \tag{A.3}$$

```
int calcVelocityCorrection(Vector **uc,
     double **dp,
     Vector **a_u,
     IntVector size)
{
 int i, j;

 Vector E;
 Vector d;
 Vector delta=globalVars.delta;

 double delta_t=globalVars.delta_t;

 /* loop for u_c. Yes I should've cached the d term,
 but you know what C's structure is like, ay? */

 for(i=1; i<size.x-1; i++)
 for(j=1; j<size.y-1; j++)
 {
 E.x = delta_t * a_u[i][j].x / (delta.x*delta.y);
 E.y = delta_t * a_u[i][j].y / (delta.x*delta.y);

 d.x = E.x * delta.y/( (1+E.x)*a_u[i][j].x );
 d.y = E.y * delta.x/( (1+E.y)*a_u[i][j].y );
 uc[i][j].x = d.x * ( dp[i][j]-dp[i+1][j] );
 uc[i][j].y = d.y * ( dp[i][j]-dp[i][j+1] );
 }

 return 0;

}
```

These are simple routines for updating pressure and velocity arrays.

```
int updatePressure(FlowCell **fc, double **deltaP, IntVector size)
{
 int i, j;

 for(i=1; i<size.x-1; i++)
 for(j=1; j<size.y-1; j++)
 fc[i][j].pressure+=deltaP[i][j];

 return 0;

}

int updateVelocities(Vector **u, Vector **ug, Vector **uc, IntVector size)
{
 int i, j;

 for(i=1; i<size.x-1; i++)
 for(j=1; j<size.y-1; j++)
 {
 u[i][j].x = ug[i][j].x + uc[i][j].x;
 u[i][j].y = ug[i][j].y + uc[i][j].y;
 }

 return 0;

}
```

A.2. A simple triangular element streamline solver

The following is a dissection of a finite element fluid dynamics solver adapted from T.J. Chung and written in Fortran. Given Neumann and Dirichlet boundary conditions, it calculates the streamline of steady flow round a cylinder using triangular elements. Due to symmetry, only a quarter of the area is simulated – a quarter of the cylinder, and the upper half of the upstream flow. The upper boundary is taken to be a moving plate of constant velocity.

A.2.1. Program structure

There are several steps of the program's execution. They are as follows:

1. **Initialisation** – read in node coordinates and element node lists from pre-existing files.
2. **Coefficient matrices calculation** – calculate local coefficient matrix per node, then global coefficient matrix.
3. **Determination of boundary conditions** – read in boundary streamline values, calculate input vectors for each node accordingly.
4. **Linear equations solution** – solve system of linear equations to calculate unknown streamline values.
5. **Displaying results** – print streamlines and horizontal velocites on right-hand boundary.

A.2.2. List of variables and parameters

Parameters

There parameters are set at compile time. In the example program, there are 10 nodes, 10 elements and 8 boundary nodes.

- nde – number of nodes in simulation
- nel – number of elements in simulation
- nbc – number of boundary nodes

Variables

There are the most important variables in the program; the use of others should be apparent.

- $xnode[nde]$, $ynode[nde]$ – x, y coordinates of nodes
- $nenn[nel, 3]$ – list of node numbers for each element
- $anm[3, 3]$ – the local coefficient matrix, used for each node.
- $sk[nde, nde]$ – the global coefficient matrix
- $mode[nbc]$ - the type of each of the boundary nodes: a value of 0 denotes Neumann, while 1 denotes Dirichlet
- $node[nbc]$ – an array of nodes that are boundaries
- $val[nbc]$ – the value of the streamline on the boundaries
- $r[nde]$ – the input vector for each node
- $psi[nde]$ – The value of the streamline at each node

A.2.3. Program Listing and Description

Initialisation

This chunk of code declares all the main arrays, for the large part by dimension only, as their type is declared implicitly as Fortran allows (a to i for integers, j to z for real numbers). $sk(nde, nde)$ is declared as double precision for compatibility with the $invertMatrix()$ subroutine

```
c  Flow round a cylinder, triangular finite elements
c  Flow round a cylinder, triangular finite elements
c  From T.J.Chung, p336 Sect B-1
 program CylinderFlow

c  nde = no. nodes
c  nel = no. elements
c  nbc = nr of Dirichlet bound conds (with psi=0)
c  nenn = table of node no's by element
c  anm = coeff of matrix for an element
c  r = input vector
c  sk = global coeff matrix
c  xnode = x coord for global node
c  ynode = y coord for global node

 parameter (nel=10, nde=10, nbc=8)
 parameter (nvn=2)

c  WARNING -- all nodes along the vertical line above the crest
c  of the cylinder should be numbered consecutively from top to bottom
c  in order to be compatible with velocity calculations used in this
c  program

 common /trng/ anm(3,3)
 common /geom/ xnode(nde), ynode(nde), nenn(nel,3)


 real*8 sk(nde, nde)
 dimension r(nde), psi(nde), mode(nbc), node(nbc), val(nbc), jc(nde)
 logical finished(nde), err


c  clear global matrix


 do i=1, nde
 do j=1, nde
  sk(i,j) = 0.0
  end do
 end do
```

This code below reads in node co-ordinates, and write them to the screen: dat/node_coords.txt must exist beforehand.

```
c  read coordinates
 open(unit=10, status='old', file='dat/node_coords.txt')


 read (10,501) (xnode(i), ynode(i), i=1, nde)
 close(10)


 write (6,502)
 write (6,503) (i, xnode(i), ynode(i), i=1, nde)
```

Then we read in the node numbers for each pair of triangular elements.

```
c  read element node no's


 open(unit=11, status='old', file='dat/element\_numbers.txt')


 read  (11,504) ((nenn(i,j), j=1,3), i=1, nel)
```

```
close(11)


write (6,505)
write (6,506) (i, (nenn(i,j), j=1,3), i=1, nel)
```

Coefficient matrices calculation

This section requires explanation of the finite element technique used. Considering the streamfunction only within one element only, and assuming that it varies linearly within an element, the finite element approximation of the streamline is

$$\psi = \Phi_N \psi_N \tag{A.4}$$

where $(1 \leq N \leq 3)$ for each node in the element, and the basis functions given as

$$\Phi_N = a_N + b_N x + c_N y \tag{A.5}$$

and

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} = \begin{bmatrix} (x_2y_3 - x_3y_2)/2A & (x_3y_1 - x_1y_3)/2A & (x_1y_2 - x_2y_1)/2A \\ (y_2 - y_3)/2A & (y_3 - y_1)/2A & (y_1 - y_2)/2A \\ (x_3 - x_2)/2A & (x_1 - x_3)/2A & (x_2 - x_1)/2A \end{bmatrix} \tag{A.6}$$

where x_i, y_i (or $xnode(i)$ and $ynode(i)$ from the program) are the i^{th} node's coordinates, and A is the area of the element, given by

$$A = \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix} \tag{A.7}$$

The local finite element equation is described by

$$A_{NM}\psi_M = F_N \tag{A.8}$$

Since

$$A_{NM} = \int_{\Omega} \Phi_{N,i}\Phi_{M,i} d\Omega \tag{A.9}$$

(Chung, eqn. 5-10)

The local coefficient matrix can be written as

$$\begin{aligned} A_{NM} &= A \begin{bmatrix} b_1^2 + c_1^2 & b_2b_1 + c_2c_1 & b_3b_1 + c_3c_1 \\ b_2b_1 + c_2c_1 & b_2^2c_2^2 & b_3b_2 + c_3c_2 \\ b_3b_1 + c_3c_1 & b_3b_2 + c_3c_2 & b_3^2c_3^2 \end{bmatrix} \\ &= anm(3,3) \end{aligned} \tag{A.10}$$

in the program A_{NM} is calculated in the routine $solveElement()$.

This local coefficient matrix exists for each element, so eventually there will be nel 3x3 matrices to be assembled into the global coefficient matrix, each denoted $(A_{NM})_j$ where $1 \leq j \leq nel$.

The global finite element equation is given by

$$A_{\alpha\beta}\psi_\beta = F_\alpha \tag{A.11}$$

and $A_{\alpha\beta}$ and F_α are related to A_{NM} and F_N accordingly by

$$A_{\alpha\beta} = \sum_{e=1}^{nel} A_{NM}^{(e)} \Delta_{N\alpha}^{(e)} \Delta_{M\beta}^{(e)} \tag{A.12}$$

and

$$F_\alpha = \sum_{\Gamma} F_N^{(e)} \Delta_{N\alpha}^{(e)} \tag{A.13}$$

where $\Delta_{Ni} = 1$ if local node N coincides with global node i, and is 0 otherwise.

The program segment below calls solveElement() for each element which returns that element's A_{NM}, of which parts (according to $\Delta_{N\alpha}^{(e)}\Delta_{M\beta}^{(e)}$, ie. in the inner loops for j and k) are added to the global coefficient matrix, $A_{\alpha\beta}$ / $sk()$.

```
c  compute coeff matrix for each element
 do i=1, nel
 call solveElement(i)

c  assemble globally

 do j=1, 3
 jj = nenn(i,j)

 do k=1,3
 kk = nenn(i,k)
 sk(jj,kk) = sk(jj,kk) + anm(j,k)
 end do
 end do
 end do
```

Determination of boundary conditions

Below, the code reads in boundary streamline values, setting the input vector F_α(in the program known as $r\,(i)$) initially to zero.

```
c  Read Dirichlet boundary conditions information

 write(6, 107)
 open(unit=12, status='old', file='dat/boundary\_conds.txt')

 do i=1, nbc
 read  (12, 105) mode(i), node(i), val(i)
 write (6, 108) mode(i), node(i), val(i)
 end do

 close(12)
 do i=1, nde
 r(i) = 0.0
 end do
```

Now we calculate F_α -

- if $mode(i) \neq 0$ then boundary conditions are Dirichlet (program label 84 to 82)
- if $mode(i) = 0$ then boundary conditions are Neumann (lines of code before 84)

```
c  Introduce boundary conditions

 do 82 i=1, nbc
 nx = node(i)

 if(mode(i).ne.0) go to 84

 do 85 j=1, nde
 sk(nx,j) = 0.0
 sk(j,nx) = 0.0
 85   continue

 sk(nx,nx) = 1.0
 go to 82

c  compute input vector

 84  continue

  do 86 k=1, nde
```

```
86  r(k) = r(k) - sk(k, nx) * val(i)


do 87 k=1, nde
sk(k,nx) = 0.0
sk(nx,k) = 0.0


87  continue


sk(nx,nx) = 1.0
r(nx) = val(i)


82  continue
```

Linear equations solution

This part inverts the global coefficient matrix $A_{\alpha\beta}$ in the equation $A_{\alpha\beta}\psi_\beta = F_\alpha$ so we get

$$\psi_\beta = A^{-1}_{\alpha\beta}F_\alpha \tag{A.14}$$

and solves this for the streamline, ie. for the i^{th} streamline

$$\psi_i = \begin{bmatrix} z_{i1} & z_{i2} & ... & z_{iE} \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ ... \\ F_E \end{bmatrix} \tag{A.15}$$

where z_{ij} is an element of A^{-1}

```
write(6, *) 'Global coefficient matrix'


do xx=1, nde
write(6, *) (sk(xx,yy), yy=1, nde)
end do
```

```
c  Invert global matrix and multiply by vector

 call invertMatrix(sk, nde, sk, err, finished)

 if(err) then
 write(6,*) '1 Matrix is singular'
 stop
 end if

 do i=1, nde
 psi(i) = 0.0

 do j=1, nde
 psi(i) = psi(i) + sk(i,j)*r(j)
 end do
 end do
```

A.2.4. Displaying results

Calculating the horizontal velocity

This part prints results, and calculates the horizontal velocity v_x at the right-ward boundary. Taken from the definition of the streamline, we have

$$v_x = \frac{\partial \psi}{\partial y} \tag{A.16}$$

If we multiply by dy and integrate thus

$$\int_a^b d\psi = \int_a^b v_x dy \tag{A.17}$$

we get

$$v_x = \frac{\psi_b - \psi_a}{y_b - y_a} \tag{A.18}$$

Allowing us to easily calculate the horizontal velocity component; the code below uses the last equation to do this.

```
c  Write nodal streamline function data

 write(6, 998)
 write(6, 999) (i, psi(i), i=1, nde)

c  Velocity profile on crest of cylinder

 nnn = nde - nvn+1
 mmm = nde - 1

 write(6, 590)

 do i=nnn, mmm
 j = i+1

 rd = ynode(i) - ynode(j)
 rm = (ynode(i) + ynode(j)) / 2.0

 sid = psi(i) - psi(j)
 vel = sid/rd

 write(6, 591) rm, vel
 end do

 stop
```

Upon a successful run, the program terminates at this point.

Formatting Rules

Various formatting rules used to read data in from the text files, or to print variables out. Of particular note are:

1. **501** – two floating point numbers, for the node coordinates
2. **504** – six integers in a row; this lists the node numbers for each element pair
3. **107** – two integers (boundary type and node number), plus float (streamfunction value)

```
c  Formatting rules...

 105  format(2i5, f10.0)

 107  format('1  Dirichlet boundaryconditions', //6x,'mode', 5x,
 &  'node', 5x, 'value'/)

 108  format(' ', 2i9, f10.2)

 501  format(2f10.2)
 502  format('1node  x-coord y-coord')
 503  format(' ', i3, 2f12.5)
 504  format(6i5)
 505  format('0element node numbers',/, '  elmt  1  2  3')
 506  format(4i5)

c  Angus formatting rules

 510  format(3f10.5)
 590  format(1h1, ' velocity profile on crest of cylinder'///
 &  6x, 'y', 16x, 'vel'//)
 591  format(f10.5, f20.8/ )


 998  format(1h1, 'nodal streamline function data'//)
 999  format(i5, f15.8)


 end


c  End of compilation: no diagnostics
```

A.2.5. Subroutines

Local Coefficient Matrix Solver

Coefficient matrix solver, which converts local coefficient matrix into global coefficient matrix: this is described in the local coefficient matrix in Chung [18], section 2.3.2.

```
c  ------------------ subroutine section ------------------------
c  Subroutine to solve element coefficient matrix
 subroutine solveElement(n)


c  x = x coord for local node
c  y = y coord for local node
 parameter (nel=10, nde=10, nbc=8)


 common /trng/ anm(3,3)
 common /geom/ xnode(nde), ynode(nde), nenn(nel,3)


 real x(3), y(3)
 real b(3), c(3)


c  local node x and y coords
```

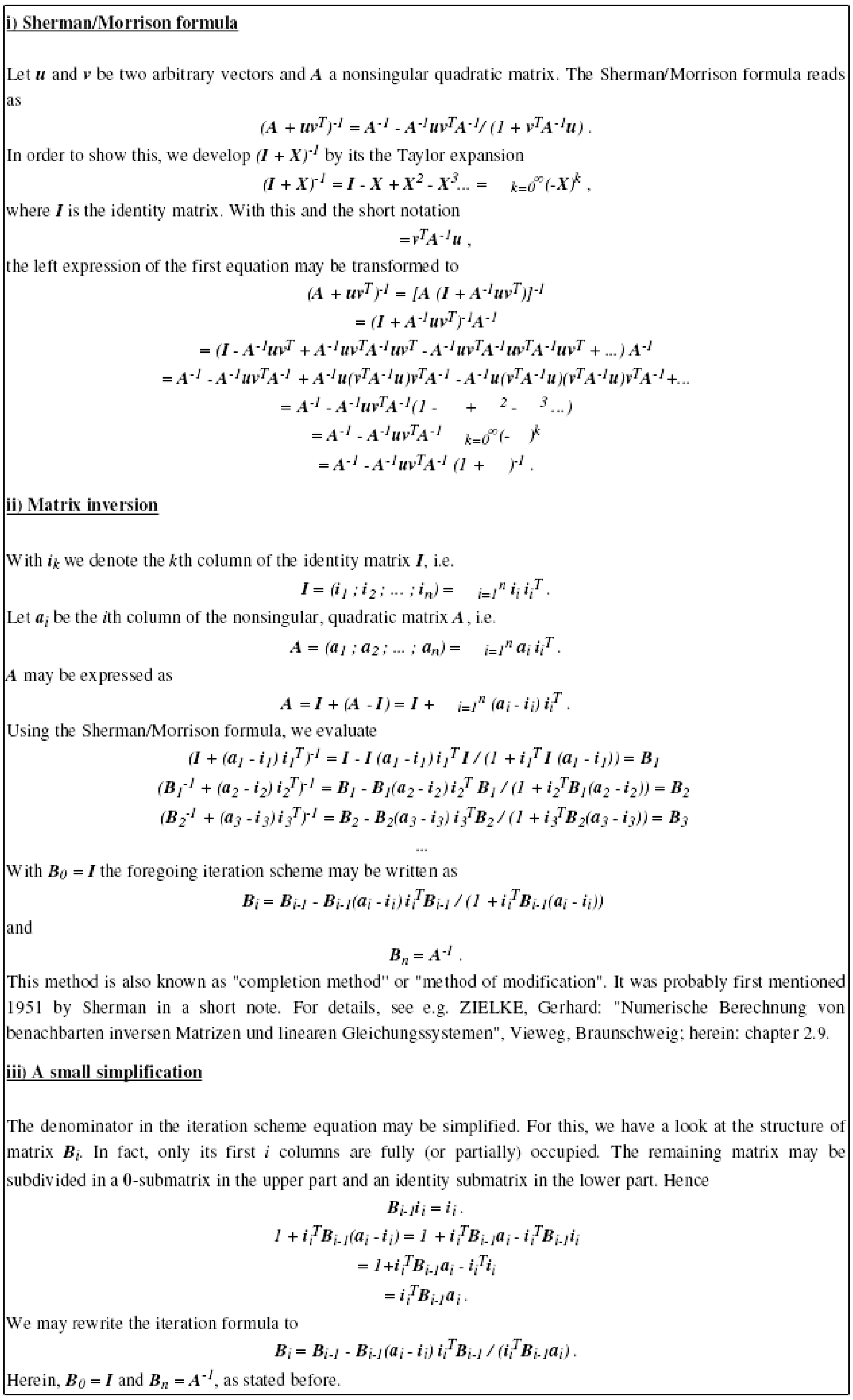
```
 do l=1, 3
 i = nenn(n, l)

 x(l) = xnode(i)
 y(l) = ynode(i)

 end do

c  compute 2 x area

 a2 = x(2)*y(3) + x(3)*y(1) + x(1)*y(2)
 \&  - x(2)*y(1) - x(3)*y(2) - x(1)*y(3)

 a = a2 / 2.0

 b(1) = (y(2) - y(3))/a2
 b(2) = (y(3) - y(1))/a2
 b(3) = (y(1) - y(2))/a2

 c(1) = (x(3) - x(2))/a2
 c(2) = (x(1) - x(3))/a2
 c(3) = (x(2) - x(1))/a2

 do i=1,3
 do j=1,3
 anm(i,j) = (b(j)*b(i) + c(j)*c(i)) * a
 end do
 end do

 return
 end
```

Matrix Inversion Routine

The subroutine *invertMatrix*() uses the Sherman-Morrison formula to iteratively calculate the inversion of a matrix. The following (figure A.2.5) has been copied verbatim from material by Albert Meyers from Ruhr-University Bochum.

This is the Fortran implementation of that algorithm.

```
c  Invert matrix using Sherman/Morrison formula
 subroutine invertMatrix(a,n,b,err,finished)


c  Inversion b=a**(-1), where a is a (n*n) matrix. On error
c  (a singular) err=.true.; additional storage finished needed.
c  a and b may share memory (a will be overwritten by its inverse)


 integer  n
 logical  finished(n), err
 real*8  a(n,n), b(n,n)


c  internal parameter
 integer  count,i,i0,i1,j,k
 real*8  bij,bki,eps,xn,xna


c  eps: if pivot is less than this value --> error
 parameter (eps=1d-37)


c  Mark all columns not finished


 do 10 i=1,n
 finished(i)=.false.
 10  continue
```



i) Sherman/Morrison formula

Let $\boldsymbol{u}$ and $\boldsymbol{v}$ be two arbitrary vectors and $\boldsymbol{A}$ a nonsingular quadratic matrix. The Sherman/Morrison formula reads as

$$(A + uv^T)^{-1} = A^{-1} - A^{-1}uv^TA^{-1} / (1 + v^TA^{-1}u) \,.$$

In order to show this, we develop $(I + X)^{-1}$ by its the Taylor expansion

$$(I + X)^{-1} = I - X + X^2 - X^3 ... = \quad {}_{k=0}{}^{\infty}(-X)^k \,,$$

where $\boldsymbol{I}$ is the identity matrix. With this and the short notation

$$= v^TA^{-1}u \,,$$

the left expression of the first equation may be transformed to

$$\begin{aligned}(A + uv^T)^{-1} &= [A\,(I + A^{-1}uv^T)]^{-1}\\ &= (I + A^{-1}uv^T)^{-1}A^{-1}\\ &= (I - A^{-1}uv^T + A^{-1}uv^TA^{-1}uv^T - A^{-1}uv^TA^{-1}uv^TA^{-1}uv^T + ...)\,A^{-1}\\ &= A^{-1} - A^{-1}uv^TA^{-1} + A^{-1}u(v^TA^{-1}u)v^TA^{-1} - A^{-1}u(v^TA^{-1}u)(v^TA^{-1}u)v^TA^{-1} + ...\\ &= A^{-1} - A^{-1}uv^TA^{-1}(1 - \quad + \quad {}^2 - \quad {}^3 ...)\\ &= A^{-1} - A^{-1}uv^TA^{-1} \quad {}_{k=0}{}^{\infty}(- \quad)^k\\ &= A^{-1} - A^{-1}uv^TA^{-1}\,(1 + \quad)^{-1} \,.\end{aligned}$$

ii) Matrix inversion

With i_k we denote the kth column of the identity matrix $\boldsymbol{I}$, i.e.

$$I = (i_1 \,;\, i_2 \,;\, ... \,;\, i_n) = \quad {}_{i=1}{}^n\, i_i\, i_i^T \,.$$

Let a_i be the ith column of the nonsingular, quadratic matrix $\boldsymbol{A}$, i.e.

$$A = (a_1 \,;\, a_2 \,;\, ... \,;\, a_n) = \quad {}_{i=1}{}^n\, a_i\, i_i^T \,.$$

$\boldsymbol{A}$ may be expressed as

$$A = I + (A - I) = I + \quad {}_{i=1}{}^n\,(a_i - i_i)\, i_i^T \,.$$

Using the Sherman/Morrison formula, we evaluate

$$(I + (a_1 - i_1)\, i_1^T)^{-1} = I - I\,(a_1 - i_1)\, i_1^T I \,/\, (1 + i_1^T I\,(a_1 - i_1)) = B_1$$

$$(B_1^{-1} + (a_2 - i_2)\, i_2^T)^{-1} = B_1 - B_1(a_2 - i_2)\, i_2^T B_1 \,/\, (1 + i_2^T B_1(a_2 - i_2)) = B_2$$

$$(B_2^{-1} + (a_3 - i_3)\, i_3^T)^{-1} = B_2 - B_2(a_3 - i_3)\, i_3^T B_2 \,/\, (1 + i_3^T B_2(a_3 - i_3)) = B_3$$

...

With $B_0 = I$ the foregoing iteration scheme may be written as

$$B_i = B_{i-1} - B_{i-1}(a_i - i_i)\, i_i^T B_{i-1} \,/\, (1 + i_i^T B_{i-1}(a_i - i_i))$$

and

$$B_n = A^{-1} \,.$$

This method is also known as "completion method" or "method of modification". It was probably first mentioned 1951 by Sherman in a short note. For details, see e.g. ZIELKE, Gerhard: "Numerische Berechnung von benachbarten inversen Matrizen und linearen Gleichungssystemen", Vieweg, Braunschweig; herein: chapter 2.9.

iii) A small simplification

The denominator in the iteration scheme equation may be simplified. For this, we have a look at the structure of matrix B_i. In fact, only its first i columns are fully (or partially) occupied. The remaining matrix may be subdivided in a $\boldsymbol{0}$-submatrix in the upper part and an identity submatrix in the lower part. Hence

$$B_{i-1} i_i = i_i \,.$$

$$\begin{aligned}1 + i_i^T B_{i-1}(a_i - i_i) &= 1 + i_i^T B_{i-1} a_i - i_i^T B_{i-1} i_i\\ &= 1 + i_i^T B_{i-1} a_i - i_i^T i_i\\ &= i_i^T B_{i-1} a_i \,.\end{aligned}$$

We may rewrite the iteration formula to

$$B_i = B_{i-1} - B_{i-1}(a_i - i_i)\, i_i^T B_{i-1} \,/\, (i_i^T B_{i-1} a_i) \,.$$

Herein, $B_0 = I$ and $B_n = A^{-1}$, as stated before.

Figure A.1: The Sherman-Morrison formula used to invert a matrix (courtesy of Albert Meyers).

```
c  i0 first, i1 last non finished column
 i0=1
 i1=n

c  main loop
 do 100 count=1,n

c  find biggest denominator
 i=i0
 xn=0d0

 do 30 j=i0,i1
 if (.not.finished(j)) then
 xna=a(j,j)

 do 20 k=1,n
 if (finished(k)) xna=xna+b(j,k)*a(k,j)

 20    continue

 if (abs(xna).gt.abs(xn)) then
 xn=xna
 i=j
 endif
 endif

 30  continue

 err=abs(xn).lt.eps
 if (err) return

c  Evaluate ith column
 do 50 k=1,n
 if (finished(k)) then
 bki=0d0
 else
 bki=a(k,i)
 endif

 if (k.eq.i) bki=bki-1d0

 do 40 j=1,n
 if (finished(j)) bki=bki+b(k,j)*a(j,i)
 40   continue

 b(k,i)=-bki/xn

 50   continue

 b(i,i)=1d0+b(i,i)

c  Evaluate remaining columns

 do 70 j=1,n
 if (finished(j)) then
 bij=b(i,j)

 do 60 k=1,n
 b(k,j)=b(k,j)+bij*b(k,i)
 60   continue
```

```
b(i,j)=b(i,j)-bij
endif
70  continue

finished(i)=.true.

80  if (finished(i0)) then
i0=i0+1
goto 80
endif

90  if (finished(i1)) then
i1=i1-1
goto 90
endif

100  continue

end
```

Bibliography

[1] Marine current turbines ltd. website (www.marineturbines.com).

[2] Turbine power calculator at Danish Wind Industry Association website (www.windpower.org/en/tour/wres/pow).

[3] Vestas website (www.vestas.com).

[4] Visualization toolkit website (www.vtk.org).

[5] Fortran compiler benchmarks. Technical report, Polyhedron Software, 2007.

[6] P.H. Alfredsson, F.H. Bark, and J.A. Dahlberg. Some properties of the wake behind horizontal axis wind turbines. *International Symposium on Wind Energy Systems*, 3:469–484, 1980.

[7] ANSYS. *CFX R11 Manual.*

[8] Vestas Wind Systems A/S. V52-850 kw, the turbine that goes anywhere. Vestas brochure, 2008.

[9] R. J. Barthelmie, L. Folkerts, G. Larsen, K. Rados, S.C. Pryor, S. Frandsen, B. Lange, and G. Schepers. Comparison of wake model simulations with offshore wind turbine wake profiles measured by sodar. *Journal of atmospheric and oceanic technology*, 23:888–901, 2006.

[10] Rebecca Barthelmie, Hans Bergstrom, Wolfgang Schlez, Kostas Rados, Bernhard Lange, Per Volund, Soren Neckelmann, Soren Mogensen, Gerard Schepers, Luuk Folkerts, and Mikael Magnusson. Endow (Efficient

Development of Offshore Wind Farms): Modelling wake and boundary layer interactions. *Wind Energy Conversion and Management*, 7:225–245, 2004.

[11] R.J. Barthelmie, E. Politis, J. Prospathopoulos, S.T. Frandsen, O. Rathmann, K. Hansen, S.P. van der Pijl, J.G. Schepers, K. Rados, D. Cabezn, W. Schlez, J. Phillips, and A. Neubert. Power losses due to wakes in large wind farms. *World Renewable Energy Congress*, 10:2114–2119, 2008.

[12] John Bartholomew. *Advanced Atlas of Modern Geography (3rd Edition)*. McGraw-Hill, 1957.

[13] I. Bryden, S. Naik, P. Fraenkel, and C.R. Bullen. Matching tidal current plants to local flow conditions. *Energy*, 23:669–709, 1998.

[14] Ian Bryden. Tidal energy. *Encyclopedia of Energy*, 6:130–150, 2004.

[15] Ian Bryden and S.J. Couch. ME1 - marine energy extraction: tidal resource analysis. *Renewable Energy*, 31:133–139, 2006.

[16] T. Burton, D.J. Sharpe, N. Jenkins, and E. Bossanyi. *Wind Energy Handbook*. Wiley, 2001.

[17] Donata Melaku Canu, Cosimo Solidoro, and Georg Umgiesser. Modelling the responses of the Lagoon of Venice ecosystem to variations in physical forcings. *Ecological Modelling*, 170:265–289, 2003.

[18] T.J. Chung. *Finite Element Analysis in Fluid Dynamics*. McGraw-Hill, 1978.

[19] S.J. Couch. *Numerical modelling of tidal flows around headlands and islands*. PhD thesis, University of Strathclyde, 2001.

[20] A. Crespo, J. Hernandez, and S. Frandsen. Survey of modelling methods for wind turbine wakes and wind farms. *Wind Energy*, 2:1–24, 1999.

[21] S. Danilov, G. Kivman, and J. Schroter. A finite-element ocean model: principles and evaluation. *Ocean Modelling*, 6 (2):125–150, 2004.

[22] Otto de Vries. On the theory of the horizontal-axis wind turbine. *Annual Review of Fluid Mechanics*, 15:77–96, 1983.

[23] M. Drago and L. Lovenitti. Sigma coordinates hydrodynamic numerical model for coastal and ocean three-dimensional circulation. *Ocean Engineering*, 27, 2000.

[24] Zhaohui Du and M.S. Selig. The effect of rotation on the boundary layer of a wind turbine blade. *Renewable Energy*, 20:167–181, 2000.

[25] T. Edwards. SB 04-09 wave and tidal power- harnessing the energy of the sea. Technical report, Scottish Parliament, 2004.

[26] C.A.J. Fletcher. *Computational Techniques for Fluid Dynamics*, volume 2. Springer Series in Computational Physics, 1991.

[27] C.A.J Fletcher. *Computational Techniques for Fluid Dynamics*, volume 1. Springer Series in Computational Physics, 1991.

[28] R. Ford, C.C. Pain, M.D. Piggott, A.J.H. Goddard, C.R.E. de Oliveira, and A.P. Umpleby. A nonhydrostatic finite-element model for three-dimensional stratified oceanic flows. Part I: Model formulation. *Monthly Weather Review*, 132:28162831, 2004.

[29] R. Ford, C.C. Pain, M.D. Piggott, A.J.H. Goddard, C.R.E. de Oliveira, and A.P. Umpleby. A nonhydrostatic finite-element model for three-dimensional stratified oceanic flows. Part II: Model validation. *Monthly Weather Review*, 132:28322844, 2004.

[30] P.L. Fraenkel. Power from marine currents. *Journal of Power and Energy*, 216:1–14, 2002.

[31] Christopher Garrett. Tidal resonance in the Bay of Fundy and Gulf of Maine. *Nature*, 238:441–443, 1972.

[32] B. Gjevik, H. Moe, and A. Ommundsen. Strong topographic enhancement of tidal currents: tales of the Maelstrom. *Department of Mathematics, University of Oslo*, 1997.

[33] B. Gjevik and T. Straume. Model simulations of the M2 and K1 tide in the Nordic Seas and the Artic Ocean. *Tellus*, 41A:73–96, 1989.

[34] H. Glauert. *Aerodynamic Theory*, chapter X, page Div. L. Springer, 1935.

[35] R. Gomez-Elvira, A. Crespo, E. Migoya, F. Manuel, and J. Hernandez. Anisotropy of turbulence in wind turbine wakes. *Journal of Wind Engineering*, 93:797–814, 2005.

[36] U. Högstrom, D. N. Asimakoupoulos, H. Kambezidis, C. G. Helmis, and A Smedman. A field study of the wake behind a 2 mw wind turbine. *Atmospheric Environment*, 22:803–820, 1988.

[37] M.Z. Hossain, H. Hirahara, Y. Nonomura, and M. Kawahashi. The wake structure in a 2d grid installation of the horizontal axis micro wind turbines. *Renewable Energy*, 32:2247–2267, 2007.

[38] W. Huang and R.D. Russell. Moving mesh strategy based on a gradient flow equation for two-dimensional equations. *SIAM Journal of Scientific Computing*, 20:9981015, 1999.

[39] W. Huang and M. Spaulding. 3d model of estuarine circulation and water-quality induced by surface discharges. *Journal of Hydraulic Engineering*, 121:300–311, 1995.

[40] N. Jarrin, S. Benhamadouche, D. Laurence, and R. Prosser. A synthetic-eddy-method for generating inflow conditions for large-eddy simulations. *International Journal of Heat and Fluid Flow*, 27:585–593, 2006.

[41] M. Jureczko, M. Pawlak, and A. Mezyk. Optimisation of wind turbine blades. *Journal of Materials Processing Technology*, 167:463–471, 2005.

[42] Z. Kowalik and T.S. Murty. *Numerical Modelling of Ocean Dynamics.* 1993.

[43] S. Legrand, V. Legat, and E. Deleersnijder. Delaunay mesh generation for an unstructured-grid ocean general circulation model. *Ocean Modelling*, 2:17–28, 2000.

[44] George Lemonis. Wave and tidal energy conversion. *Encyclopedia of Energy*, 6:385–396, 2004.

[45] Marcel Lesieur and Olivier Metais. New trends in large-eddy simulations of turbulence. *Annual Review of Fluid Mechanics*, 28:45–82, 1996.

[46] C.W. Li and B. Zhu. A sigma coordinate 3D k-epsilon model for turbulent free surface flow over a submerged structure. *Applied Mathematical Modelling*, 26:1139–1150, 2002.

[47] P.B.S. Lissaman. Energy effectiveness of arbitrary arrays of wind turbines. *Journal of Energy*, 3:323–328, 1979.

[48] Marine Current Turbines Ltd. Worlds first offshore tidal current turbine successfully installed. Press release, June 2003.

[49] M. Magnusson and A.-S. Smedman. Air flow behind wind turbines. *Journal of Wind Engineering*, 80:169–189, 1999.

[50] John Marshall, Alistair Adcroft, Chris Hill, Lev Perelman, and Curt Heisey. A finite-volume, incompressible Navier-Stokes model for studies of the ocean on parallel computers. *Journal of Geophysical Research*, 102 C3:5753–5766, 1997.

[51] John Marshall, Chris Hill, Lev Perelman, and Alistair Adcroft. Hydrostatic, quasi-hydrostatic, and nonhydrostatic ocean modelling. *Journal of Geophysical Research*, 102 C3:5733–5752, 1997.

[52] B. Massey. *Mechanics of Fluids, 8th ed.* Nelson Thornes, 1998.

[53] Mathematics and Computer Science Division, Argonne National Laboratory, United States. *MPICH2 Users' Guide.*

[54] Charles Meneveau and Joseph Katz. Scale-invariance and turbulence models for large-eddy simulation. *Annual Review of Fluid Mechanics*, 32:1–32, 2000.

[55] Robert Mikkelsen. *Actuator disc methods applied to wind turbines.* PhD thesis, Technical University of Denmark, 2003.

[56] Luke Myers and A.S. Bahaj. Wake studies of a 1/30th scale horizontal axis marine current turbine. *Ocean Engineering*, 34:758–762, 2007.

[57] D. Nechaev, J. Schrter, and M. Yaremchuk. A diagnostic stabilized finite-element ocean circulation model. *Ocean Modelling*, 5 (1):37–63, 2003.

[58] E. Nøst. Calculating tidal current profiles from vertically integrated models near the critical latitude in the Barents Sea. *Journal of Geophysical Research*, 99 C4:7885–7901, 1994.

[59] UK Department of Trade and Industry. Development, installation and testing of a large scale tidal current turbine. Technical report, 2005.

[60] C.C. Pain, M.D. Piggot, A.J.H. Goddard, F. Fang, G.J. Gorman, D.P. Marshall, M.D. Eaton, P.W. Power, and C.R.E. de Oliveira. Three-dimensional unstructured mesh ocean modelling. *Ocean Modelling*, 10:533, 2005.

[61] F Pascheke and PH Hancock. Wake development and interactions within an array of large turbines. In *European Wind Energy Conference (EWEC)*, 2008.

[62] S.V. Patankar and D.B. Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International Journal of Heat Mass Transfer*, 15:1787–1806, 1972.

[63] M.D. Piggott. *Fluidity/ICOM Manual.* Applied Modelling and Computation Group, Imperial College in London, England, 2007.

[64] M.D. Piggott, C.C. Pain, G.J. Gorman, P.W. Power, and A.J.H. Goddard. h, r, and hr adaptivity with applications in numerical ocean modelling. *Ocean Modelling*, 2004.

[65] J. W. Schwiderski. On charting global ocean tides. *Reviews of Geophysics and Space Physics*, 18:243–268, 1980.

[66] D.J. Sharpe. A general momentum theory applied to an energy-extracting actuator disc. *Wind Energy*, 7:177–188, 2004.

[67] D. Smith and G.J. Taylor. Further analysis of turbine wake development and interaction data. *BWEA Wind Energy Conference*, 13:325–331, 1991.

[68] Jens Nørkær Sørensen and Wen Zhong Shen. Numerical modeling of wind turbine wakes. *Journal of Fluids Engineering*, 124:393–399, 2002.

[69] J.N. Sørensen, W.Z. Shen, and X. Munduate. Analysis of wake states by a full-field actuator disc model. *Wind Energy*, 1:73–88, 1998.

[70] Kenneth Thomsen and Helge Aagaard Madsen. A new simulation method for turbines in wake - applied to extreme response during operating. *Wind Energy*, 8:35–47, 2005.

[71] D.J. Tritton. *Physical Fluid Dynamics.* Oxford University Press, 1988.

[72] G. Umgiesser and A. Bergamasco. A staggered grid finite element model of the Venice Lagoon. *Finite Elements in Fluids*, 2, 1993.

[73] L.J. Vermeer, J.N. Srensen, and A. Crespo. Wind turbine wake aerodynamics. *Progress in Aerospace Sciences*, 39:467–510, 2003.

[74] C.B. Vreugdenhil. *Numerical Methods for Shallow-Water Flow.* Kluwer Academic Publishers, 1994.

[75] Steffen Wußow, Lars Sitzki, and Thomas Hahm. 3D-simulation of the turbulent wake behind a wind turbine. *Journal of Physics: Conference Series*, 75:1–8, 2007.

[76] Genki Yagawa, Yasushi Nakabayashi, and Hiroshi Okuda. Large-scale finite element fluid analysis by massively parallel processors. *Parallel Computing*, 23:1365–1377, 1996.

[77] Keisuki Yamaguchi, Ju Lin, Arata Kaneko, Tokuo Yayamoto, Noriaki Gohda, Hong-Quang Nguyen, and Hong Zheng. A continuous mapping of tidal current structures in the Kanmon Strait. *Journal of Oceanography*, 61:283–294, 2005.