Reduced Order Modelling of Hydraulic Valves

Peter Hall

Submitted to Swansea University in fulfilment of the requirements for the Degree of Doctor of Engineering

Swansea University

2022

Summary

There is a need for accurate models of small and highly dynamic one-way valves such as those found in engine oil systems. Such reduced order models can then be included in larger oil system models that are too complex to model at high resolution.

Two one-way valves are introduced and analysed using a fluid structure interaction simulation. The behaviour of the valves is examined, and the results generated are then used to develop two reduced order methodologies.

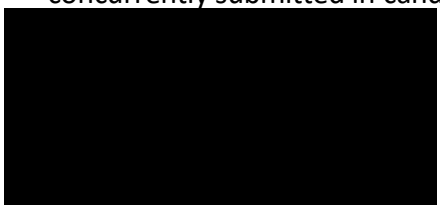
The first is physics based. It combines physical phenomena and steady state simulation results to build up an accurate valve model. This model provides good results but is limited by the applicability of steady state simulations to dynamic behaviour, which is itself dependent on valve geometry and fluid properties.

The second approach is based on a type of continuous-time recurrent neural network. This allows training on fluid structure interaction simulations with no need for physical insight. The network is trained directly on the dynamic behaviour and accurately reproduces it. Training methods are discussed. A multiple particle swarm methodology and in situ training technique is presented. The neural network-based model is applied to a more complex valve. Network architecture and size are investigated. A comparison is done over a range of time step sizes.

Finally, an example workflow is given presenting the steps to train a neural network and use the resulting reduced order model within a larger oil system model.

Declarations

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.



.....

Date: 31/03/2022

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

.....

Date: 31/03/2022

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

.....

Date: 31/03/2022

The University's ethical procedures have been followed and, where appropriate, that ethical approval has been granted.

.....

Date: 31/03/2022

Contents

Acknowledgements

Many thanks to my supervisors professor Wulf Dettmer and professor Djordje Peric for the advice and guidance. Thank you to Dr Chennakesava Kadapa for putting up with my many questions about MPAP. My thanks must also go to Dr Jochen Broz and Dr. Oliver Hahn at Schaeffler. It is a shame that covid prevented our last final meetup in Herzogenaurach.

List of Figures

List of Algorithms

Abbreviations

BC	**B**oundary **C**ondition
FSI	**F**luid **S**tructure **I**nteraction
GA	**G**enetic **A**lgorithm
KE	**K**inetic **E**nergy
LSTM	**L**ong **S**hort **T**erm **M**emory
MATLAB	**Mat**rix **Lab**oratory
MEX	MATLAB **Ex**ecutable
MPAP	**M**ulti **P**hysics **A**nalysis **P**rogram
NEAT	**N**euro**E**volution of **A**ugmenting **T**opologies
OpenFOAM	**Open**-source **F**ield **O**peration **A**nd **M**anipulation
PSO	**P**article **S**warm **O**ptimisation
QDPSO	**Q**uantum **D**elta-potential-well-based **P**article **S**warm **O**ptimisation
ReLU	**Re**ctified **L**inear **U**nit
ROM	**R**educed **O**rder **M**odel
tansig	hyperbolic **tan**gent **sig**moid

TWEANN **T**opology and **W**eight **E**volving **A**rtificial **N**eural **N**etwork

Symbols

μ	Dynamic viscosity
A	Representative valve cross sectional area
b	Neuron bias
c	Damping coefficient
C_a	Added mass coefficient
E	Fluid kinetic energy
F_{disk}	Total force on valve disk element
F_{lower}	Force on lower surface of valve element
F_{upper}	Force on upper surface of valve element
$\bar{F}$	Force on valve element look up table
h	Moving valve element height
I	Inertance
k	Spring constant
m_{added}	Added mass
m_{disk}	Mass of valve disk element

N	Reynolds number
P_{lower}	Pressure on lower surface of valve element
P_{upper}	Pressure on upper surface of valve element
$\bar{P}$	Pressure differential over domain look up table
Q	Flow rate
$\bar{Q}$	Flow rate look up table
$Q_{displacment}$	Displacement flow rate
r	Radius along disk element
r_b	Moving ball element radius
r_d	Moving disk element radius
r_d	Displacement radius
r_t	Valve throat radius
$\bar{U}$	Average velocity at valve throat radius
w	Connection weight
β	Velocity distribution factor
ν	Kinematic viscosity
ρ	Density

Chapter 1

Introduction

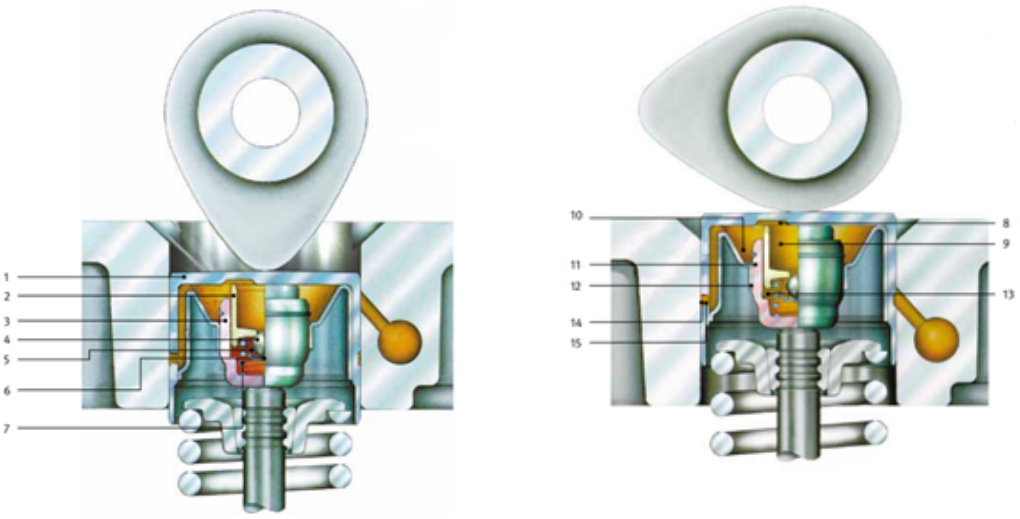
This work investigates methods for improving reduced order modelling of one-way valves. This objective is driven by a need of Schaeffler, the industrial sponsor of this research. Using Swansea's own fluid structure interaction tool, a better understanding is gained and then applied to both physics, and neural network based reduced order models. Finally, a workflow for creation of reduced order models is presented that could be used at Schaeffler in the future.

1.1 Hydraulic valves and applications

To reduce emissions and improve efficiency, internal combustion engines are becoming increasingly complex. The management systems of these engines often rely on one-way hydraulic valves to operate key components. Any improvement in modelling and understanding the response of these valves will allow the engine management system to provide a finer level of control, thus maximising performance. Many such valves must operate at a high frequency so commercial fluid dynamics tools cannot be used due to the large added mass effects. Example uses of the valves such as those discussed in this work are hydraulic valve lash

adjusters and belt tensioners.

Valve lash adjusters remove backlash between the cam shaft and the engines intake and exhaust valves. This allows more accurate valve control. Valve train systems must account for the change in length of components such as push rods and the valves themselves as the engine warms up. Prior to the introduction of hydraulic adjusters fixed systems were used. The lash had to be set with the engine hot such that expansion of components did not result in the engine seizing up. This resulted in poor performance when the engine was started cold. Fixed systems must also be regularly adjusted to account for component wear. Hydraulic lash adjusters are powered by the engine oil system. Oil pressure extends the length of the adjuster such that it is always in contact with the cam shaft. However, the load on the adjuster is much greater than that which could be provided by the engine oil pump alone. A one-way valve prevents the oil being squeezed back out of the adjuster. This means the adjuster can only increase in length. While the intake or exhaust valve is not active the oil pressure extends the adjuster length as much as possible, via the one-way valve it then locks solid when it is compressed by the cam shaft. Of course, as the engine heats up the adjuster must reduce in length, to accomplish this a small leak path is incorporated. The increase in length each cycle due to the action of the valve is always more than the reduction due to the leak path. The adjuster behaves as if solid in the high frequency time frame of the cam shaft but maintains its ability to account for thermal expansion and wear that occur in a much slower timeframe. See "The Valve Train System" for some examples of the Schaeffer components that utilise one-way valves [1], some images from that publication are included below.



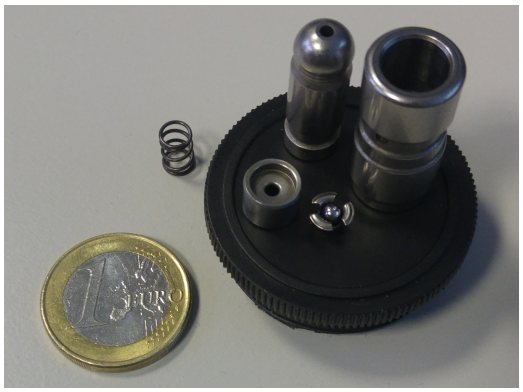
(a) The adjuster is compressed by the cam shaft, the ball valve closes and the adjuster locks. A small leak path allows the adjuster to compress very slowly to account for thermal expansion.

(b) The adjustment phase, load is reduced and the ball valve opens allowing engine oil pressure to extend the adjuster. It is this adjustment capability that takes up any ware in the system removing the need for frequent servicing.

Figure 1.1: Hydraulic adjuster operational examples from "The Valve Train System" [1]. Shown is a cam shaft and engine poppet valve. The hydraulic adjuster is the component in between, inside the adjuster is a ball valve of the type considered in this work.

1. Outer housing
2. Piston
3. Inner housing
4. Valve ball
5. Valve spring
6. Valve ball retainer
7. Return spring
8. Oil carry-over
9. Oil reservoir
10. Oil reservoir
11. Leakage gap
12. Guide gap
13. high pressure chamber
14. Oil supply grove
15. Intake bore

(a) Cut away showing internal ball valve from "The Valve Train System" [1].



(b) Disassembled example.

Figure 1.2: Finger follower type hydraulic adjuster. This is an alternative arrangement to achieve the same result. The follower is the pivot of a third class lever, the round ball bearing runs on the cam shaft and flat end actuates the poppet valve.

1.2 Industrial motivation

Schaeffler designs and produces valves that are used throughout the automotive industry. Due to the small size and high frequency nature of these valves (1000 to 7000 rpm) it is impossible to observe and measure them in situ. Higher level specifications can be found such as the opening pressure and closing time. However, to gain an insight into what is going on inside the valve simulations must be used. This allows improved geometries to be developed. This work concentrates on the next step, taking those improved geometries and allowing them to be included in full oil system simulations. Within the context of a vehicle's oil system valves cannot be considered in isolation, their effect on other components must be understood. Full fluid structure interaction simulations of

the whole system are not possible, it is too complex. The next best solution is to use a reduced order model of each component. These reduced order models can then be used in combination to model the whole system. For the whole system model to be representative each individual reduced order component model must be as accurate as possible.

1.3 Objectives

The key industrial objective of this work is to develop and document a workflow for producing an accurate reduced order valve model from high resolution simulations. The approach should be as versatile as possible allowing different valve types and geometries to be represented. The resulting reduced order models should be of a type that allows integration into simulations of larger systems.

1.4 Overview of thesis

The second chapter of this work looks at high resolution modelling of hydraulic valves. This provides results which the reduced order models developed in the following chapters will be compared to. Two variants of one-way valves are introduced, a disk valve and ball valve. The disk valve is used as a benchmark to compare Swansea's fluid simulation software against that used at Schaeffler. An in-depth look is taken into the ball valve with fluid structure interaction simulations.

The third chapter covers a physics based approach to creating a reduced order model. Firstly, purely theoretical methods are investigated. A lookup based methodology is then built up from a number of basic phenomena. An approach to parameter generation for the model is presented and then applied to the disk

valve. Finally, the methodology is applied on the ball valve and its shortcomings discussed.

In the fourth chapter a neural network approach is presented. The network architecture is described along with a scheme for integrating the network into a reduced order model. Methods to increase the accuracy and complexity of functions a network can represent are covered. Training methods are discussed along with methods to increase consistency.

The fifth chapter takes the neural network based reduced order model presented in the preceding chapter and applies it to the ball valve. An investigation is done into network architecture and reduced order model robustness.

The final chapter follows a case study applying the neural network approach to a new valve. The reduced order model is then incorporated as a smaller part of a larger system simulation. This outlines the workflow that could be used at Schaeffler and accomplishes one of the main objectives of this work. Again, an investigation into robustness and architecture is presented.

Chapter 2

High resolution computational modelling

A source of high-resolution models was required to inform and benchmark the reduced order models developed in this work. Swansea's Multi Physics Analysis Program (MPAP) was used for this. To ensure reliable results a test problem was used to compare MPAP with Open-source Field Operation and Manipulation (OpenFOAM) which is used extensively at Schaeffler. This also allowed the author to gain experience operating both programs and gain confidence in the setup and operation of MPAP. All simulations in this work are axisymmetric, the valve geometries considered lend themselves to axisymmetric simulation thanks to their simple geometry and low Reynolds numbers. Both MPAP and OpenFOAM require a visualization and analysis tool to interpret results. ParaView was used for this purpose and to perform calculations such as integrating to find the fluid kinetic energy or measuring the pressure at specific points [2].

2.1 MPAP

Swansea's Multi Physics Analysis Program (MPAP) has been developed over several years specialising in fluid structure interaction. MPAP employs a staggered scheme to solve both 2D and 3D problems [3]. In this work strictly rigid bodies are considered although MPAP has the capacity to model fluid-flexible structure interaction. MPAP was then extended to add support for a stabilised immersed boundary method on hierarchical b-spline grids [4] and solid-solid contact [5]. MPAP was run in steady state mode where there was no need to capture transient flow.

2.2 OpenFOAM

As the name suggest OpenFOAM is an open-source toolbox for numerical solvers [6]. There are dozens of solvers for various problems, on the recommendation of Schaeffler the pimpleFOAM solver was used for this comparison. PimpleFOAM is a finite volume transient solver for incompressible fluids [7] [8].

2.3 Check valve

A basic test valve was chosen for the comparison of simulation tools. This took the form of a check valve, also known as one-way valve, with a moving disk and a straight seat. In operation a positive pressure causes the disk to move away from the seat where it is be retained by a stop or spring. A negative pressure moves the disk element against the seat causing it to seal preventing any further fluid flow. There is no additional seal or sealing material required. The metal on metal contact of the disk and seat seals providing the components are manufactured with sufficient tolerances. Dimensions of the test valve are

shown in Figure 2.1. This valve is designed to operate with engine oil therefore the simulations were done using a fluid density ρ of $850kg/m^3$, kinematic viscosity ν of $3.05 * 10^{-5}m^2/s$ and dynamic viscosity μ of $0.02593kg/ms$. The valve throat radius r_t was $1.325mm$ with disk radius r_d of $2.625mm$. This comparison was performed using velocity boundary conditions of $\pm 0.5m/s$ on the upper edge of the domain. The disk element was fixed with a height h of $0.24mm$. In both MPAP and OpenFOAM the outer edges of the valve and disk element used no-slip Dirichlet boundary conditions.

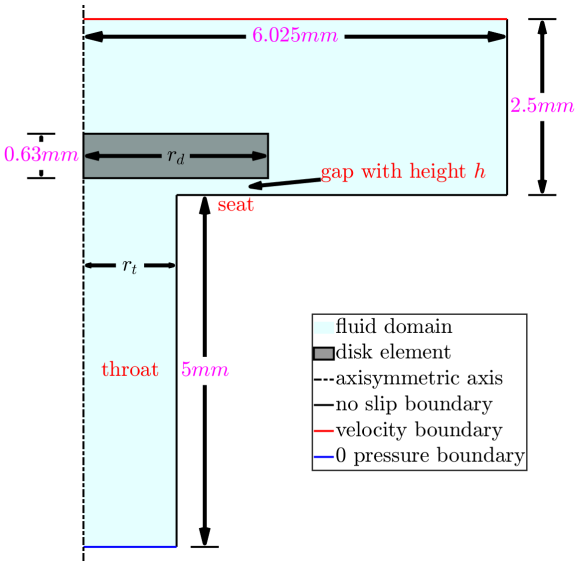
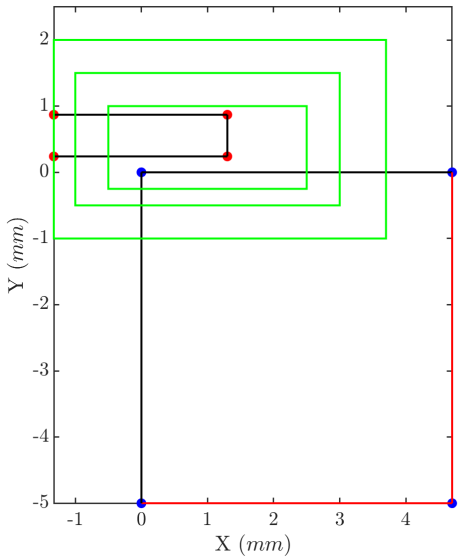


Figure 2.1: Check valve overview

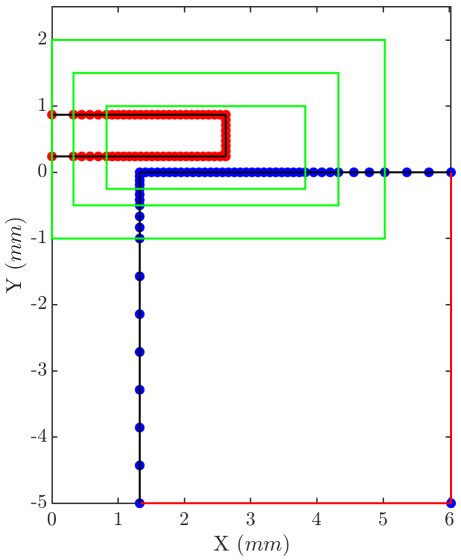
2.3.1 MPAP

MPAP requires an input file defining the vertices and edges of the geometry to be modelled. There must be at least one vertex per mesh element, vertices must be defined clockwise around the perimeter of the geometry. This process was automated using MATLAB. Thus, a rectangle could be defined with four vertexes and automatically split into the correct number of vertexes for a given

mesh density. The tool also shows a preview of the geometry and refinement zones allowing the user to quickly double check the geometry without having to generate the final mesh. An example input geometry for the test valve is shown in Figure 2.2a and the resulting geometry as output from the MATLAB script is shown in Figure 2.2b.



(a) Geometry as input into the MATLAB tool, red lines show switched off edges at the domain boundary, refinement zones shown in green, body one points in blue and body two points in red



(b) Geometry as output by MATLAB tool, element size increased to demonstrate variable point spacing depending on refinement level, bodies automatically sifted right to allow Axis-symmetric simulation about x = 0

Figure 2.2: MPAP MATLAB input file tool input (a) and output (b)

2.3.2 OpenFOAM

OpenFOAM requires a mesh to be constructed using a secondary tool, in this case BlockMesh was used [9]. In order to perform an axisymmetric simulation OpenFOAM requires a wedge shape mesh of one element thickness [10]. Again, a MATLAB script was produced to generate the input file for BlockMesh. The code takes 2D geometry blocks as defined by the user and rotates them 2.5 de-

grees clockwise and anticlockwise, resulting in the 5-degree wedge mesh. Each face must then be assigned a type and appropriate boundary conditions manually. The blocks as input into the MATLAB tool are shown in Figure 2.3. The pimpleFOAM solver does not have a dedicated steady state mode so as such transient simulations were run for a sufficiently long period to reach steady state.

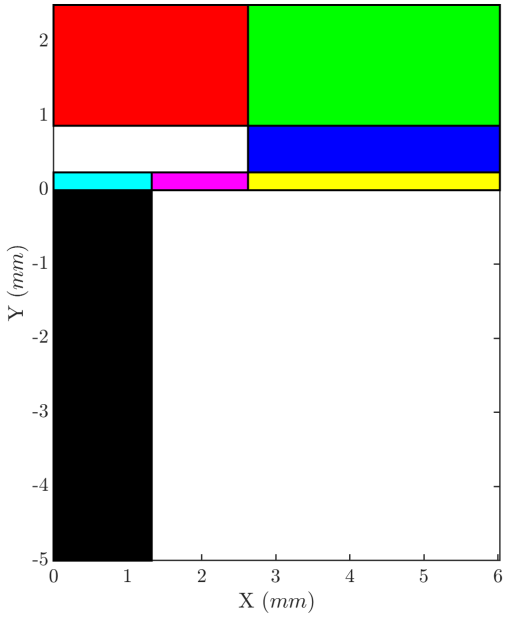


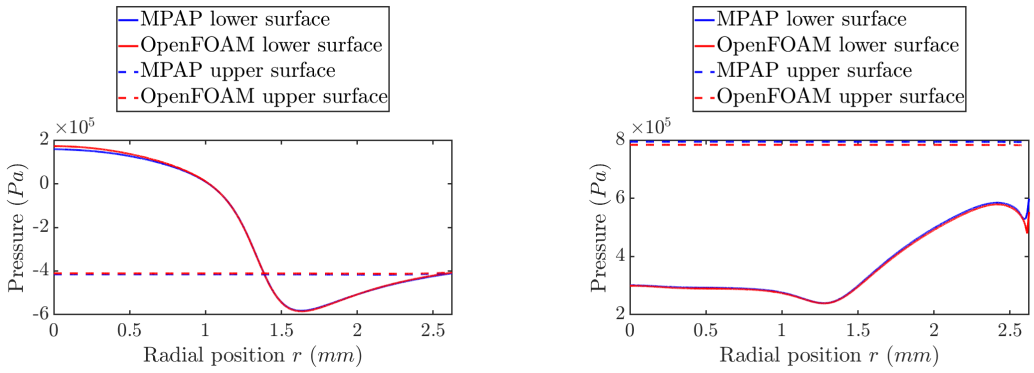
Figure 2.3: BlockMesh 2D representation of the valve, seven blocks are required to insure no block edges are split

2.3.3 Results

Both OpenFOAM and MPAP were run with a prescribed velocity of $\pm 0.5 m/s$ at the upper boundary. The force acting on the disk was calculated for each simulation to allow comparison. This force becomes important when considering moving disk simulations where the force will affect the opening and closing response. The pressure distribution on the upper and lower surface of the disk was also compared, this highlights any differences in flow behaviour that may be masked by comparing the total force alone.

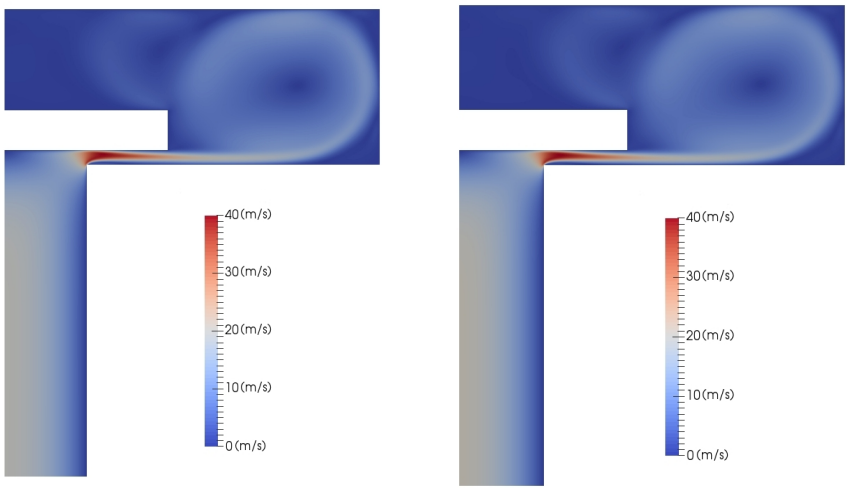
Force values for positive inflow are $1.1548N$ for MPAP and $1.0994N$ for OpenFOAM. This is a difference of 5.04%. A better match is achieved for outflow with MPAP giving $-7.976N$ and OpenFOAM $-7.908N$ a difference of 0.86%. Both flow directions result in a difference of $\approx 0.05N$, this may be due to differences in the mesh but is more likely due to OpenFOAMs transient solver picking up effects that MPAP misses in steady state mode. Comparison of the pressure distribution on the upper and lower surfaces of the disk are a close match. Inflow is shown in Figure 2.4a and outflow in Figure 2.4b.

The velocity and pressure plots also show a close correlation between MPAP and OpenFOAM. They give an insight into the flow behaviour that will have to be considered when modelling the valve in greater depth. Inflow velocity and pressure distribution are shown in Figure 2.5 and Figure 2.6 respectively. In the inflow condition a large vortex forms due to the accelerated flow through the gap. The maximum velocity is $\approx 40m/s$, using the gap height of $0.24mm$ this results in a Reynolds number of $re \approx 300$. Therefore it is safe to assume nonturbulent flow. Outflow velocity and pressure distribution are shown in Figure 2.7 and Figure 2.8 respectively.



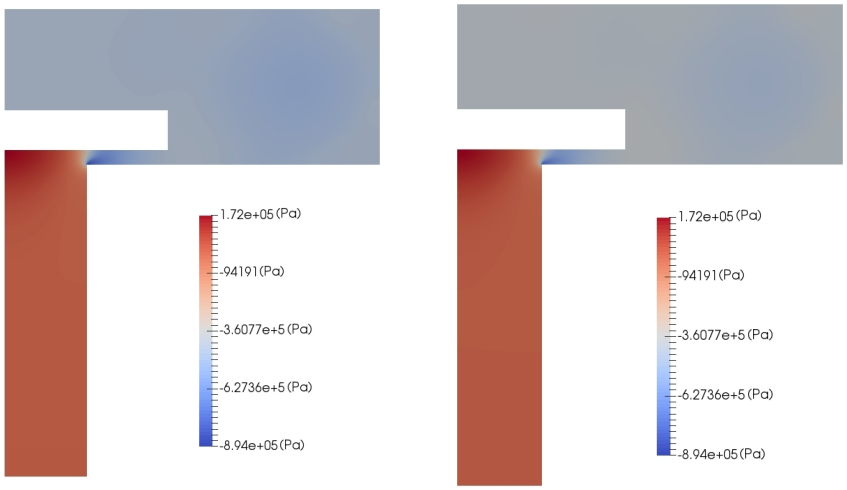
(a) Inflow (b) Outflow

Figure 2.4: Disk upper and lower surface pressure distribution comparison



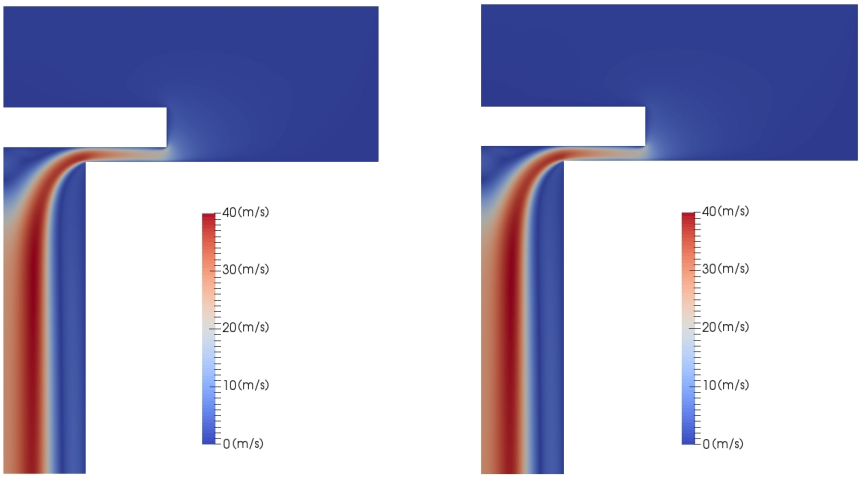
(a) MPAP velocity magnitude (b) OpenFOAM velocity magnitude

Figure 2.5: Inflow velocity distribution



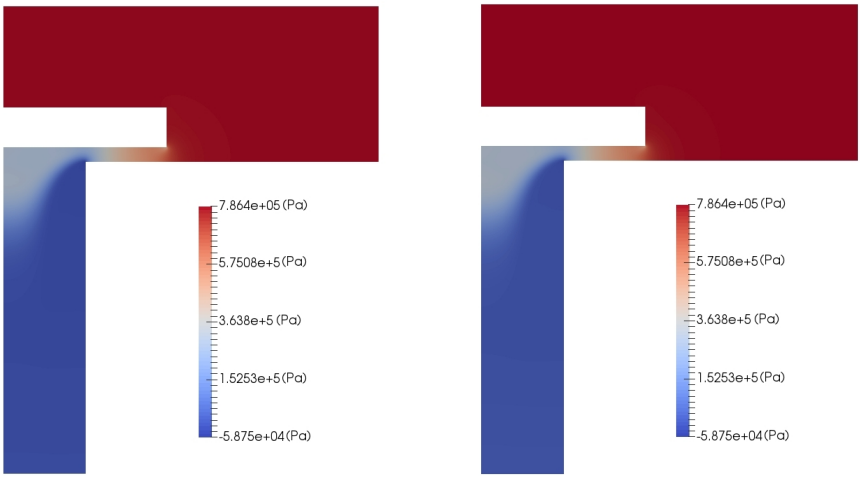
(a) MPAP pressure (b) OpenFOAM pressure

Figure 2.6: Inflow pressure distribution



(a) MPAP velocity magnitude (b) OpenFOAM velocity magnitude

Figure 2.7: Outflow velocity distribution



(a) MPAP pressure (b) OpenFOAM pressure

Figure 2.8: Outflow pressure distribution

2.4 Ball valve

With confidence gained in MPAP a more complex ball valve was simulated. Ball valves have the advantage of a reduced seat region allowing faster opening and closing. Transient fluid structure iteration simulations were run with a range of sinusoidal boundary conditions. No comparison to OpenFOAM was possible due to the large added mass effects. The valve consists of a generic ball and seat geometry with a spring holding the valve closed. Fluid with density ρ of $800kg/m^3$, kinematic viscosity ν of $2.35 * 10^{-5}m^2/s$ and dynamic viscosity μ of $0.02kg/ms$ was used. A virtual spring with preload of $0.007N$ and spring constant k of $0.3N/mm$ holds the ball against the seat. Minimum ball height is limited to 0.1mm, this prevents the domain closing completely and reduces the need for very small elements in the seat region, the flow rate through the domain approaches zero in this condition. Figure 2.9 shows the valve geometry and Figure 2.10 shows an example velocity distribution in the opening case.

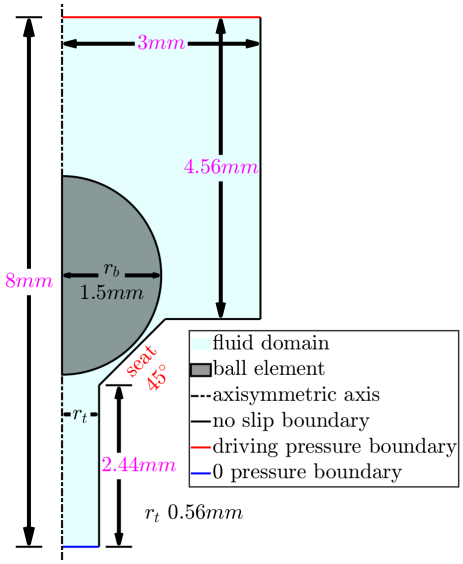


Figure 2.9: Ball valve overview and boundary conditions

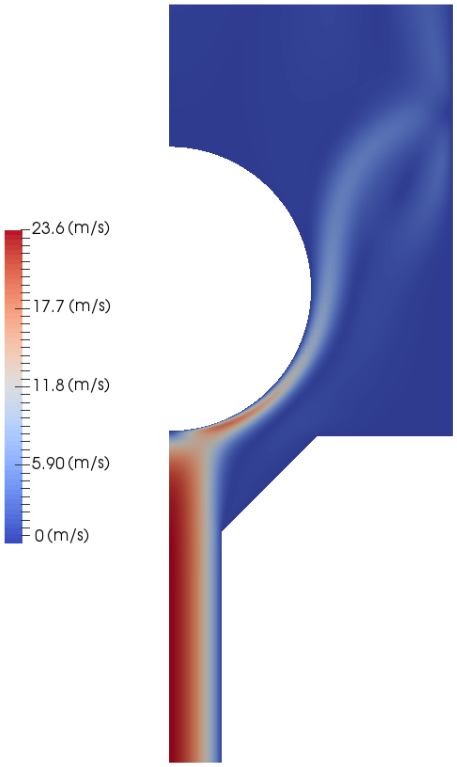


Figure 2.10: Velocity distribution, opening

2.4.1 Results

Figure 2.11 shows the results of simulations with boundary condition pressure frequencies from $160Hz$ to $1.25Hz$. Even at the highest frequency tested the behaviour remains largely periodic, the valve can close sufficiently to re-set the domain. There is some difference between the first period and subsequent periods as the flow becomes established, this is most noticeable when comparing the two highest frequency shown in red and pink. The integrated kinetic energy and vorticity over the whole domain were calculated, it was hoped that these values would give some insight into vortex formation. The three lowest frequencies tested approach steady state with little difference in response between them.

No mesh or integration convergence studies were done. The goal of this section is not to provide highly accurate simulations of a particular value geometry. Rather this work provides representative valve behaviour for the reduced order methods explored in the subsequent chapters to be benchmarked against.

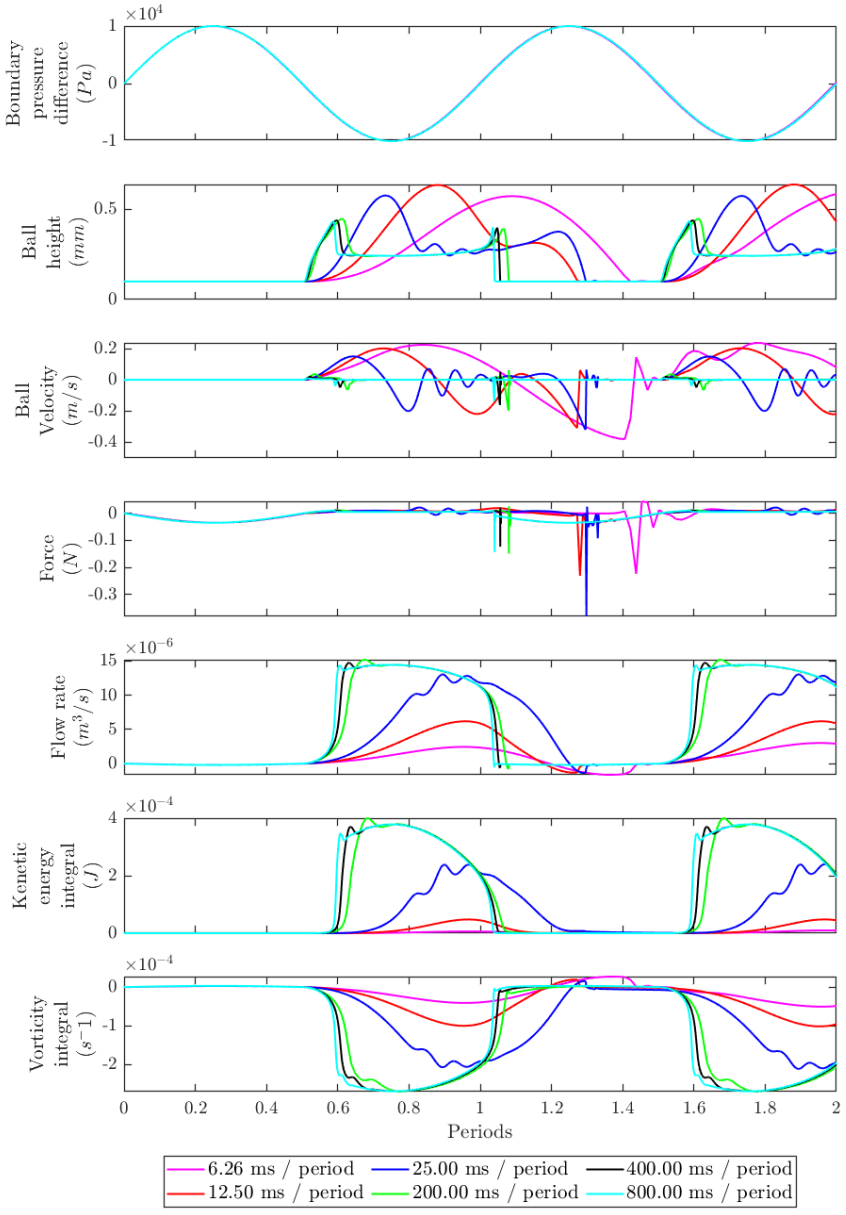


Figure 2.11: Ball valve results, $160Hz$, $80Hz$, $40Hz$, $5Hz$, $2.5Hz$, $1.25Hz$, normalised to periods

Figure 2.12 highlights the unintuitive ball height behaviour seen, this is present in all frequencies except the highest, $160Hz$. To ensure this behaviour was not an interaction with the boundary condition an increased upper domain size was also tested. As the frequency increases the peak height is delayed in the cycle due to the increasing dominance of inertial effects of both the ball and fluid. The slower simulations show a quite extreme double peak in height, the ball opens nearly as high as it does when first opening. Rather than a second opening these simulations show a reduced height at the peak pressure. This height set back is due to vortex formation above the valve. The faster simulations reach a much larger peak height. As the frequency increases the double peak effect is reduced. The fastest simulation tested shows no second peak as the vortex does not have time to form.

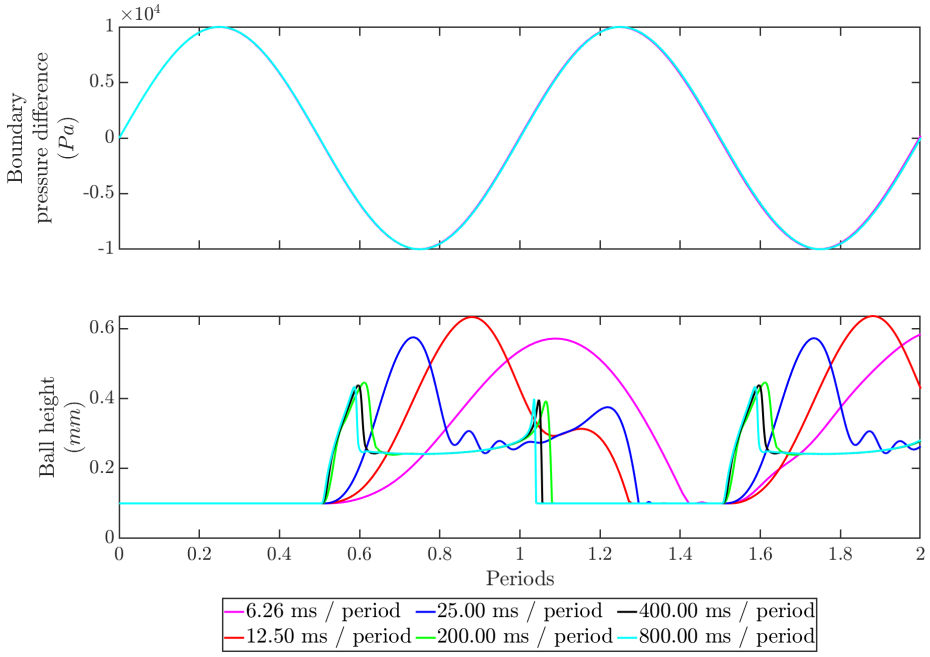


Figure 2.12: Double peak behaviour, $160Hz$, $80Hz$, $40Hz$, $5Hz$, $2.5Hz$, $1.25Hz$, normalised to periods

The vortex can be clearly seen in Figure 2.13. The vortex sits over the ball increasing the pressure on is upper surface, this causes the ball to close slightly, thus accelerating the flow in the gap region further feeding the vortex, this feedback effect is seen in the $40Hz$ simulation (blue) Figure 2.12.

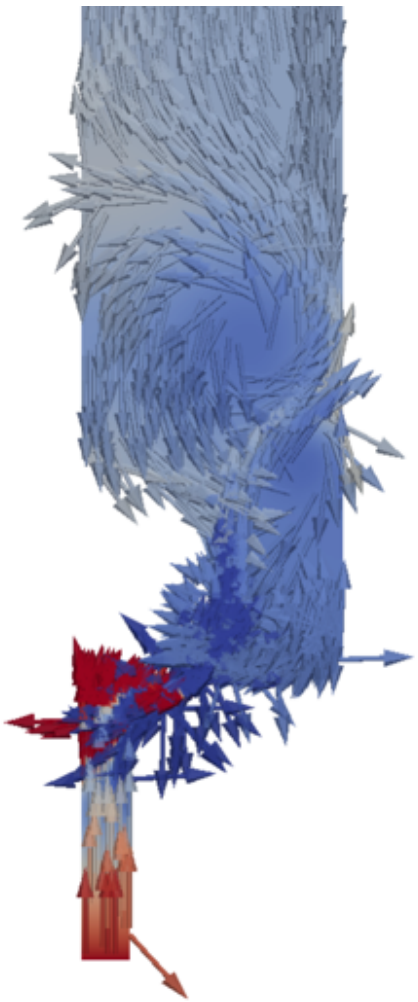


Figure 2.13: Vortex well established above ball in $1.25Hz$ simulation

Chapter 3

Physics based reduced order model

Armed with the knowledge gained from simulations a physics based reduced order model was derived. Initially this effort concentrated on the simpler disk check valve. Several opening and closing simulations were run using MPAP for comparison. In order to make the problem more physical and remove boundary effects the upper boundary condition was changed from the velocity boundary as described in section 2.3 to a pressure boundary condition. The lower boundary remains zero pressure. The goal of the reduced order model was to remove the requirement to run any full fluid structure interaction simulations. This would allow the model to be generated from simulations that could be run in OpenFOAM. Initially this work concentrated on the disk valve.

3.1 Preliminary investigations

To predict the disk's movement, the force acting on it must be calculated. This force can be calculated by finding the pressure distribution across the surface of

the disk and integrating over surface area. Theoretical methods to achieve this were investigated. Non-dimensionalization was also investigated as a method of maximising the range of valves and fluids the model could be applied to.

3.1.1 Theoretical pressure distribution

Any calculation must account for the change in gap height as the disk moves. When the disk is very close to its seat the flow rate will be greatly diminished and viscous effects will dominate. In these viscous flow conditions, the direction of flow will have no effect on the magnitude force acting on the disk, only its direction. This is because viscous pressure drop is independent of flow direction. At larger disk heights and there resulting increased flow rates inertial terms will be dominant.

To allow direct comparison with MPAP a new simulation was set up using pressure boundary conditions. This allows a direct comparison between the computational results and the theoretical calculation. The valve geometry was also extended, moving the upper edge of the domain away from the disk as shown in Figure 3.1. This removes complex interactions with the boundary conditions that could not be matched using the basic theoretical calculation.

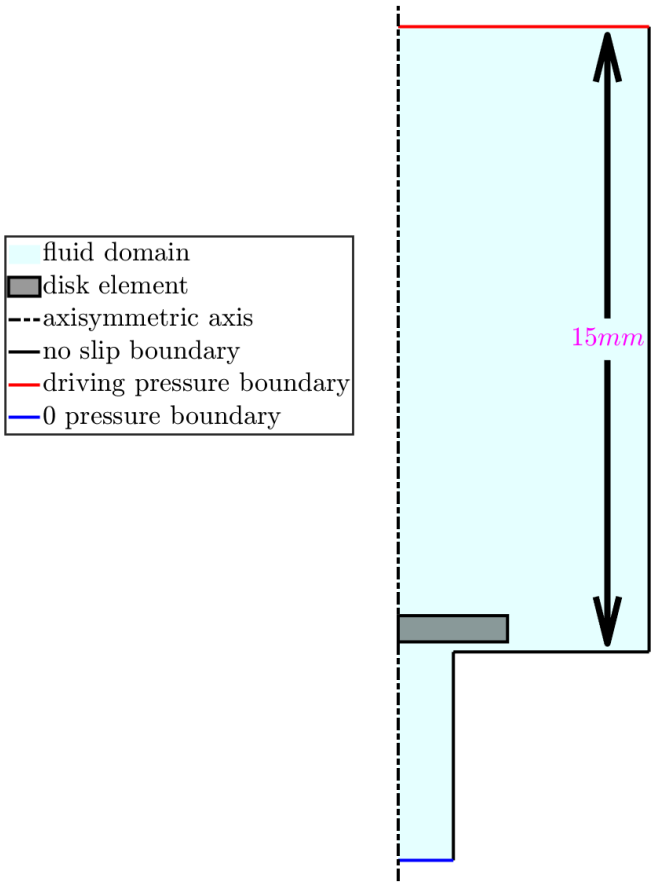


Figure 3.1: Extended disk valve geometry. Upper portion of the value is extended from $2.5mm$ to $15mm$ to move the upper boundary away from the disk. All other dimensions are unchanged, see Figure 2.1. Upper boundary condition is also changed from velocity to pressure.

Viscous pressure drop

When viscous flow is dominant the flow rate is small, therefore only the pressure drop in the gap region was considered. There it is assumed there is no pressure drop above the disk, its upper surface is considered to have a uniform pressure equal to that at the upper boundary. Likewise, the pressure on the disk above the throat is assumed to be uniform and equal to that at the lower boundary. These assumptions will result in the calculated force being larger than the simulated

one, there will always be a small pressure drop over these regions.

The pressure in the gap region is calculated using a logarithmic drop to account for the radial flow with increasing cross-sectional area. This is then integrated over the surface area of the disk to calculate the force. This calculation results in an accurate match for MPAP simulations for small disk height. As the disk height increases viscous effects become less dominant and the calculation becomes less accurate as shown in Figure 3.2. At disk heights of $0.01mm$ or less the viscous force is a close match. The effect of vortex formation above the disk is clear, there is a much larger reduction in force due to inertial effects in the opening flow direction than seen in the closing direction.

$$F_{upper} = P_{upper}\pi {r_d}^2 \tag{3.1}$$

$$F_{lower} = P_{lower}\pi {r_t}^2 + \pi \int_{r_t}^{r_d} \left(\frac{\log(r/r_d)}{\log(r_t/r_d)}(P_{upper} - P_{lower}) + P_{lower} \right) r \tag{3.2}$$

$$F_{disk} = F_{lower} - F_{upper} \tag{3.3}$$

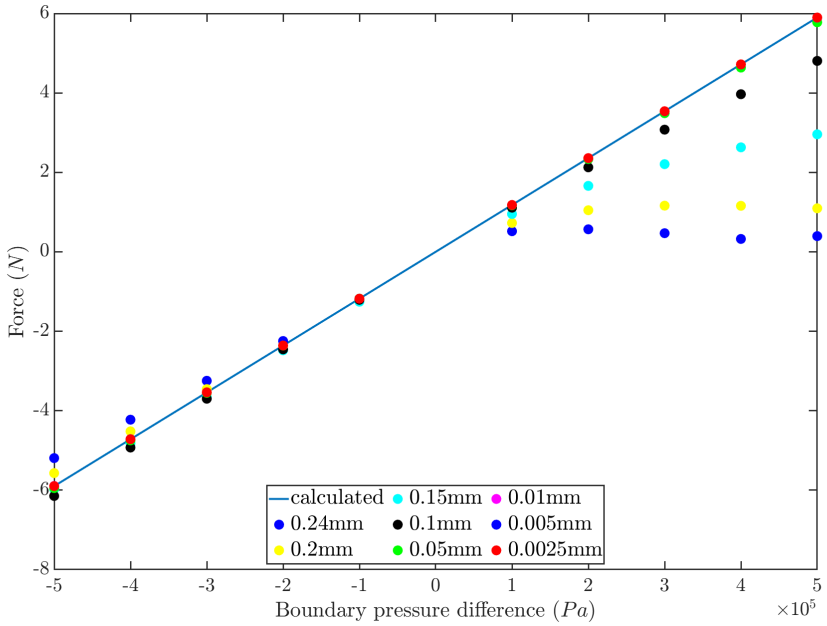


Figure 3.2: Calculated force from viscous pressure drop vs simulated

Velocity profile pressure drop

To accurately calculate the force at higher disk heights where inertial terms dominate a different method is required. An equation was found that accounts for the change of area and velocity profile as the flow travels through the gap, see Figure 3.3, Equation 3.4 [11]. This equation is derived from first principles assuming superposition of viscous wall shear and inviscid ideal diffusion. This allows the calculation to better match the simulated results. Although in this case the height of the disk is included in the equation the pressure is assumed to be uniform across the gap height. The MPAP simulation was sampled to give the pressure at half the gap height, this should give the best-case match for the equation.

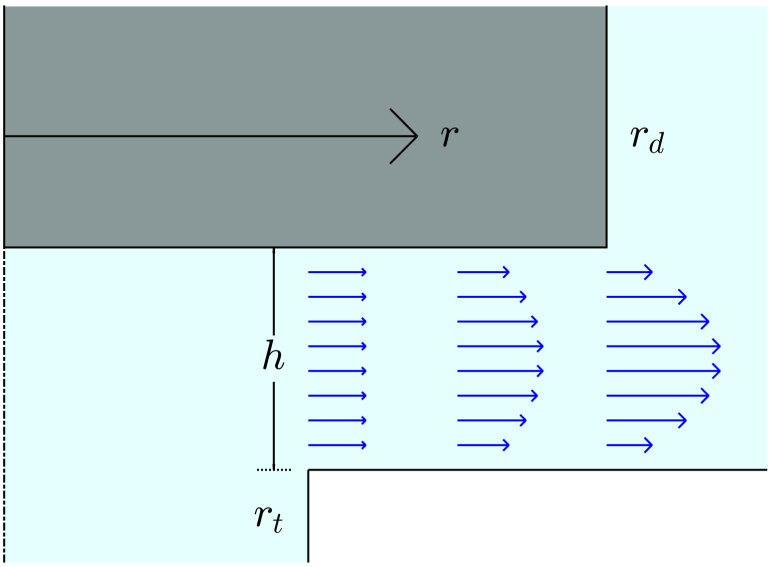


Figure 3.3: Calculating pressure distribution as a function of flow rate and velocity profile

$$\frac{P(r) - P_{upper}}{\rho \bar{U}^2/2} = 24 \frac{r_d/h}{N_{Rh}} \left(\frac{r_t}{r_d}\right) \ln\left(\frac{r_d}{r}\right) + \beta_{erf} \left(\frac{r_t}{r_d}\right)^2 - \beta_e \left(\frac{r_t}{r}\right)^2 \tag{3.4}$$

$$\bar{U} = \frac{Q}{2\pi r_t h} \tag{3.5}$$

$$N_{rh} = \bar{U} h/\nu \tag{3.6}$$

This equation is driven by the average velocity at the throat radius ($\bar{U}$), this can be easily calculated from the flow rate (Equation 3.5). The Reynolds number with respect to disk height is also required, see Equation 3.6. Correct selection of the velocity distribution factor β can result in a close fit to the simulated pressure drop. The velocity distribution factor must be defined at each point of the radius. It was found that using three values for β provided a good fit. β is

defined at r_t, r_d and halfway between the two, a piecewise linear interpolation between the three points is then used. A value of $\beta = 1$ results in a uniform flow profile, $\beta = 1.5$ would be a parabolic flow profile. It was found that β values of ≈ 2 provided a good fit, see Figure 3.4.

Although a good fit for the pressure drop in the gap was achieved it was found to be difficult to successfully calculate the pressure drop between the upper and lower surface of the disk and the pressure acting on the inner disk radius. Correct selection of values resulted in a good fit for one disk height and one flow direction, new values were required if either were changed. To get a good fit for a range of disk heights the values would have to be selected to provide the best match for a simulation. If a simulation had to be run for each valve height anyway calculation of the force is unnecessary, a lookup table can be used. Therefore, this method of force calculation was not taken forward.

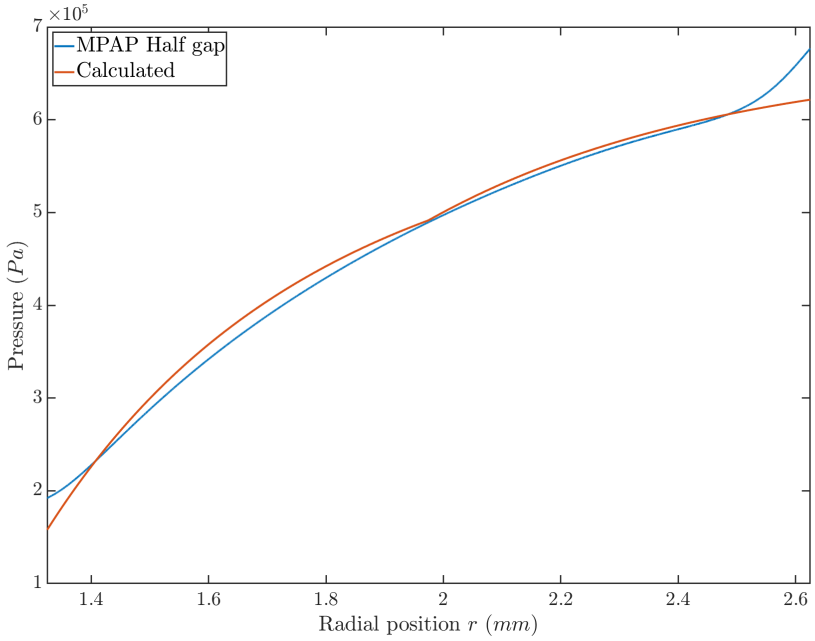


Figure 3.4: Half gap pressure vs radial position. Calculated using Equation 3.4 compared to simulation value for the outflow case.

3.1.2 Buckingham π non-dimensionalization

As theoretical calculations were found to provide poor results a lookup table-based method was considered. Buckingham π non-dimensionalization is a method that would allow a minimum number of simulations to be run and provide a representative sample for similar valves and fluids [12]. For the test valve a range of simulations could be carried out and a look-up table constructed, the non-dimensionalization allows this look-up table to also apply to other valves.

Pi parameters

To perform a Buckingham π non-dimensionalization for force and flow rate the independent parameters must be listed. These are the parameters that, if changed, would affect the force or flow rate. In this case three parameters define the fluid properties and the applied pressure; a single geometry parameter is required for flow rate and three geometry parameters for force. All parameters contain only three fundamental units: mass, length, and time. This means that force can be reduced to three non-dimensional parameters (Equation 3.9) and flow rate can be reduced to a single non-dimensional parameter (Equation 3.10).

$$F_{disk} = F(P, \rho, \nu, h, r_t, r_d) \tag{3.7}$$

$$A = 2\pi r_t h \qquad\qquad Q = Q(P, \rho, \nu, A) \tag{3.8}$$

$$\frac{F_{disk}}{P{r_t}^2} = f\left(\frac{P{r_t}^2}{\rho\nu^2}, \frac{h}{r_t}, \frac{r_d}{r_t}\right) \tag{3.9}$$

$$\frac{Q}{\sqrt{A}\mu} = f\left(\frac{PA}{\mu^2\rho}\right) \tag{3.10}$$

From simulation data the π parameters can be calculated, and then the un-

known functions plotted. A line or curve of best fit will allow un-simulated results to be interpolated. Steady state simulations of the disk valve are used to illustrate this below. Flow rate fits a curve well at low disk height values but is poor for the highest height and resultant larger flow rate, Figure 3.5. Force fits a curve when the two remaining π parameters are constant (Figure 3.6) but not when they are varied (Figure 3.7).

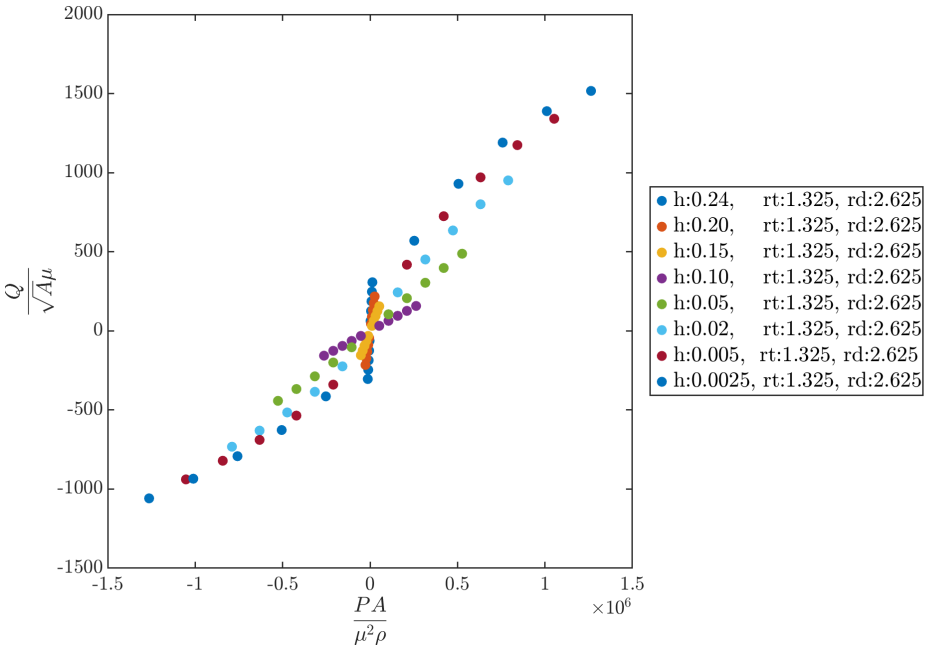


Figure 3.5: Flow rate Buckingham π terms, disk valve. Small disk heights provide good results, poor fit as large disk heights where large flow rates develop.

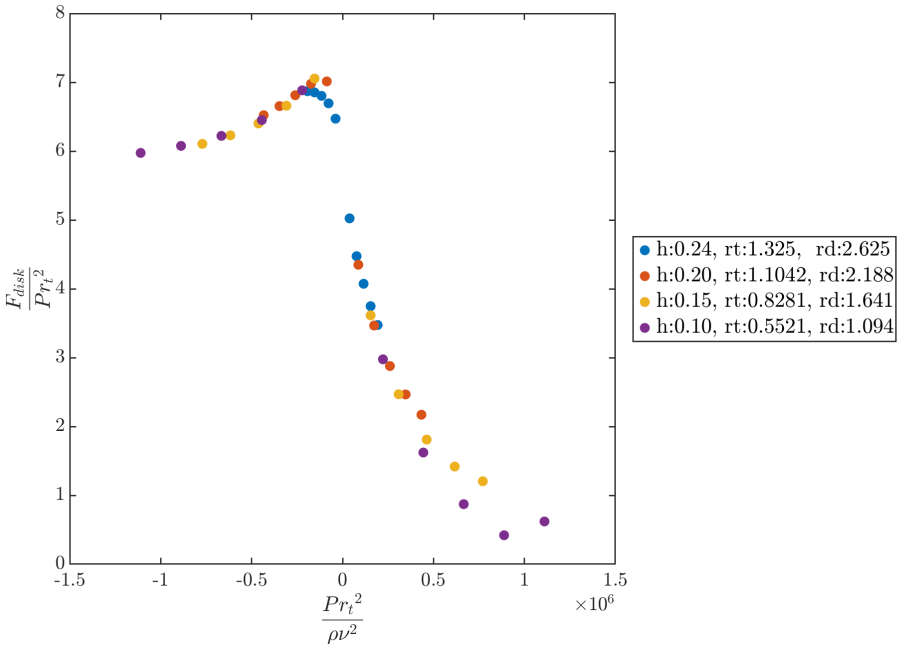


Figure 3.6: Force Buckingham π terms, $\frac{h}{r_t}$ and $\frac{r_d}{r_t}$ constant, disk valve.

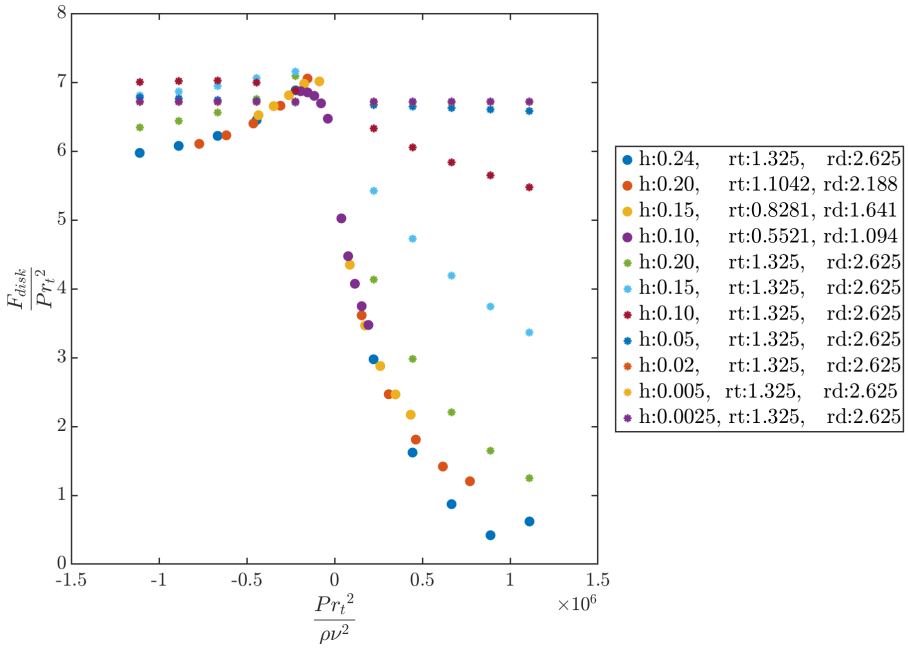


Figure 3.7: Force Buckingham π terms,$\frac{r_d}{r_t}$ constant, disk valve.

3.2 Physical phenomena

The naive reduced order model has two very simple equations. $\bar{F}$ and $\bar{Q}$ represent look-up tables based on Buckingham π theory as described in the previous section. Equation 3.11 was split into two first order differential equations for disk height. Those equations where then solved using MATLABs ODE45 tool [13]. As shown in Figures 3.8 and 3.9 the force and therefore the disk height are a poor match for MPAP. The force is too large causing the disk to move too fast and shut much sooner than the simulated valve. The flow rate shown in Figure 3.10 is also a poor match, it jumps to the steady state value rather than starting at zero.

$$\bar{F}(P,h) - m_{disk}\dot{v} = 0 \tag{3.11}$$

$$Q = \bar{Q}(P,h) \tag{3.12}$$

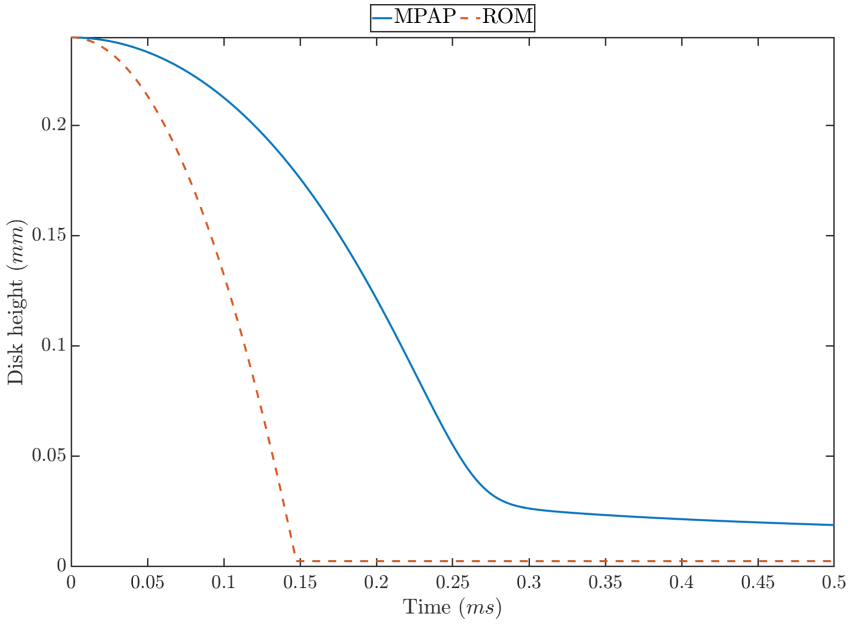


Figure 3.8: Disk height naive model

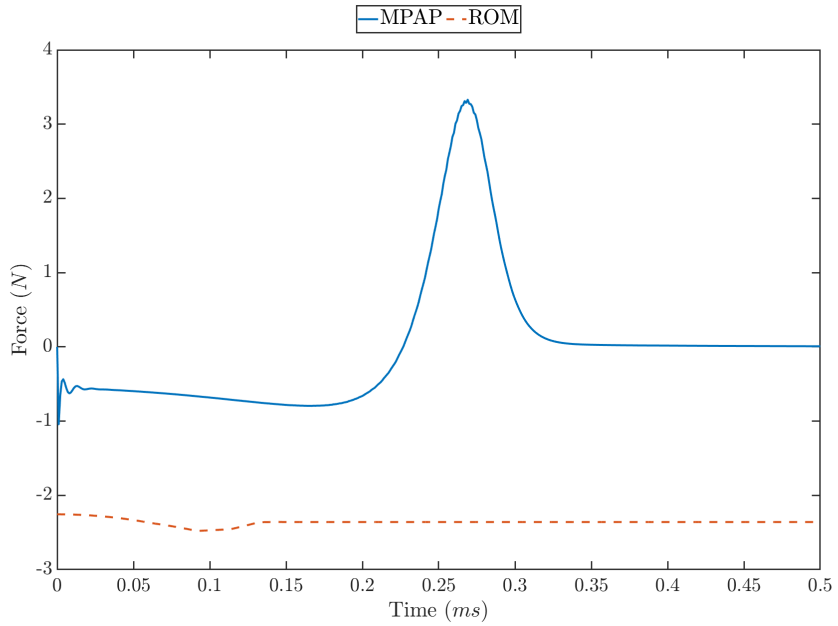


Figure 3.9: Disk force naive model

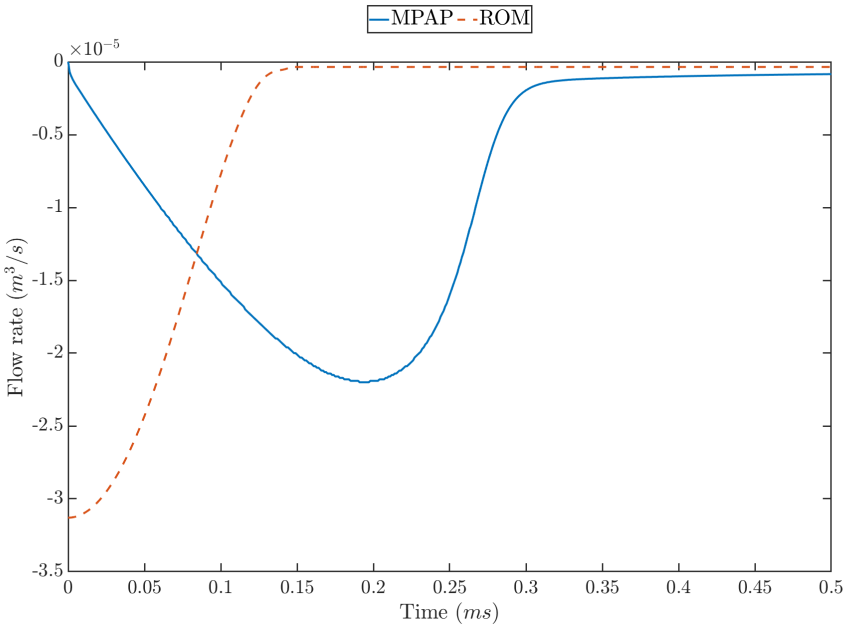


Figure 3.10: Flow rate naive model

3.2.1 Added mass

When the disk is moved some fluid will also have to be moved out of the way and into the void left behind. This is called added mass. In the case that the disk was as wide as the tube it travelled in no fluid could get past the disk, thus the entire fluid would have to be accelerated with the disk. The opposite of this is the disk in free space which has a added mass coefficient of $C_a = \frac{2}{\pi}$ [14]. This coefficient is multiplied by the mass of a sphere of equal radius to the disk with the density of the surrounding fluid. The given value is based on theoretical calculations, it presents a lower limit in this case due to the proximity of the valve walls. The equation of motion can be updated to take this into account (Equation 3.14). This updated equation of motion applies the force to both the disk mass and the added mass. In order for a direct force comparison with MPAP the force acting on only the disk must be found, see Equation 3.15. Equation 3.11 then becomes Equation 3.16. Correct selection of the added mass coefficient results in an improved disk height fit, see Figure 3.11, force and flow rate are still a poor fit Figures 3.12, 3.13. The step change in force is due to the removal of the added mass when the disk becomes stationary.

$$m_{added} = C_a \rho \left(\frac{4}{3}\right) \pi {r_d}^3 \tag{3.13}$$

$$\bar{F}(P,h) - (m_{disk} + m_{added})\dot{v} = 0 \tag{3.14}$$

$$F_{disk} = \bar{F}(P,h)\frac{m_{disk}}{m_{disk} + m_{added}} \tag{3.15}$$

$$\bar{F}(P,h)\frac{m_{disk}}{m_{disk} + m_{added}} - m_{disk}\dot{v} = F_{disk} - m_{disk}\dot{v} = 0 \tag{3.16}$$

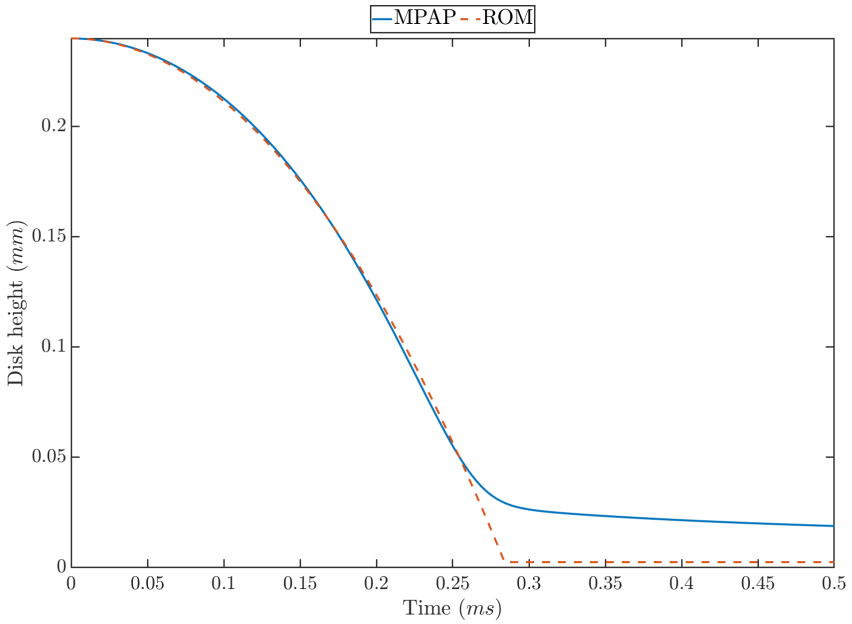


Figure 3.11: Disk height, added mass model

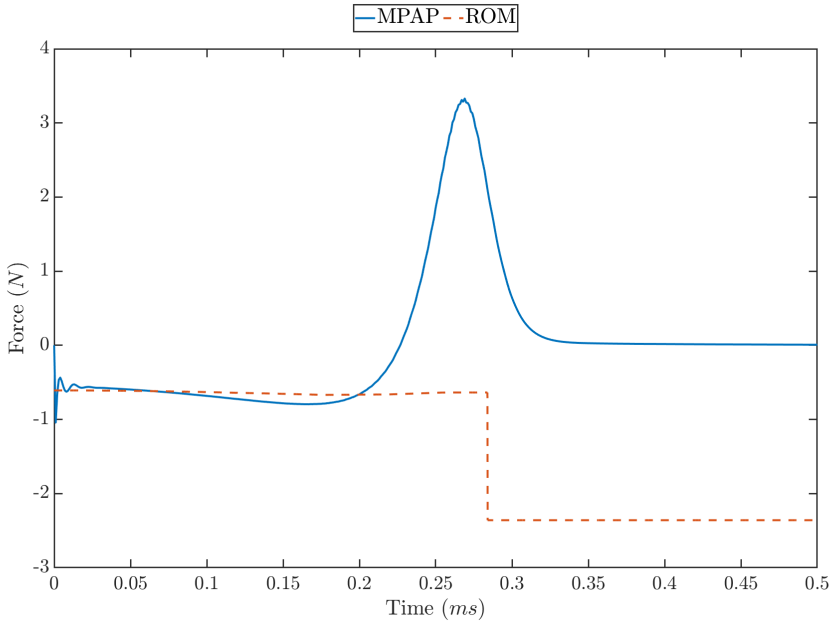


Figure 3.12: Disk force, added mass model

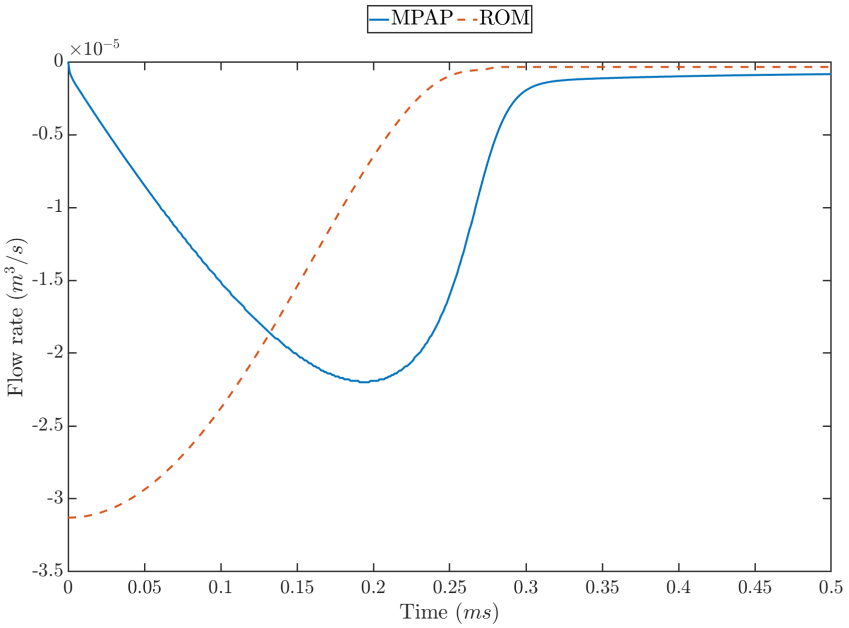


Figure 3.13: Flow rate, added mass model

3.2.2 Damping

If under zero flow rate the disk was manually moved there would be a resistance to movement. This is damping, the disk-seat-system is acting like a dashpot. Unlike an ideal dashpot the geometry of the valve changes significantly as the disk moves towards the seat. After a little trial and error the effect of the changing geometry was found to be proportional to $\frac{1}{h^3}$, this is the 'squeezing' effect as fluid must be quickly forced out the gap region as it becomes increasingly small. This is again included in the force Equation 3.17. The damping coefficient c is chosen for best fit, this results in improved force and disk height fit, see Figures 3.15 and 3.14. Flow rate remains a poor fit, Figure 3.16.

$$F_{disk} = \bar{F}(P,h)\frac{m_{disk}}{m_{disk}+m_{added}} - vc\frac{1}{h^3} \tag{3.17}$$

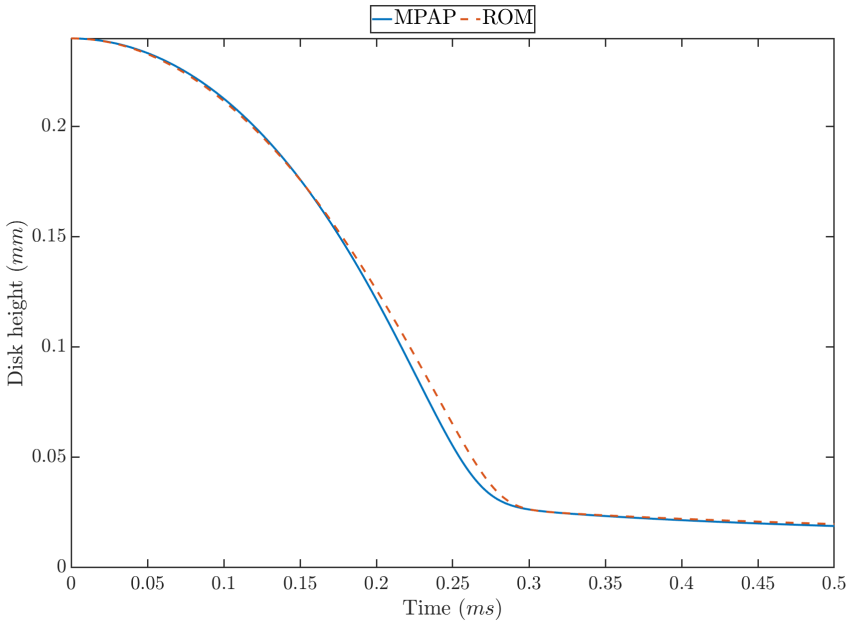


Figure 3.14: Disk height, damping model

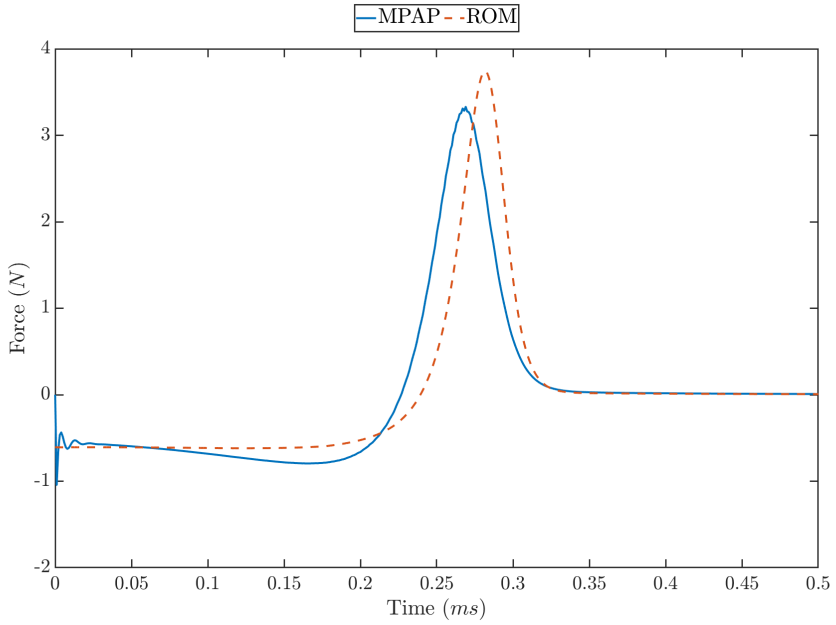


Figure 3.15: Disk force, damping model

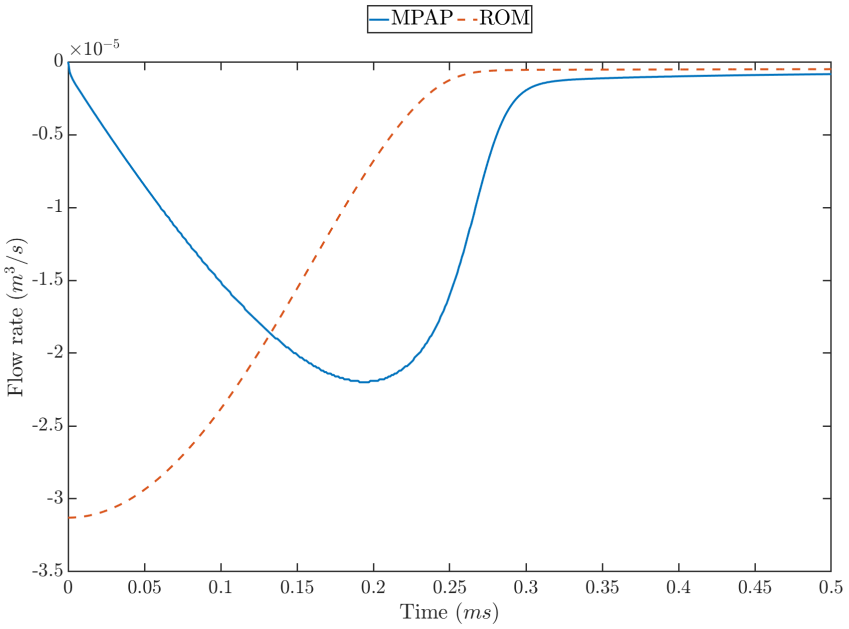


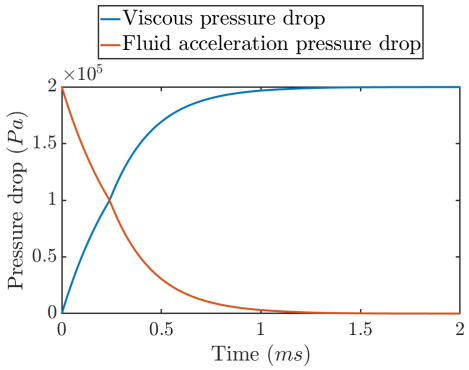
Figure 3.16: Flow rate, damping model

3.2.3 Fluid inertance

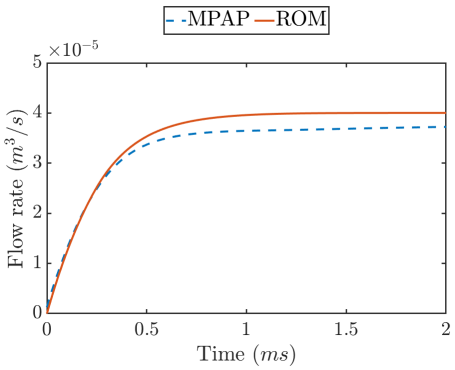
To improve the flow rate results the fluid mass must be accounted for. Just as with solids the fluid must start at zero flow rate and accelerate proportionally to its mass. Fluids however may not be moving at a uniform velocity, so it is much harder to calculate their inertance. To make the comparison easier a range of MPAP simulations were done with the disk at a fixed height, this allows the reduced order model to be simplified. The look-up table for flow rate is inverted such that $\bar{Q}(P,h)$ becomes $\bar{P}(Q,h)$. The change in flow rate can then be calculated, Equation 3.18. This is then solved as the third equation in the MATLAB ODE45 tool. Figure 3.17a shows that the acceleration and viscous pressure drop always add up to the total pressure drop applied at the boundary, $200kPa$ in this case. The flow rate shown in Figure 3.17b is initially a good fit but reaches steady state much faster than the simulation. This is due to the

formation of the large vortex as seen in Figure 2.5. This vortex causes a large increase in the fluid inertance that is not accounted for in the reduced order model.

$$\dot{Q} = \frac{P - \bar{P}(Q,h)}{I} \tag{3.18}$$



(a) Relationship between viscous and acceleration pressure drop



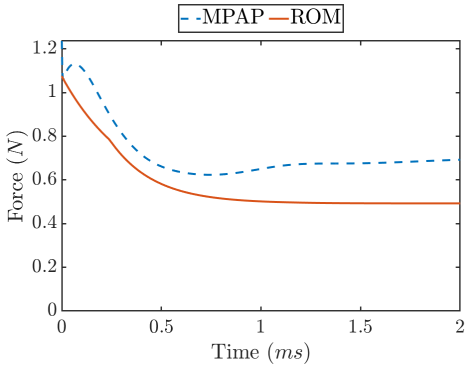
(b) Flow rate

Figure 3.17: Fluid inertia, fixed disk at 0.24mm

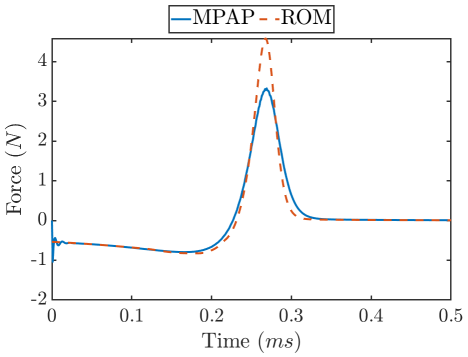
The flow rate also influences the force acting on the disk. This is due to the developing flow causing a pressure to change. The force however does not start at zero, it may be more or less than its steady state value. It was found a good approximation for this initial force is the average between the magnitude of the steady state force at the pressure being tested and its negative. This results in Equation 3.19. The force is then scaled by the ratio of $\bar{P}(Q,h)$ to P so the force reaches its steady state value at the same time as the flow rate. F_{disk} therefore becomes Equation 3.20. This results in an improved force fit for both the static and dynamic valve, Figure 3.18 a and b. The disk height is still a good fit (Figure 3.19) and the flow rate is much improved (Figure 3.20).

$$F_{inital} = \frac{|\bar{F}(P,h)| + |\bar{F}(-P.h)|}{2} sign(\bar{F}(P,h)) \tag{3.19}$$

$$F_{disk} = \left(F_{initial} \left(1 - \frac{\bar{P}(Q,h)}{P} \right) + \bar{F}(P,h) \frac{\bar{P}(Q,h)}{P} \right) \frac{m_{disk}}{m_{disk} + m_{added}} - vc\frac{1}{h^3} \quad (3.20)$$



(a) Fixed disk at 0.24mm



(b) Moving disk, closing

Figure 3.18: Force on disk with fluid inertia term

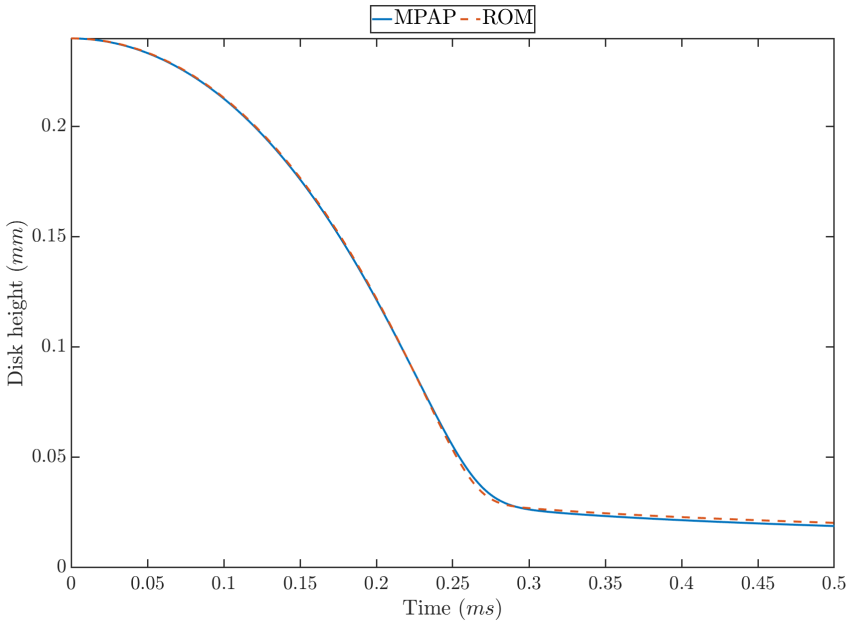


Figure 3.19: Disk height, fluid inertia model

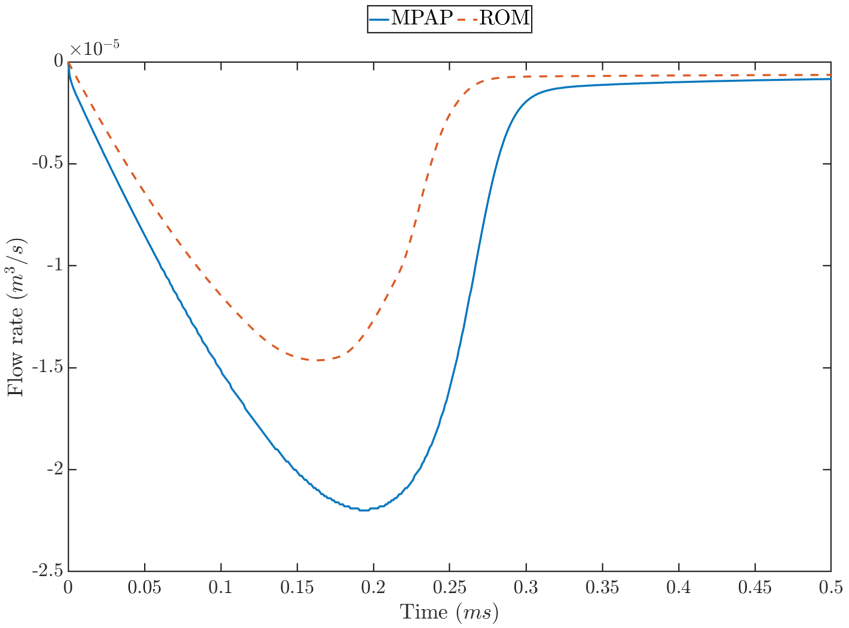


Figure 3.20: Flow rate, fluid inertia model

3.2.4 Displacement flow rate

If under no pressure the disk was moved it would not only need to overcome the damping of the fluid but would cause a resultant flow rate through the valve. This flow rate will have the same velocity as the disk but may not have the same area. Not all the fluid moved by the disk will exit the domain, some may circulate internally, Figure 3.21. The area of the displacement flow rate is defined by an effective radius r. The inertia of this fluid mass is accounted for by the added mass. The force it causes is accounted for as the damping force. The displacement flow rate is therefore simply added to the previously calculated flow rate outside of the ODE45 solver. It is assumed the displacement flow rate has no effect on force or disk movement. The addition of the displacement flow rate further improves the flow rate in both opening and closing tests. Once the opening valve reaches the end of its travel the jump in flow rate is replicated.

$$Q_{displacment} = v\pi r_d^2 \qquad Q_{total} = Q + Q_{displacment} \tag{3.21}$$

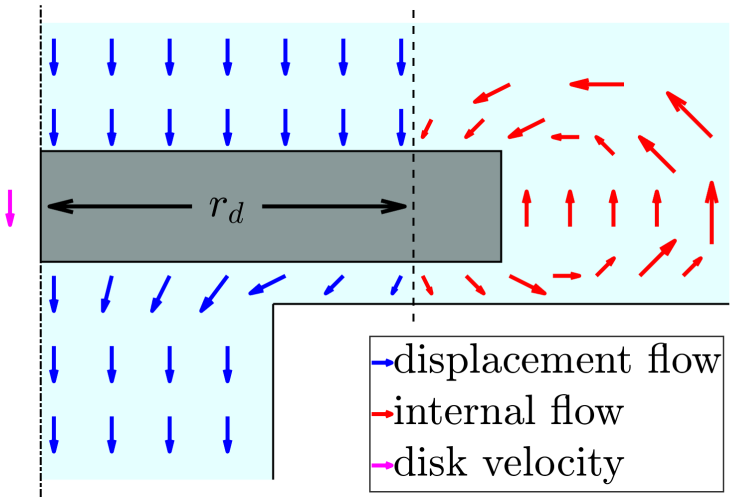
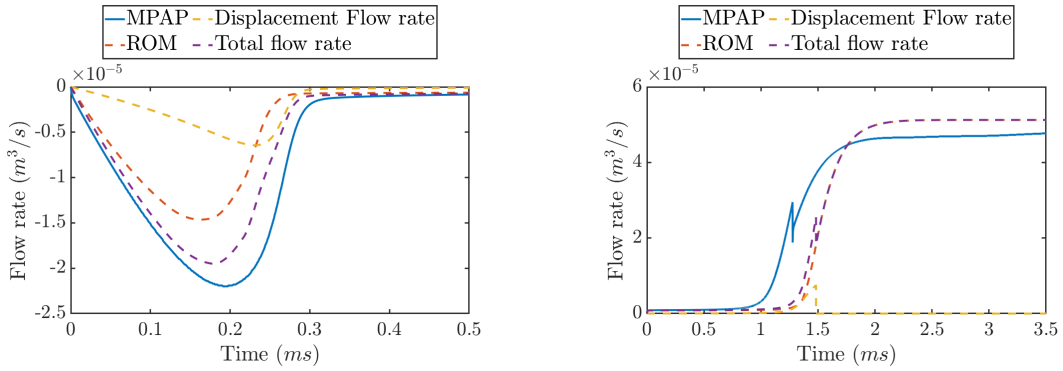


Figure 3.21: Displacement flow rate



(a) Opening (b) Closing

Figure 3.22: Displacement flow rate

3.3 The reduced order model

The final reduced order model incorporates all the effects above to provide a close match for MPAP results. The model requires a look-up table for force and flow rate based on steady state simulations. Five variables must be defined to allow best fit. These values have been chosen to provide the best fit to MPAP dynamic simulations.

$$\dot{h} = v$$

$$\dot{v} = f_1(P, h, v, Q) \qquad f_1(P, h, v, Q) = \frac{f_3(P, h, v, Q)}{m_{disk}}$$

$$\dot{Q} = f_2(P, h, Q) \qquad f_2(P, h, Q) = \frac{P - \bar{P}_v(Q, h)}{I}$$

$$F_{disk} = f_3(P, h, v, Q)$$

$$f_3(P, h, v, Q) = \left(F_{initial} \left(1 - \frac{\bar{P}(Q, h)}{P} \right) + \bar{F}(P, h) \frac{\bar{P}(Q, h)}{P} \right) \frac{m_{disk}}{m_{disk} + m_{added}} - vc \frac{1}{h^3}$$

$$F_{inital} = \frac{|\bar{F}(P, h)| + |\bar{F}(-P.h)|}{2} sign(\bar{F}(P, h))$$

$$m_{added} = C_a \rho \left(\frac{4}{3} \right) \pi {r_d}^3$$

$$Q_{total} = Q + Q_{displacment} \qquad Q_{displacment} = v \pi r_d^2$$

$$C_a = 3.6 \qquad c = 2.74752e{-}13 \qquad I = 1.4125e6 \frac{kg}{m^4} \qquad r_d = 1.2mm$$

3.4 An alternative derivation

An alternative derivation that may be more intuitive to the reader was also found. This approach splits the force required to move the disk into a solid and a fluid force. That is the force to move the mass of the disk itself and the force required to move the fluid out of the disk's way.

Total force is the sum of that acting on the fluid and solid.

$$F_{total} = F_{fluid} + F_{solid} \tag{3.22}$$

Solid acceleration applies the solid force to only the solid mass.

$$a = \frac{F_{solid}}{m_{disk}} \tag{3.23}$$

Fluid acceleration applies the fluid force to the fluid, including the damping term.

$$\dot{Q} = \frac{\left(\frac{F_{fluid} - F_{steady}}{\pi {r_d}^2}\right)}{I_{disp}} = \frac{F_{fluid} - F_{steady}}{I_{disp} \pi {r_d}^2} \qquad F_{steady} = vc \frac{1}{h^{c_{power}}} \tag{3.24}$$

$$\dot{Q} = a\pi {r_d}^2 = \frac{F_{solid}}{m_{disk}} \pi {r_d}^2 \tag{3.25}$$

The two $\dot{Q}$ equations are then equated and $F_{total} - F_{solid}$ substituted in for F_{fluid}.

$$\frac{F_{total} - F_{solid} - F_{steady}}{I_{disp} \pi {r_d}^2} = \frac{F_{solid}}{m_{disk}} \pi {r_d}^2 \tag{3.26}$$

This can then be solved for F_{solid}.

$$F_{solid} = (F_{total} - F_{steady}) \frac{m_{disk}}{m_{disk} + I_{disp}\pi^2 {r_d}^4} \qquad F_{steady} = vc\frac{1}{h^{c_{power}}} \tag{3.27}$$

This approach replaces original added mass term with displacement inertance. This can be seen by substituting Equation 3.15 and Equation 3.27.

$$m_{added} = I_{disp}\pi^2 {r_d}^4 \tag{3.28}$$

3.5 Results

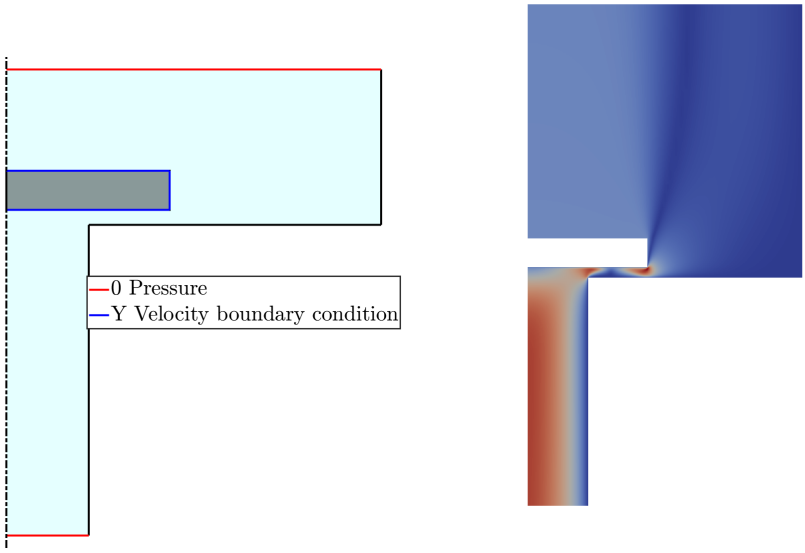
3.5.1 Obtaining parameters from fixed simulations

For the reduced order model to match the MPAP results five variables are required: Inertance I, Added mass coefficient C_a, Damping factor c, Damping power c_{power} and displacement radius r_d. It is desirable to obtain these values from non fluid structure interaction simulations. This removes the need to run full fluid structure interaction simulations to give benchmarks for manual parameter selection. This chapter outlines two simulations that allow all the required variables to be derived.

Added mass, damping force and displacement radius

In order to calculate the force and flow rate caused by the disk's movement a steady state simulation can be used to approximate the resultant fluid flow. Zero pressure is applied at the top and bottom of the domain and a constant velocity boundary condition applied to the disk for both the opening and closing case (Figure 3.23a). This is a snapshot of the disk moving through the domain,

however it is non-physical as the disk does not move. The flow would never reach the steady state simulated here due to the movement of the disk. It was found however that the steady state solution provides good results when used in the reduced order model. A range of disk heights were run at a range of boundary condition velocities. These velocities were chosen to cover the range of disk velocities seen in the dynamic simulation.



(a) Boundary conditions (b) Fluid velocity

Figure 3.23: Velocity boundary condition simulation

The damping force can then be found directly from the force on the disk. The force is shown in Figure 3.24, points show the force from the simulation and lines show the fit of the equation used in the reduced order model with the parameters calculated for the best fit. Although the equation provides a good fit if sufficient data points are gathered a look up table can be substituted for an exact match.

$$F_{damping} = -vc\frac{1}{h^{c_{power}}} \qquad c = 4.7734e-12 \qquad c_{power} = 2.838 \tag{3.29}$$

$$F_{damping} = \bar{F_{damping}}(h, v) \tag{3.30}$$

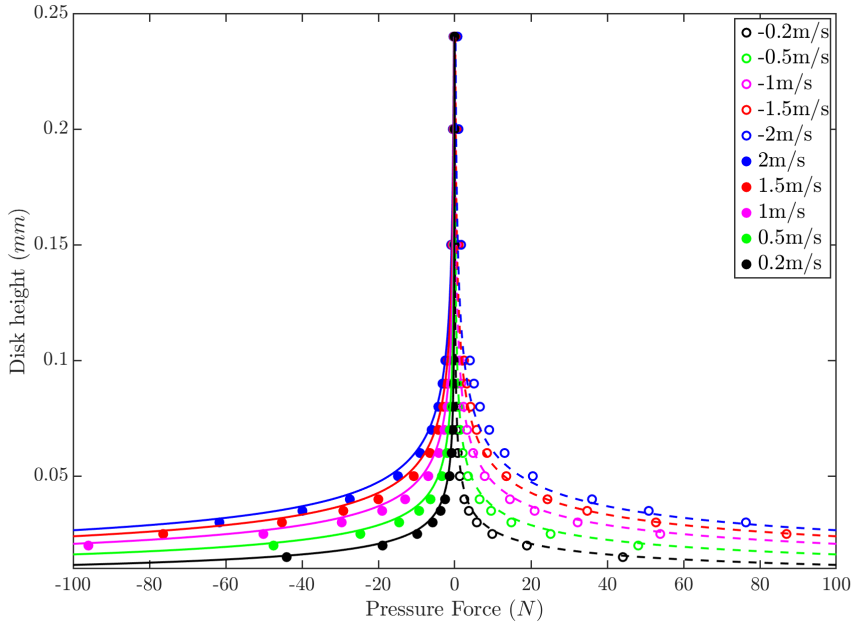


Figure 3.24: Damping force vs disk height. Points show simulations, lines show the fit resulting from Equation 3.29. The left hand side of the graph simulates the disk moving away from the seat as the value opens and the right hand side simulates the valve closing as the disk moves toward the seat.

The flow rate out of the domain caused by the velocity boundary condition can be used to calculate the displacement radius. The radius can be clearly seen in Figure 3.23b, the split between fluid flowing through the inlet and round the disk is clear. This split location is the displacement radius. It was found that a constant value of radius provides a good fit, however in this case as all the data had been collected a look up table was used (Figure 3.25).

$$Q_{displacment} = v\pi r_d^2 \qquad r_d = \sqrt{\frac{Q}{v\pi}} \qquad r_d = 1.91864mm \tag{3.31}$$

$$r_d = \bar{r_d}(h, v) \tag{3.32}$$

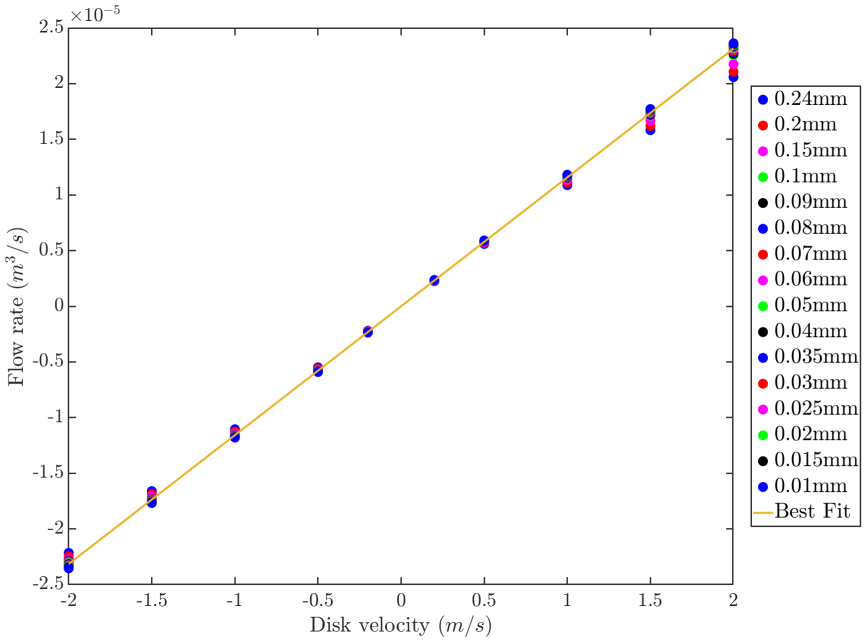


Figure 3.25: Displacement flow rate vs disk velocity

The added mass coefficient can then be calculated from the total kinetic energy of the fluid. The added mass is the mass that when moved at the same speed as the applied boundary condition velocity would provide the same kinetic energy. In this case the results found do not fit a constant value so either a one dimensional look up table with respect to height can be used or a two dimensional look up table with respect to both height and velocity, Figure 3.26.

$$E = \frac{1}{2}\rho\pi \iint_D r {v_{magnitude}}^2 \, dy \, dr \qquad m_a = \frac{2E}{v^2} \qquad Ca = \frac{m_a}{\frac{4}{3}\rho\pi {r_d}^3} \tag{3.33}$$

$$C_a = \bar{C_a}(h) \qquad\qquad C_a = \bar{C_a}(h, v) \tag{3.34}$$

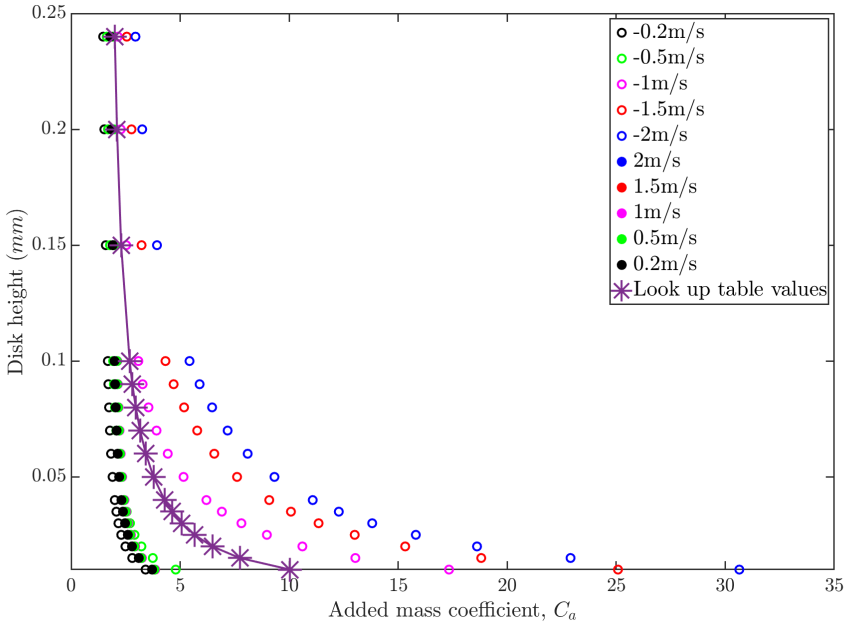
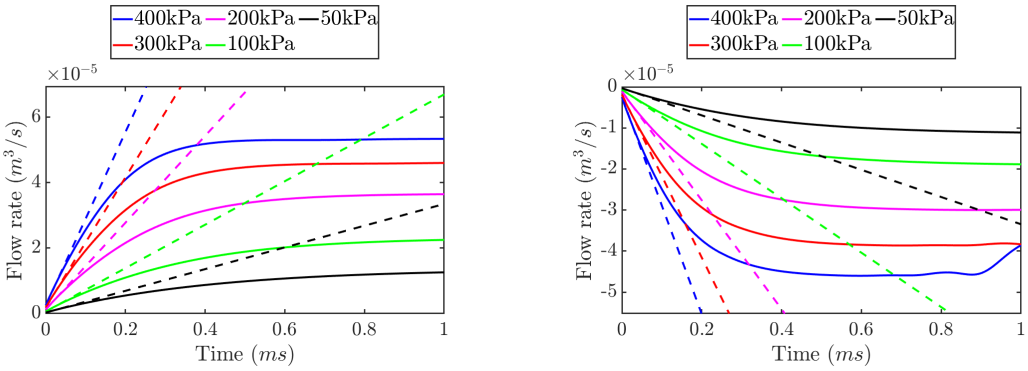


Figure 3.26: Added mass coefficient vs disk height

Inertance

Inertance can be calculated in two ways. The first method requires a fixed geometry transient simulation. This is the same simulation as run for the initial force calculations, an instantaneous constant pressure differential applied across the domain with the disk held in a fixed location. The inertance is then calculated from the change in flow rate, this is done by finding the initial gradient of the flow rate (Figure 3.27). The average inertance was calculated to be $1508200\frac{kg}{m^4}$.

$$I = \frac{P}{\dot{Q}} \tag{3.35}$$



(a) valve opening (b) valve closing

Figure 3.27: Flow inertance initial gradient method

The second method is to calculate the steady state inertance from a steady state simulation. This has the advantage of utilising simulations that have been run anyway for force calculation and eliminating the need to run transient simulations. The resulting steady state inertance is less applicable to modelling dynamic behaviour but has still been shown to provide good results.

Figure 3.28 shows a comparison between the two methods. Each method results in different values with a different trend, but both provide values of a similar scale. The steady state inertance is larger at larger disk heights due to the formation of a vortex. As a wide range of data is available a single inertance value can be substituted for a look up table.

$$E = \frac{1}{2} I Q^2 \qquad I = \frac{2E}{Q^2} \tag{3.36}$$

$$I = \bar{I}(h, v) \tag{3.37}$$

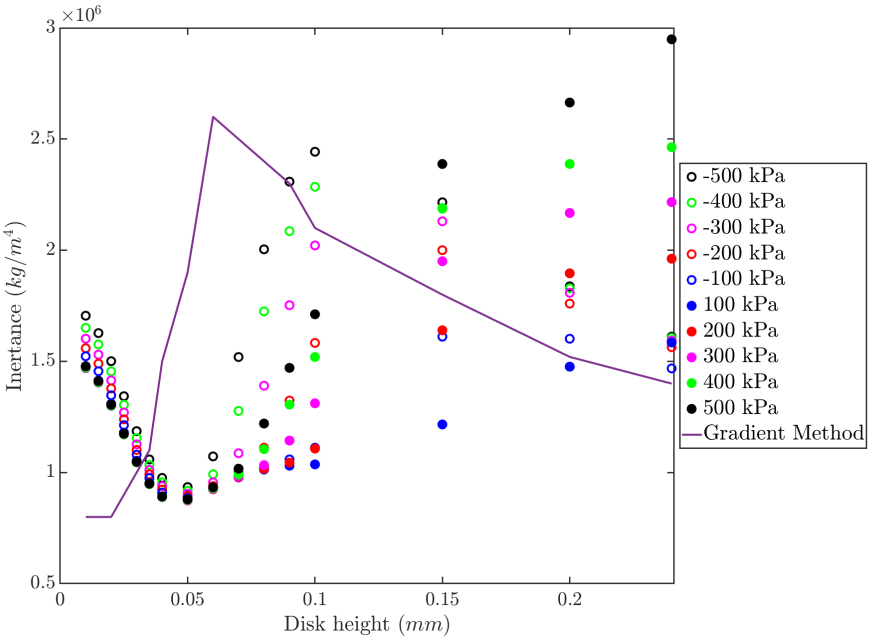


Figure 3.28: Flow inertance calculation methods comparison

Complete reduced order model

The lookup tables for model variables derived in the preceding section are then substituted into the reduced order model, resulting in the final formulation.

$$\dot{h} = v$$

$$\dot{v} = f_1(P,h,v,Q) \qquad f_1(P,h,v,Q) = \frac{f_3(P,h,v,Q)}{m_{disk}}$$

$$\dot{Q} = f_2(P,h,Q) \qquad f_2(P,h,Q) = \frac{P - \bar{P}_v(Q,h)}{\bar{I}(h,v)}$$

$$F_{disk} = f_3(P,h,v,Q) \qquad f_3(P,h,v,Q) = F_{solid}$$

$$F_{inital} = \frac{|\bar{F}(P,h)| + |\bar{F}(-P.h)|}{2} sign(\bar{F}(P,h))$$

$$F_{total} = F_{initial}\left(1 - \frac{\bar{P}(Q,h)}{P}\right) + \bar{F}(P,h)\frac{\bar{P}(Q,h)}{P}$$

$$F_{solid} = \left(F_{total} - \bar{F}_{damping}(h,v)\right)\frac{m_{disk}}{m_{disk} + \bar{I}_{disp}(h,v)\pi^2 {r_d}^4}$$

$$Q_{total} = Q + Q_{displacment} \qquad Q_{displacment} = v\pi\bar{r}(h,v)^2$$

3.5.2 Opening and closing the valve

This finalised reduced order model was then tested against the original MPAP results as showing in Figures 3.29 and 3.30. The new method of deriving parameters from simulations results in a fit as good as that achieved when selecting parameters manually for best fit (also shown in Figures 3.29 and 3.30). This

shows that the reduced order model can be fully derived from steady state and transient simulations with fixed geometry. There is no requirement to run fluid structure interaction simulations where the disk is moved by the fluid.

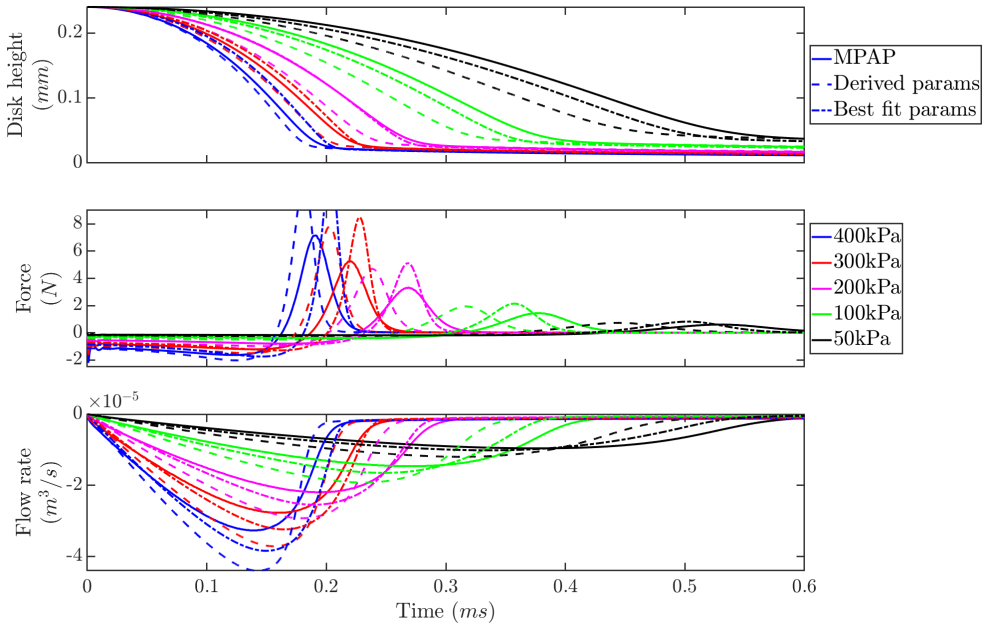


Figure 3.29: Updated model closing. Parameters derived from simulations vs those manually selected for best fit.

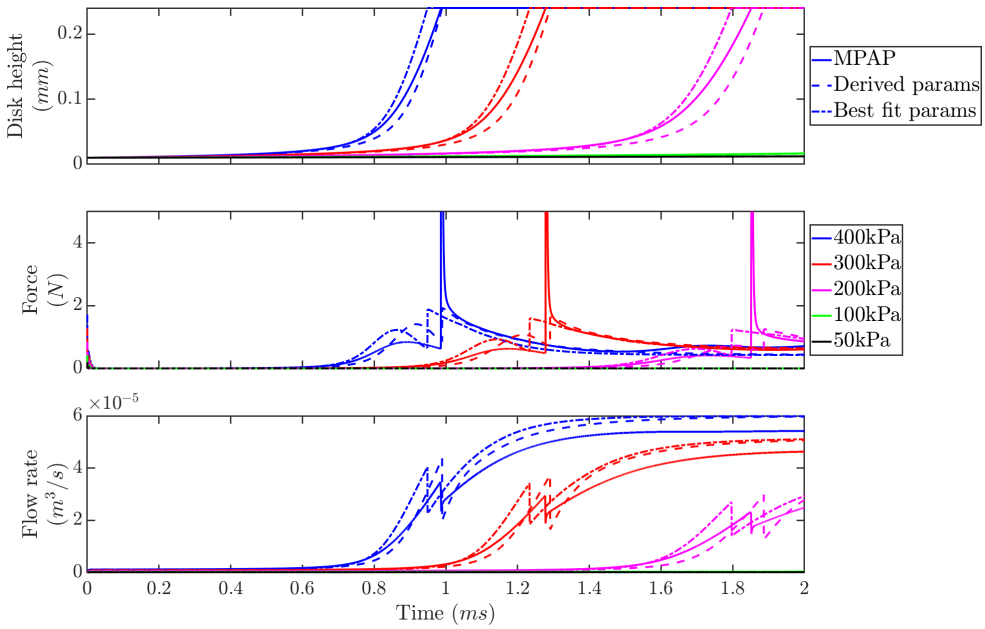


Figure 3.30: Updated model opening. Parameters derived from simulations vs those manually selected for best fit.

3.6 Limitations

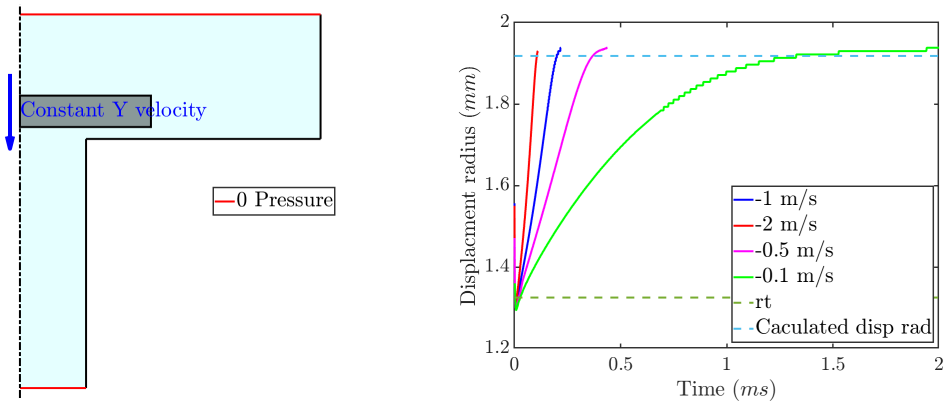
There are several limitations to this approach, it is possible to get a good fit using only fixed geometry simulations. However, a larger number of steady state and transient simulations must be run. Each simulation requires different boundary conditions to be set, this is a significant effort. More complex transient behaviour is missed completely from the steady state simulations that make up the majority of the lookup tables. Several methods for improving the reduced order model were considered but none provided satisfactory results.

No formulation could be found to include spring force holding the valve shut. The increased opening force resulting from the addition of a spring also increases the flow rates and velocities in the values, these moves the flow further from the steady state data set the reduced order model largely based on.

3.6.1 Displacement flow rate

An improvement would be to consider the interaction of flow rate and displacement flow rate and the resulting effect on inertance and added mass. An investigation was done to see if the steady state values for displacement flow rate represent the dynamic behaviour. Transient simulations where the disk was moved at a constant speed found that the displacement radius develops with time, Figure 3.31. This is because initially it is much easier for the flow to move around the disk than out of the domain. The steady state displacement radius is correct when the flow is only experiencing viscous pressure drop because it has stopped accelerating. Due to the time it takes for this flow to develop the steady state displacement radius value is incorrect when it has the largest effect, a fast moving disk.

So far, a way to model this effect accurately has not been found, using a value closer to throat radius does improve the flow rate results, but this moves away from the methodology outlined in the previous section for deriving parameters.



(a) Boundary conditions (b) Displcment radius with time

Figure 3.31: Displacement radius investigation by moving disk a constant velocity

3.6.2 Ball valve without spring

To further test the reduced order model the ball valve was considered. This valve was run in opening and closing conditions as a benchmark. The reduced order model is based on a total of 198 steady state simulations. The parameters for the reduced order model are derived as outlined in the previous section.

The reduced order model was then compared to MPAP for opening and closing simulations. Figure 3.33 and 3.32 show closing and opening respectively. Flow rate is a much poorer fit than seen for the disk valve, which is due to the large inertance values calculated from the steady state simulations. The large vortex seen in Figure 2.13 results in a large inertance value, as this vortex takes a significant time to develop. The valve opens before the vortex has formed and therefore experiences a much reduced inertance. Manually selecting an inertance value of one fifth of that calculated from steady state simulations results in a better fit. Results using the updated inertance are also shown in Figures 3.33 and 3.32, flow rate fit is much improved although some oscillation is seen at very small flow rates. This estimated inertance could potentially be replaced with a value derived using the initial gradient method, but that would require running fixed ball transient simulations. Damping effects are also much reduced due to the ball valves much more open seat geometry.

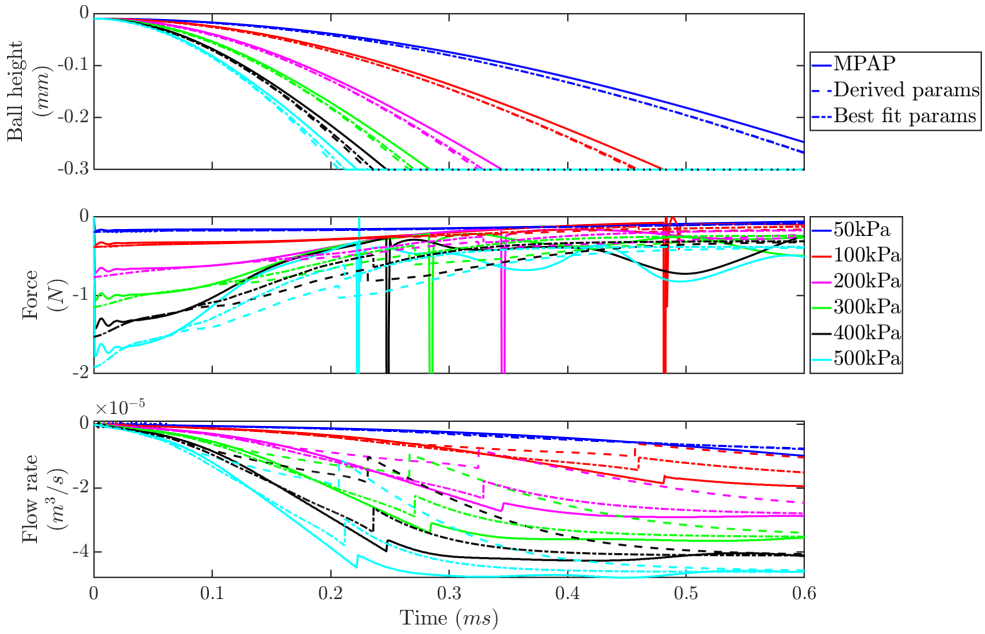


Figure 3.32: Reduced order model opening

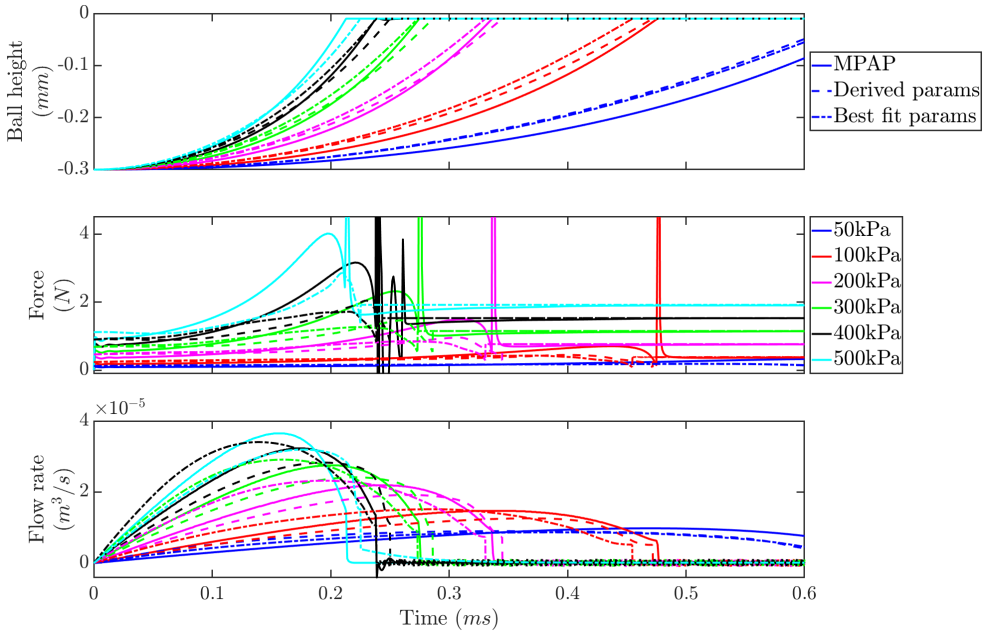


Figure 3.33: Reduced order model closing

Chapter 4

Neural network reduced order model

Having found the limitations of a physics based approach, it was thought that rather than running many steady state simulations it may be better to run a small number of full fluid structure interaction simulations to build a training dataset. Although these take significantly longer to run there is a reduction in the manual setup and post processing time associated with running a range of initial conditions and the unusual boundary conditions presented in section 3.5.1. Neural networks are a method of modelling complex systems based on such a training dataset. They typically require no physical insight, this work includes some insight in order to improve the performance for a given network.

4.1 Neural networks

Networks operate as a black box transfer function. A network is made up of at least one neuron in at least one layer. As the name implies neurons are inspired by those found in nature, taking several inputs, and returning a single output.

The properties of the neuron are tuned by two values: a weight and a bias. Each input value is multiplied by a weight then the bias value added. A simple transfer function is then applied, and the resulting value passed on to the next neuron [15].

Multiple neurons are arranged in a series of layers to make up an interconnected network. Each neuron in a layer is connected to each neuron in the following layer. Where a single neuron receives inputs from several neurons in the preceding layer each connection has a weight, the weighted values are then summed, see Figure 4.1. Networks can be interconnected in more complex ways, for this work only fully connected feed-forward networks were considered. Relatively small networks were found to be sufficient to achieve the goal of this work, larger networks with many layers and more complex structures are loosely collected under the umbrella term "Deep Learning" [16].

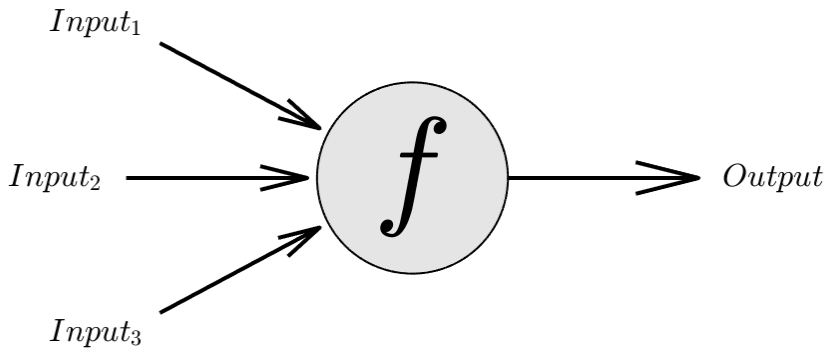


Figure 4.1: Single neuron with three input values

Equations 4.1 to 4.3 can be used to evaluate a feed forward network. Input and output layers are a special case, the layers in-between, if any, are known as hidden layers and there can be any number of these. The output layer applies a linear transfer function, see Figure 4.2.

Input layer: $Input$ is a vector with length of the number of inputs. w_1 is a matrix of size: inputs neurons, hidden layer neurons. Biases b_1 and output a_1 are arrays of length hidden layer neurons.

$$\text{Input layer} \rightarrow a_1 = f\left(b_1 + w_1 Input\right) \tag{4.1}$$

Hidden layer: again b_2 and a_2 are of length of the number of neurons in the layer. w_2 is a matrix of size: neurons in previous layer, neurons in current layer.

$$\text{Hidden layer} \rightarrow a_2 = f\left(b_2 + w_2 a_1\right) \tag{4.2}$$

Output layer: as above, however no transfer function is applied (a linear transfer function of one).

$$\text{Output layer} \rightarrow out = b_3 + w_3 a_2 \tag{4.3}$$

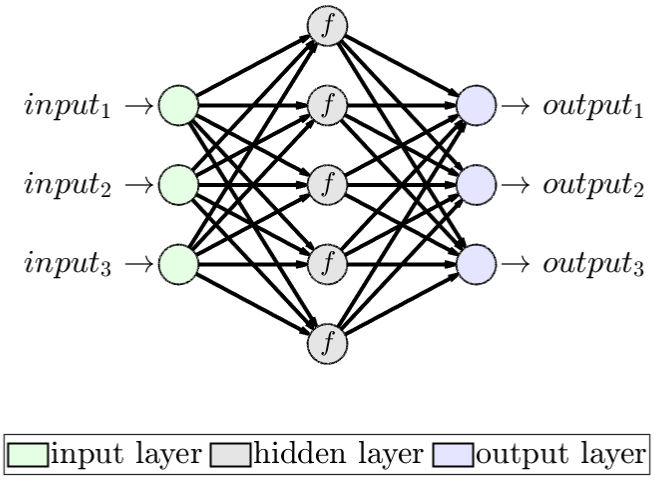


Figure 4.2: Neural network architecture

4.1.1 Transfer functions

The transfer function in each individual neuron is selected to suit the problem and training method, although other transfer functions were experimented with this work focuses on two. Firstly a hyperbolic tangent sigmoid (Equation 4.4), commonly known as tansig. The second transfer function used was the rectified linear unit, commonly known as ReLU (Equation 4.5). Both transfer functions are shown in Figure 4.3.

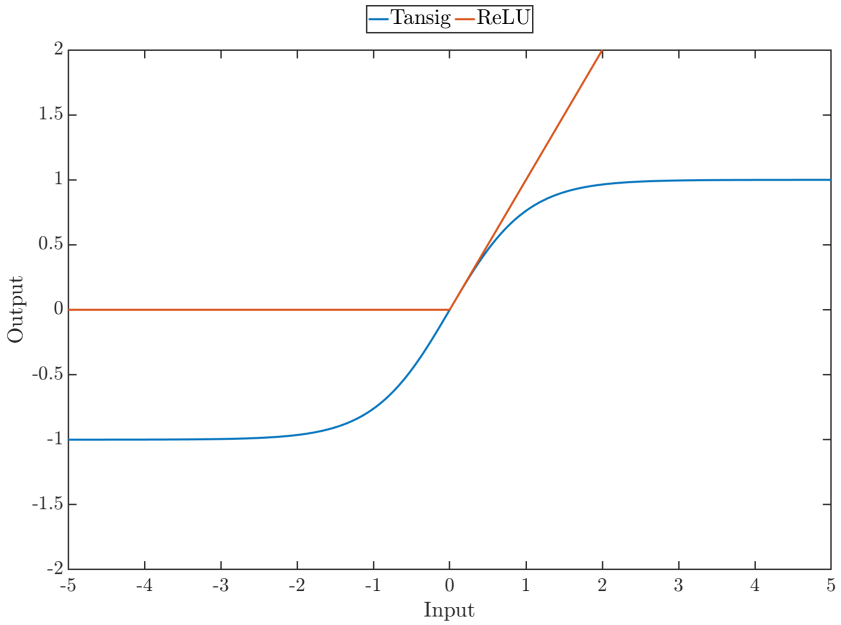


Figure 4.3: Neuron transfer functions

$$tansig\,(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{4.4}$$

$$ReLU\,(x) = max(0, x) \tag{4.5}$$

4.2 The reduced order model

The network itself takes two types of input. Firstly, driving variables, in this case pressure. A more advanced model could have multiple driving variables, fluid temperature for example. These driving variables are equivalent to boundary conditions. The second input type is the known state of the valve. This is ball height, flow rate and integrated kinetic energy of the domain. The rate change of the known state variables are given as the network output. Unlike long short term memory (LSTM) networks outputting the rate change of variables allows (and requires) this model to be used within a external time integration scheme. This architecture is shown in Figure 4.4, for simplicity each variable group is shown as a single input and output neuron, in reality there maybe more than one neuron per group.

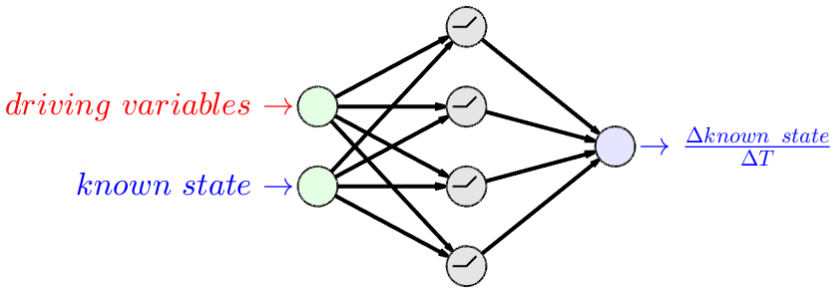


Figure 4.4: Simplified network as used in reduced order model

To use the network within a reduced order model the network is evaluated in a forward Euler integration scheme. This gives a form of continuous-time recurrent neural network [17]. There are several other methods of modelling time series data with networks, however in this case the ability to use variable time step sizes is key for integration into larger system models.

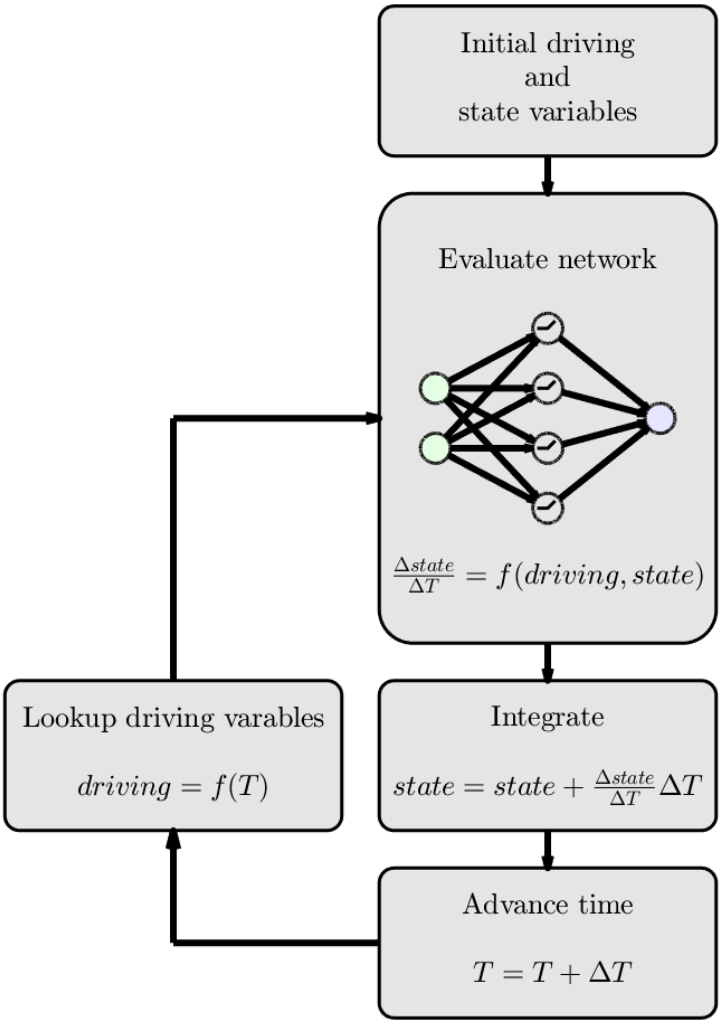


Figure 4.5: Neural network in reduced order model

Algorithm 1 shows the scheme is implemented in pseudocode.

Algorithm 1 Network evaluation and integration scheme

```
procedure EVALUATION(weight, bias, driving_variable, time, initial_conditions)

    state[1, :] = initial_conditions

    for n ← 1, length(time) do
        Dt = time(n+1) - time(n)

        // network input values, including single delta time multiple
        network_inputs = [driving_variable[n], state[n,:]]

        // network input layer with ReLu transfer function
        net_out = max(0, bias[1]+weight[1]*network_inputs)

        // evaluate hidden layers
        for i ← 1, num_layers do
            net_out = max(0, bias[1+i]+weight[1+i] * net_out)
        end for

        // output layer
        net_out = bias[2+num_layers]+weight[2+num_layers]*net_out

        // forward Euler integration
        state[n+1,:] = state[n,:] + net_out * Dt

    end for

    return state
end procedure
```

4.2.1 Unknown State variables

Some of the valves internal state is captured by the known state variables: ball height, flow rate and kinetic energy. However, the valve has some additional state that is not captured by these. Physically one might think of the vorticity might change how the valve responds, after all vortexes are seen in the simulation

results. However, knowing the vorticity value of the fluid is does not add value to the larger system model. Extra known state variables also require extra time to train as they must be included in the cost function. Rather than including extra known state variables states with unknown values are added. These are integrated along with the known state variables giving the network additional ability to track the systems state. We could think of these unknown states as tracking the vorticity, or some other physical property, the reality is not so tangible. The unknown nature of these new states requires that all training simulations start with the same initial conditions, unknown states are initialised at 0. The updated structure is shown in Figure 4.6.

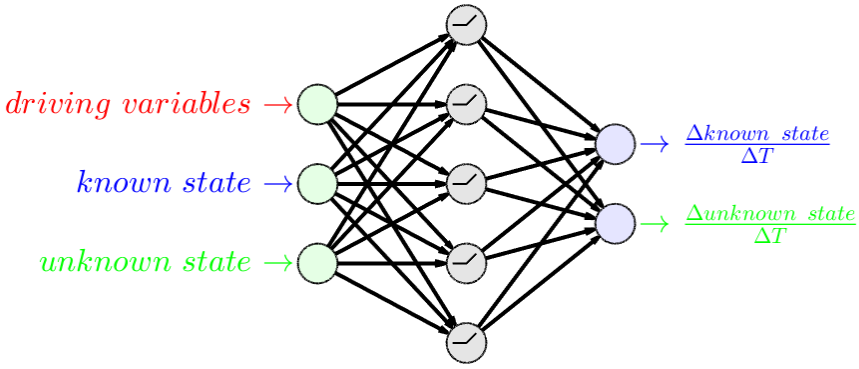


Figure 4.6: Network with unknown states

4.2.2 Delta time multiples

In the valve example one of the known network states is ball height, the corresponding output is therefore ball velocity. The equation for this motion is known:

$$v = v_0 + a_0 \Delta T + \frac{1}{2} j \Delta T^2 \tag{4.6}$$

However, a network with the simple feed forward architecture cannot represent

multiplications of inputs directly. Given enough neurons the network can make good approximations. As we have some insight into the physical nature of this problem, we can provide the inputs we know the network will need. This allows equation 4.6 to be exactly represented by a single neuron with three weights and no transfer function, Figure 4.7.

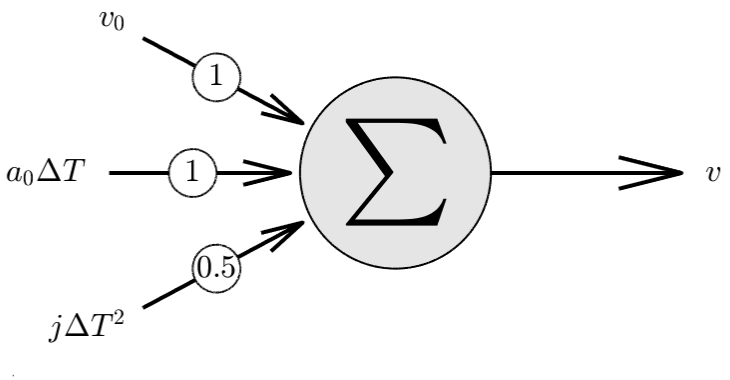


Figure 4.7: Neuron representation of Equation 4.6

In reality we do not have exactly the velocity, acceleration and jerk as known state variables. However, they could be represented in the unknown state variables. As the representation of the unknown states is unknown all states are given a ΔT multiple. Training is relied upon to apply zero weighting to those that are not useful. Any number of ΔT multiples could be used; however, each adds a significant number of weights that must be trained. In practice one or two provide a good middle ground, resulting in a network architecture as shown in Figure 4.8.

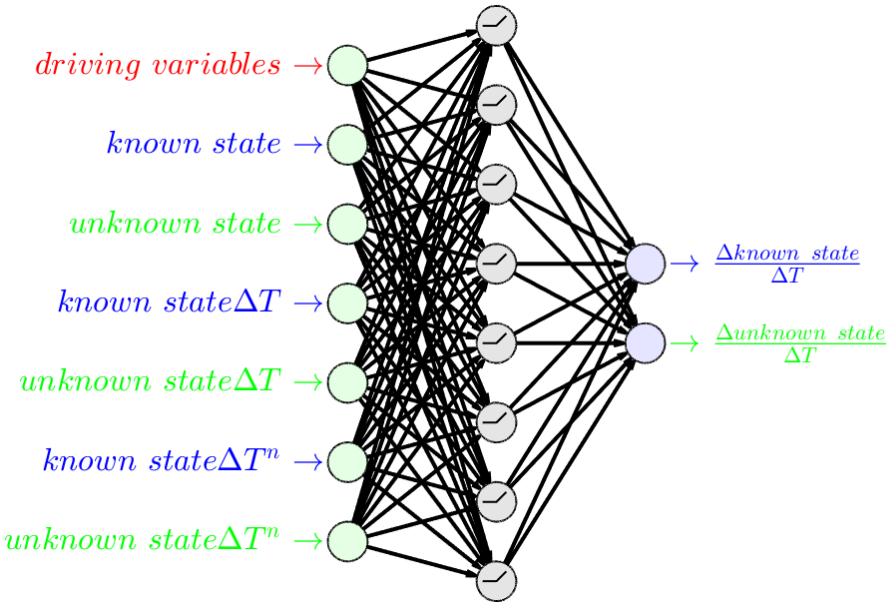


Figure 4.8: Network with unknown states and two delta time multiples

This approach gives the network a "head start" based on insight into the physical system but is not a requirement. A larger network could represent the same equations of motion, however being larger it will also require a longer training period.

4.3 Training

The values of the network's weights and biases must be found such that the network outputs the desired values for given inputs. For even a small network this results in tens or hundreds of design variables. For example the network shown in Figure 4.8 has ninety weights and twelve biases for a total of one hundred and two design variables. All training methods require hundreds of iterations and more typically tens of thousands. The size of the network must be considered, a larger network with more neurons and more hidden layers can fit increasingly complex functions. This comes at the expense of requiring more design variables

and therefore a longer training time [18]. It is possible to train a network on a dataset for too long, which is known as over fitting [19]. In that case the network continues to improve its fit to the training data set but begins degrading its fit to the underlying function. Some training methods hold back a portion of the dataset for validation testing to protect against over fitting.

Backpropagation

Backpropagation training takes advantage of the known network architecture to directly calculate the required change in weight and bias values to move the network towards the desired output values. This requires that network is feed forward, and that the neuron transfer function is differentiable. In this work the Levenberg-Marquardt back propagation method included in MATLAB's neural network toolbox was used [20][21]. The tansig transfer function is used, the training datasets input values must be scaled to ± 1. This puts the starting point of the network on the steep gradient of the tansig curve allowing for rapid gradient descent.

Common back propagation methods can only train directly on the network's output. This prevents training on the complete time integration scheme. The required output value must be calculated beforehand, and the network trained to give those values. This method has the advantage that all data points can be evaluated in parallel. This however results in a fragile model when included within the time integration scheme. Small errors build up over time moving the state away from the range of the training data, so the output then quickly diverges.

4.3.1 Problem agnostic optimisation methods

Generic optimisation methods can also be used with neural networks, this does not require any knowledge of the network architecture. This allows nondifferentiable transfer functions and more complex architecture. This also allows complex cost functions that do not need to train directly on the network output. With generic training the ReLU transfer function is preferred, this has two advantages. Firstly, there is no constraint on the magnitude of either input or output value, this removes the need to pre-scale the training data set. Secondly the function is quicker to evaluate, with tens of neurons, thousands of data points, and tens of thousands of iterations the transfer function is evaluated millions of times. Saving a couple of floating-point operations and the exponential required for the tansig function can save hours from the total training time. When evaluating the network, the array of design variables from the optimisation must be converted into a network structure of weights and biases.

In Situ cost function

Generic optimisation methods allow an in situ cost function to be used. This evaluates both the networks performance and that of the time integration scheme. This removes any build-up of errors over time and ensures a stable model. This also allows the network to cancel out any inaccuracy caused by the integration scheme. The disadvantage of this method is the extra time required to calculate the cost value; each training data set must be run in series from start to end. Figure 4.9 gives an example of cost function, the area between the target output and current output is cost value. This cost function is the same as the final use case of the network within the time integration scheme.

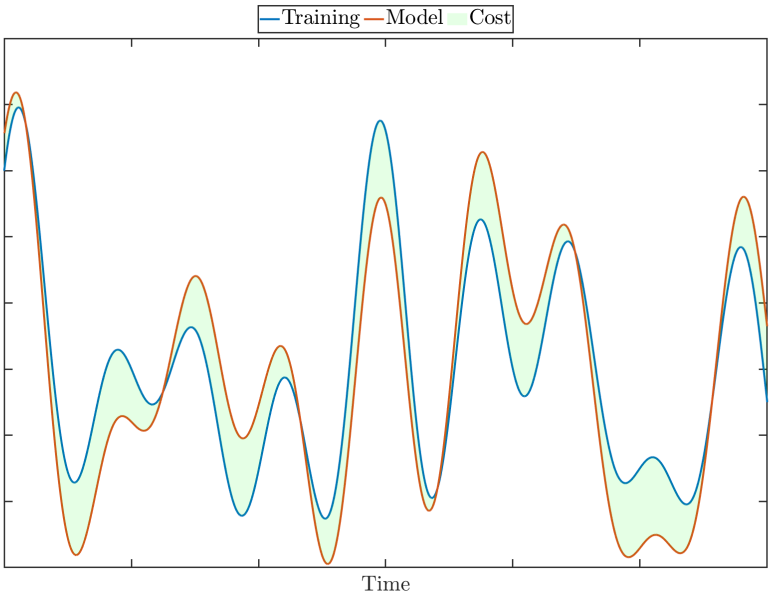


Figure 4.9: Cost calculated from the integral of the difference between the data and simulation

If there are several known state values, the cost of each is summed. In the case of a valve this might be ball height and flow rate. A weighting must be applied to the cost of each output variable to normalise them relative to each other, this stops the optimisation favouring one output over another because it happens to have a larger value due to the units used. An alternate approach, that was not considered in this work, is a multi-objective optimisation method [22].

Genetic Algorithm

Expanding on the natural origins of neural networks genetic algorithms were initially used for network training. Genetic algorithms are inspired by natural selection, a population of possible networks are all evaluated [23]. The best of each generation is taken forward and the worst removed. Initially the MATLAB GA tool was used, for greater flexibility and speed a custom algorithm was developed

[24]. The initial population is randomly generated. Genetic algorithms utilise several ways to manipulate the population to explore the design space. The population is first sorted from best to worst. The best is always kept unchanged. A set percentage of the population is discarded using a randomised selection process preferring to discard lower performing individuals. Children are then generated from the remaining population to restore the original population size. These children are made up of a random number of values from each of two random parents. The whole population is then mutated slightly, except the best individual. Figure 4.10 shows an example convergence, the cost is initially very rapidly improved, the improvement then levels off towards a local minimum. A mutation develops that moves out of the local minimum and it eventually converges to a second minima. There is a small improvement in almost every iteration however the improvement is so gradual that it is impractical to run the algorithm long enough that it gets stuck completely.

One of the limitations of this method is that many different combinations of network values can produce the same result. For example, a network with two neurons in a single layer will give the same output if the weights and biases of each neuron is swapped. A child of two such good networks made up of one neuron from each parent results in a much worse fit.

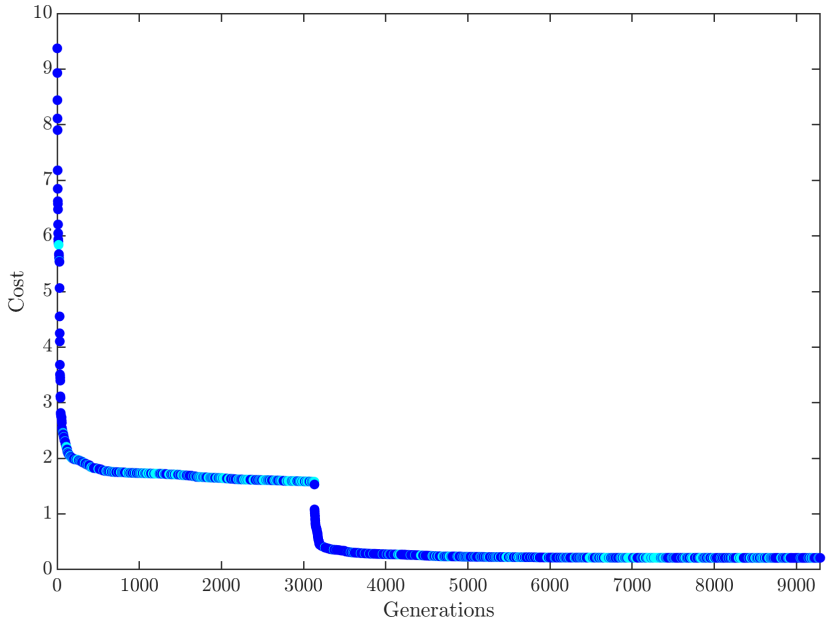


Figure 4.10: Convergence of genetic algorithm, improved generations in dark blue, no improvement in light blue

One method tested to improve the speed of training is to initially train on a smaller network with fewer neurons. As the optimisation begins to reach a minimum a new neuron is added allowing it to move past that limit. Although this does not improve the fit for a given network size, it was hoped that the initial fit would be sped up because it is done on a smaller network. This is a type of Topology and Weight Evolving Artificial Neural Network (TWEANN) [25]. Figure 4.11 shows the evolution, initially the populations is dominated by networks with a single neuron. Quickly these die out and are replaced by more complex networks, in this example the maximum network size was limited to fifteen neurons. After one thousand generations this largest network becomes dominant.

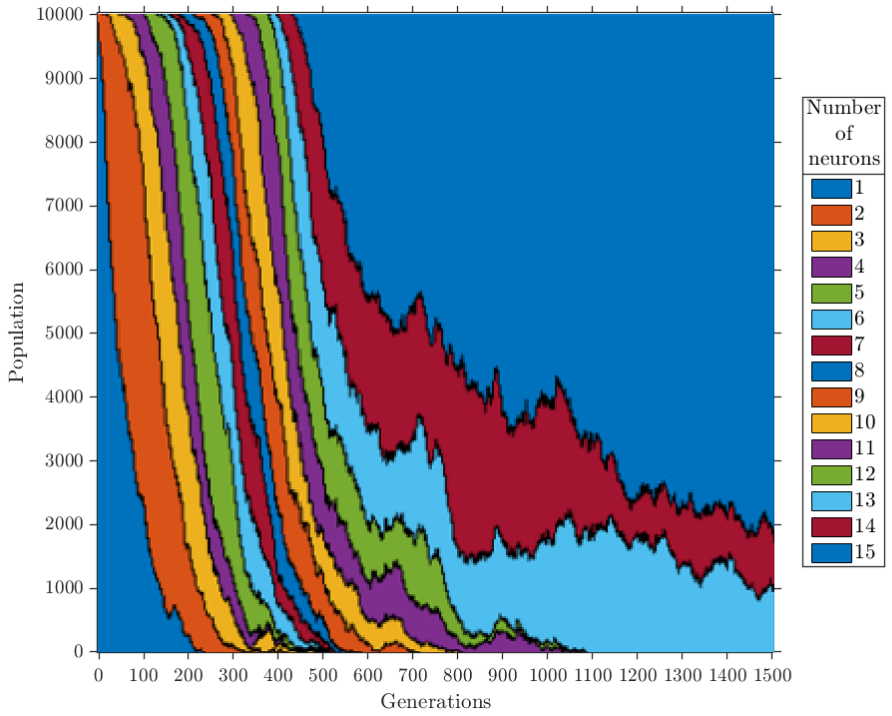


Figure 4.11: Number of neurons in population vs generations for a TWEANN algorithm. Initially the whole population is made up of networks with only a single neuron, as the population evolves neurons are added allowing a more optimal solution to be found. As the optimisation progresses larger networks begin dominate. In this case the maximum network size was limited to fifteen neurons.

More complex optimisation methodologies were also tested, one of which was NeuroEvolution of Augmenting Topologies (NEAT) [26]. This is a type of TWEANN, giving the genetic algorithm more control over the network architecture. Rather than just adding neurons NEAT allows links to be added and removed, including loop backs such that the network may no-longer be feed forward only. It was found that more complex algorithms would increase the per generation cost value improvement, but at the expense of longer execution times per generation. Due to this more advanced methods resulted in little or no improvement for a fixed training period. This effect is very problem dependant,

in this case the genetic algorithm update took a significant proportion of the evaluation time. Problems with larger datasets where cost evaluation dominates execution time will greatly benefit from more advanced optimisation schemes.

Particle Swarm Optimisation

In the hope of speeding up the training process Particle Swarm Optimisation was investigated [27]. This proved to train faster than the genetic algorithm providing equivalent results. Particle Swarm Optimisation overcomes the short comings of genetic algorithms approach to combining two parents that does not apply well to a neural network use case. Genetic algorithms population becomes a swarm in particle swarm terminology, both refer to the pool of candidate solutions. Each particle within the swarm is initialised at a random position with a random velocity vector, each an array whose size is the number of dimensions of the problem. As each iteration is run the best result of each individual particle is recorded along with the overall best of the whole swarm. The velocity of each particle is then updated to be the last velocity plus some velocity towards the particles best and some velocity towards the swarm's global best. Each of these velocities has a scale factor. The factor applied to the particles last velocity is called inertia, the personal and global best also each have a coefficient. These scale factors are set to balance exploration where the particle has a high inertia and tends to keep moving in one direction and exploration where the particle is drawn toward the global best and exploits that local area. Typically, an optimisation will be run with a high initial value of inertia and reduce it over the course of the optimisation. This allows the swarm to cover a wide area initially and then concentrate all particles around the global best as the optimisation begins to converge, this is known as inertia damping. Algorithm 2 gives pseudocode for the PSO algorithm used in this work, the full implementation can be found in

the appendix.

Algorithm 2 Particle Swarm optimisation pseudocode - part 1

```
// randomly initialise particle position and velocity
position = rand(swarm size, number of dimensions)
velocity = rand (swarm size, number of dimensions)

// variables to store the best position and cost
best position = nan [swarm size, number of dimensions]
best cost = inf [swarm size]
global best position = nan [number of dimensions]
global best cost = inf

for i← 1, max iterations do

    // evaluate the whole swarm
    for j← 1, swarm size do
        cost = cost_function(position[j])

        if cost < best cost[j] then
            // update particle best
            best cost[j] = cost
            best position[j] = position[j]

            if cost < gloabal best cost then
                // update global best
                global best cost = cost
                global best position = position[j]
            end if
        end if

    end for

    if gloabal best cost < target then
       return global best position
    end if
```

Algorithm 2 Particle Swarm optimisation pseudocode - part 2

```
    // calculate velocity contributions
    inertia velocity = velocity ∗ inertia
    best velocity = best position ∗ best coefficient
    global velocity = global best position ∗ global best coefficient

    // update velocity and position
    velocity = inertia velocity + best velocity + global velocity
    position = position + velocity

    // update inertia to move towards exploitation
    inertia = inertia * damping
end for
return global best position)
```

The PSO was found to give good results, at least as good as the genetic algorithm in less training time. Several alternate variations on PSOs were also tested, for example Quantum Delta-potential-well-based Particle Swarm Optimization (QDPSO) [28]. These variations did result in performance equivalent or better to the standard PSO algorithm for a fixed number of iterations. However, as seen with the genetic algorithm, due to the additional complexity those iterations would take longer. This realisation prompted a review of the PSO and cost evaluation function with a view to improving efficiency. Staying within the MATLAB environment this re-work concentrated on removing unnecessary operations and vectorising calculations. Investigation was done into parallelisation; it was found that for relatively small swarm sizes of a few hundred parallelisation provided little additional benefit with more than four workers.

Multiple swarms

Both the testing done with the genetic algorithm and particle swarm showed variable results. The optimisation would have to be run several times to get two

good results, occasionally getting very good results but more often a poor fit. This variation between runs is due to the random nature of the initial starting points. This hit and miss training also makes it difficult to evaluate algorithm changes as several runs are needed to give a fair comparison. This variation along with reaching the limit of parallelisation improvements spurred an investigation into methods running multiple particle swarms in parallel.

As with the PSO itself several more complex methods were tested before settling on a relatively simple scheme. Several swarms are set off, in this case eight each with four workers for parallel cost function evaluation. Each of these swarms keep running until they fail to improve for a set number of iterations. These swarms then return their best results to a management function. If the result is the best seen so far, the swarm is re-started keeping its previous particle positions, but its particle velocities and inertia are re-set. The MATLAB implementation for this can be found in the appendix. This restarts an exploration phase around the previous best. If the swarm is stuck and does not have the best result so far it is restarted completely fresh. Figure 4.12 shows such a parallel swarm optimisation. The blue line ahead of the others on the far right is the global best, because of this it has the most iterations. The ending point of the lines show the variability of a single swarm optimisation, it also shows several swarms do reach very similar results. This parallel approach to the optimisation ensures repeatable results. It was found to be more convenient to pick a run-time limit than specify some acceptable cost threshold for completion.

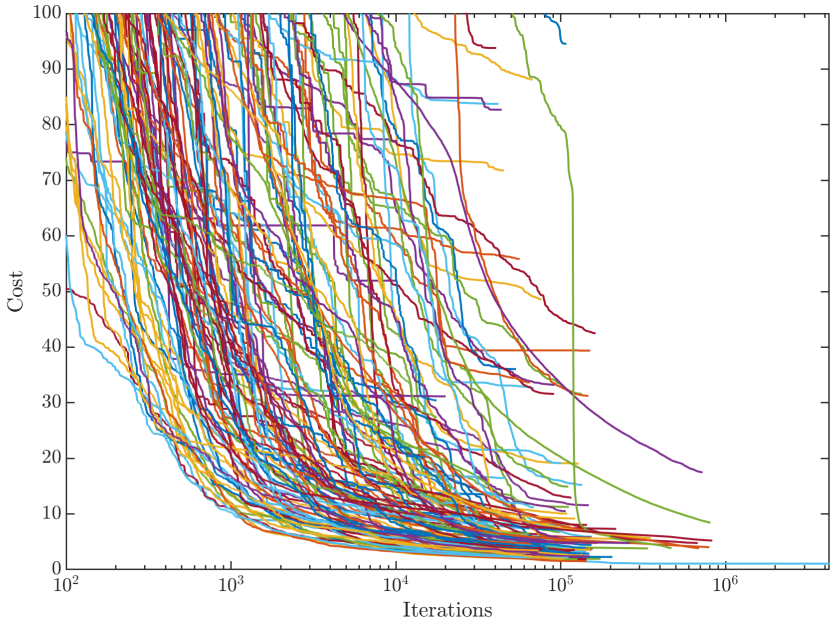


Figure 4.12: Convergence of parallel swarms, cost vs swarm iterations

Chapter 5

Neural network based reduced order model of ball valve

The neural network model was applied directly to the ball valve with spring. This is a more realistic valve including double peak behaviour that could not be picked up by the physics based model using steady state data alone. The neural network approach also struggles to capture the step change in flow rate seen in the disk valve as the disk reaches its software enforced upper limit. This step change is a limitation of the simulation model, the addition of the spring removes it. Valves are used with no spring; however, they still require a mechanical stop that is not reflected in the simplified geometry explored in this work. Such an upper stop would show a similar damping effect to that seen between the ball and seat; this also results in a smoothing of the step change in flow rate that would be easier for a neural network to represent.

5.1 Network architecture

The network architecture is show in Figure 5.1. Pressure is the driving variable. Ball height, flow rate and kinetic energy are the known state variables, two unknown state variables are used. A hidden layer of fifteen neurons was used. This network contains two hundred and forty weights and twenty biases for a total of two hundred and sixty design variables. This network size gave the best results for this problem, a number of different network sizes were also tested, this is explored in section 5.3.2.

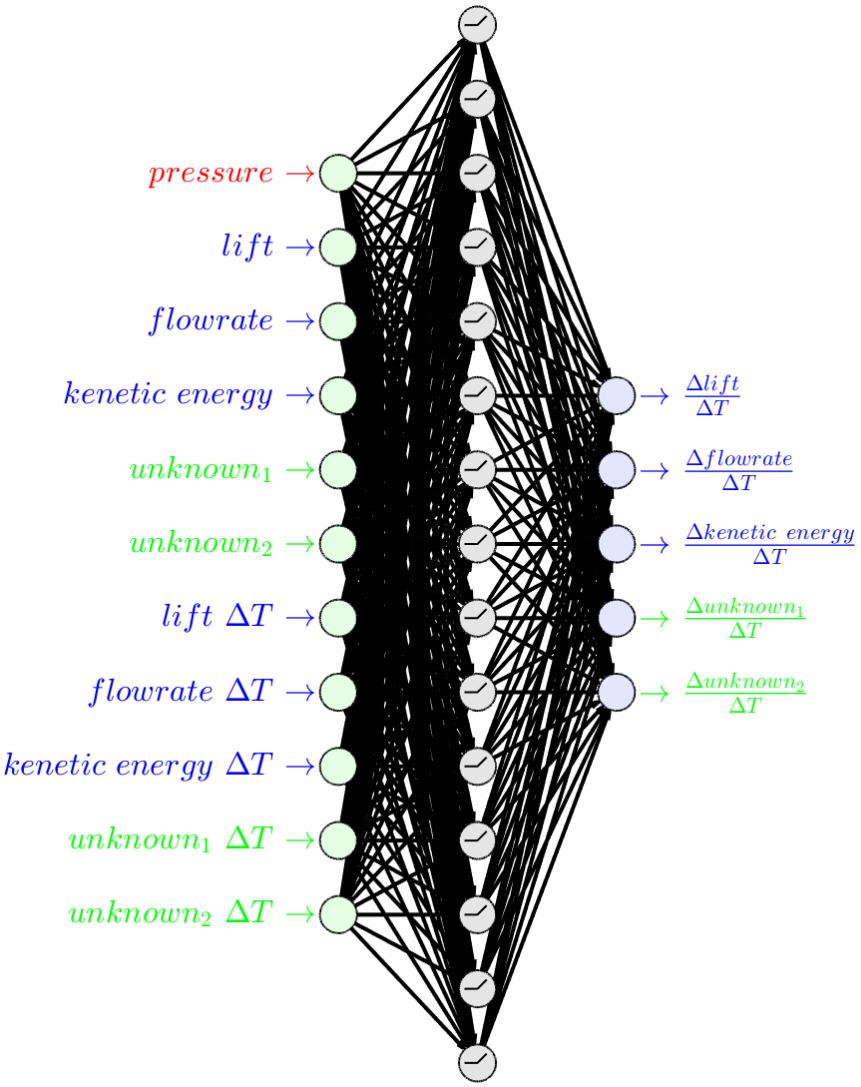


Figure 5.1: Network with a single driving variable, three known states, two unknown states and a single delta time multiple

5.2 Results

5.2.1 Training

Training was done on four simulations each with sinusoidal pressure boundary conditions ranging from 5Hz to 160Hz. This is a reactively small data set by typical machine learning standards, the serial nature of the cost function evaluation for each simulation makes long simulations prohibitively time consuming to train. To ensure all state variables were weighted equally all data was scaled to ± 1. This removes any tenancy to favour one state over another due to the units. All data was scaled up-front, this removes the need to scale as part of the cost function thus speeding up training. Again, to speed-up training the simulations were limited to two and a half periods, this is enough to capture two opening and closing cycles of the valve. The number of time steps was also reduced. As this network evaluation loop is run millions of times over the course of a training run significant effort was put in to speeding it up. This was done using MATLAB coder; the evaluation function was generated as MEX file resulting in an approximate ten times speed-up. Pre-existing platforms such as PyTorch and TensorFlow were used as they provided limited frameworks for the In-Situ training schemes used in this work [29][30]. The cost value was summed for each training simulation with a scale factor applied to weight each equally independent of length. This singular cost value was then used with the PSO scheme outlined in the previous chapter. Figure 5.2 shows the convergence of the training run, in this case a 32 hour run using 28 workers. The convergence plot shows that many of the swarms reached low cost values, giving confidence that this result is not a lucky one off.

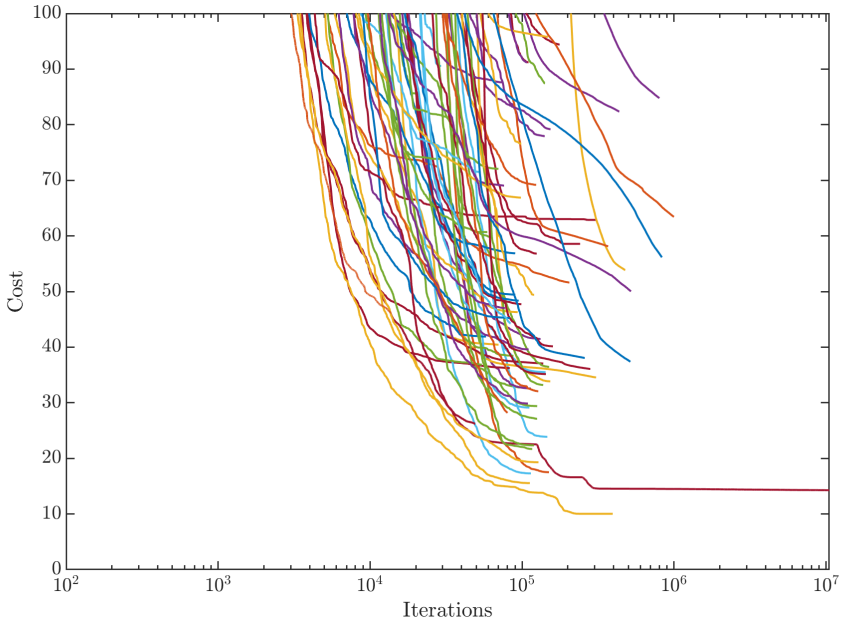


Figure 5.2: PSO training convergence

Fifteen neurons in a single hidden layer were used giving a total of two hundred and sixty values to be found. Over the course of the optimisation three hundred and fifty separate swarms were started each with one hundred particles. The resulting fit is shown in Figures 5.3 to 5.6 training in green and network in red, the error value of each of the variables is given in Y axis label. The network fits the training data well, only missing some small oscillations in the open position.

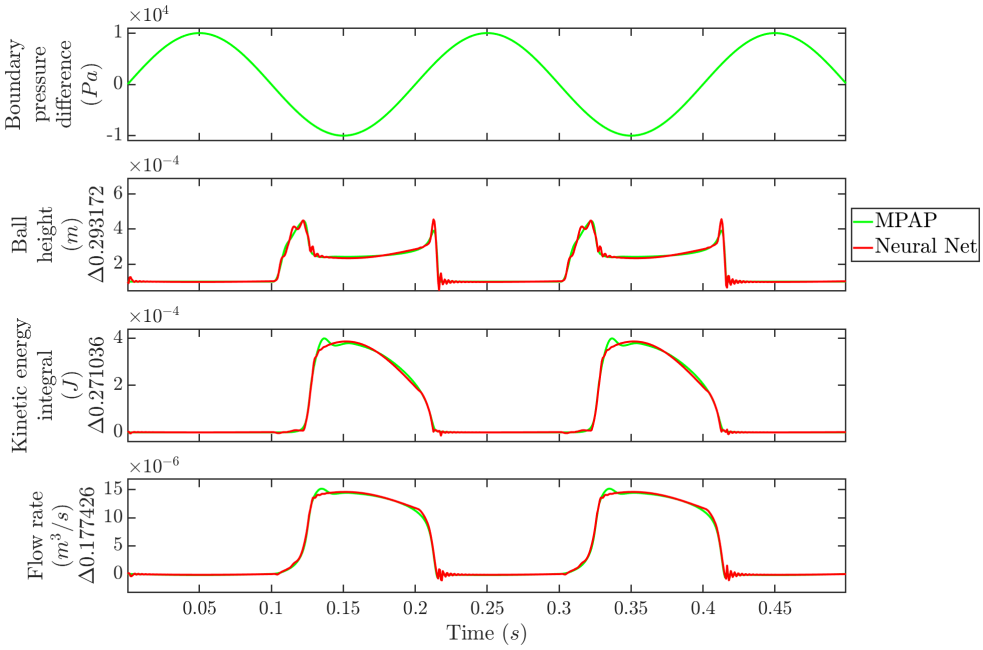


Figure 5.3: Training simulation - $5Hz$

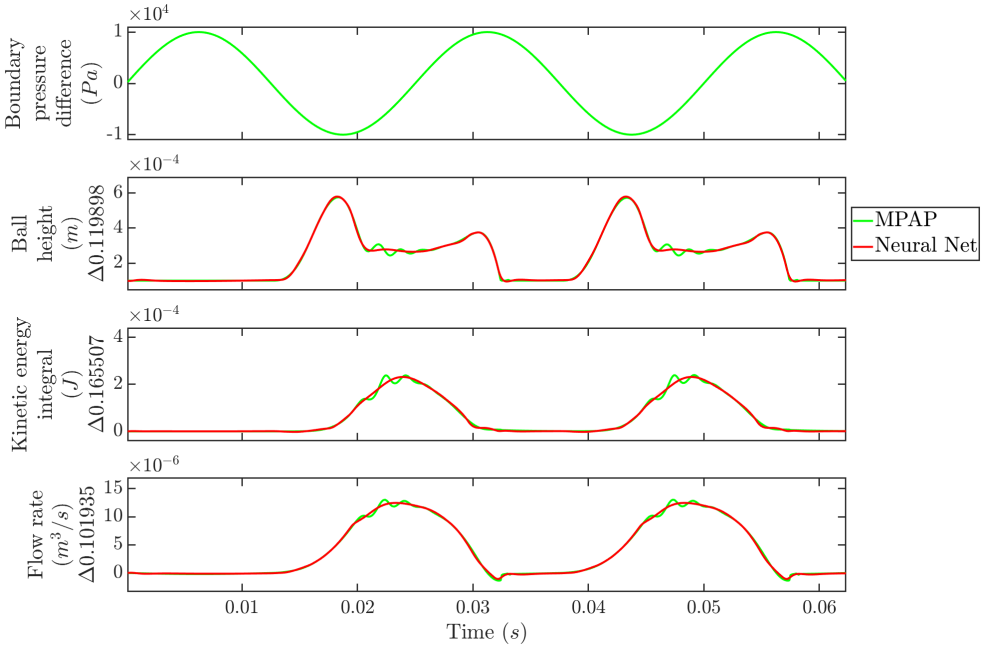


Figure 5.4: Training simulation - $40Hz$

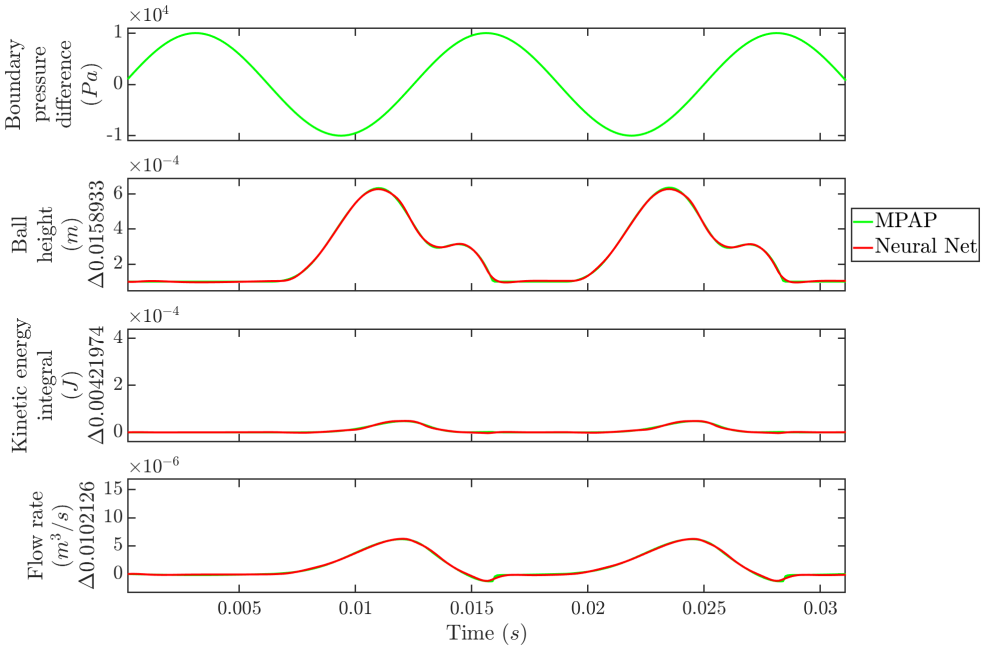


Figure 5.5: Training simulation - $80Hz$

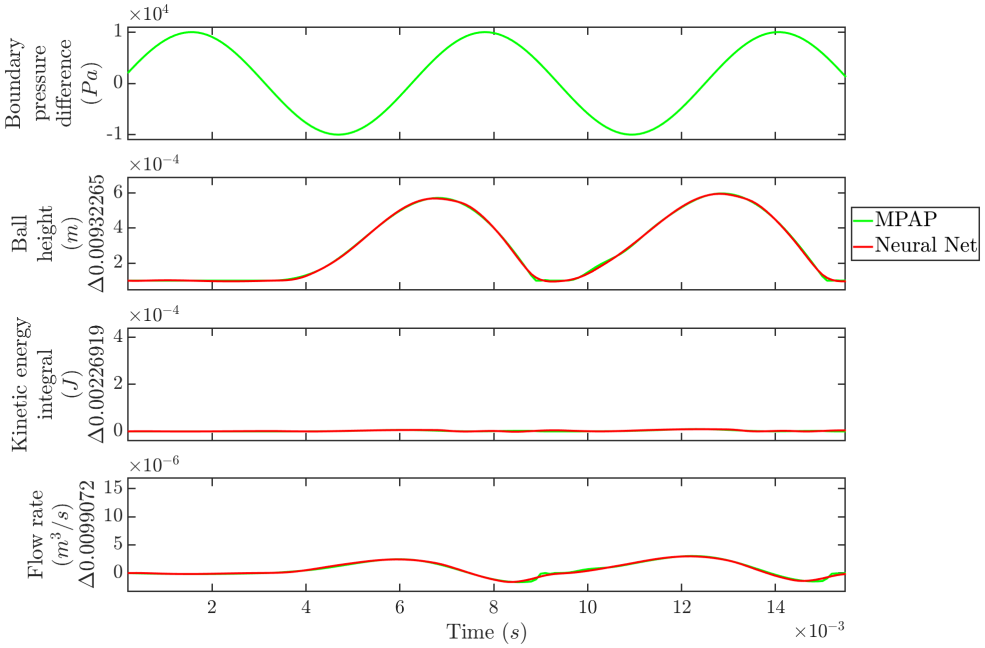


Figure 5.6: Training simulation - $160Hz$

5.2.2 Validation

Validation is a step taken to provide a unseen data set to the model this will reveal over fitting. A model may train to a very low error by learning the training data set rather than the underlying process, that would result in a poor validation cost.

To validate the trained model, it was compared against the full-length training simulations at their original time step size along with four unseen simulations. Rather than applying the validation throughout the training process the best particle from each swarm is recorded and the validation check applied as a post process. Figure 5.7 shows this validation cost compared with the training cost for networks over the course of the optimisation. As expected, most points are above the diagonal, their validation cost is worse than training. A couple of outliers are better at validation than would be expected from there training cost, this is just random luck. Some points have a low training cost and high validation cost suggesting those networks may be over fitted. With increased training time points should move diagonally down toward the origin.

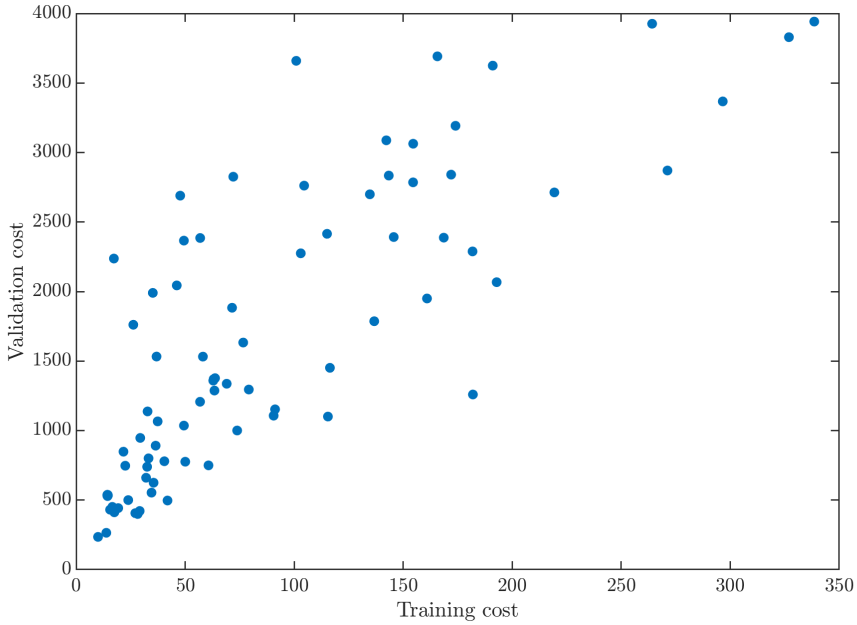


Figure 5.7: Validation vs training cost for the best particle in each swarm. In normal training the validation cost would be expected to decrease in sync with training cost, this results in the diagonal trend with points moving towards the lower left corner with increased training time. If over fitting the training cost continues to reduce but the validation cost increases resulting in points in the upper left corner. Some networks get lucky and have a lower validation cost than is expected for their training cost resulting in points in the lower right corner.

In this case the best validation network is not the same as the best training network, although they are very similar. Figures 5.8 to 5.15 shows the best validation network evaluated on the validation simulations.

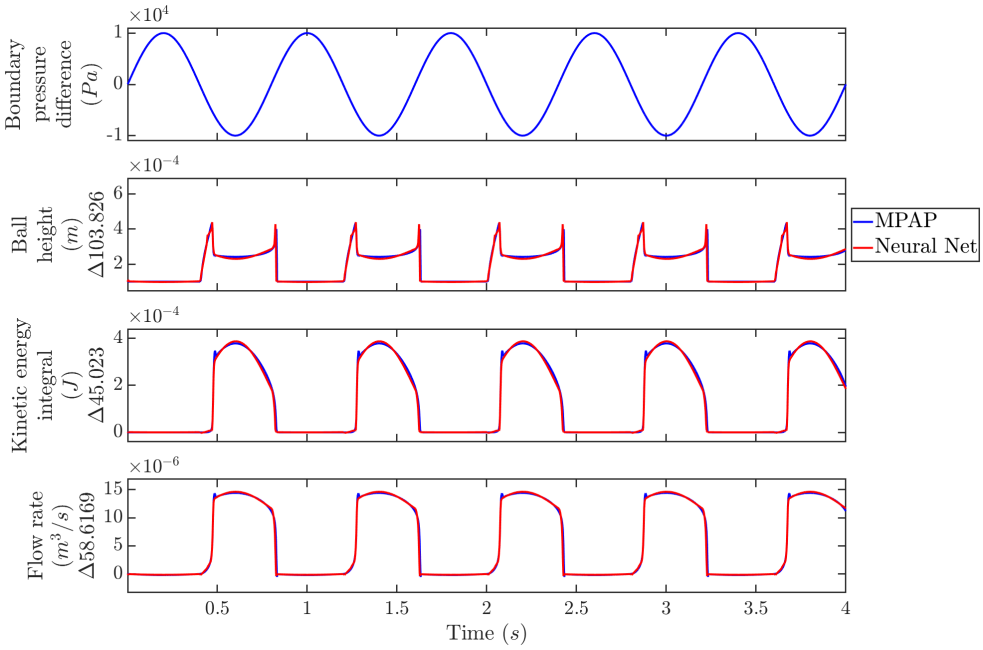


Figure 5.8: Validation simulation - 1.25Hz

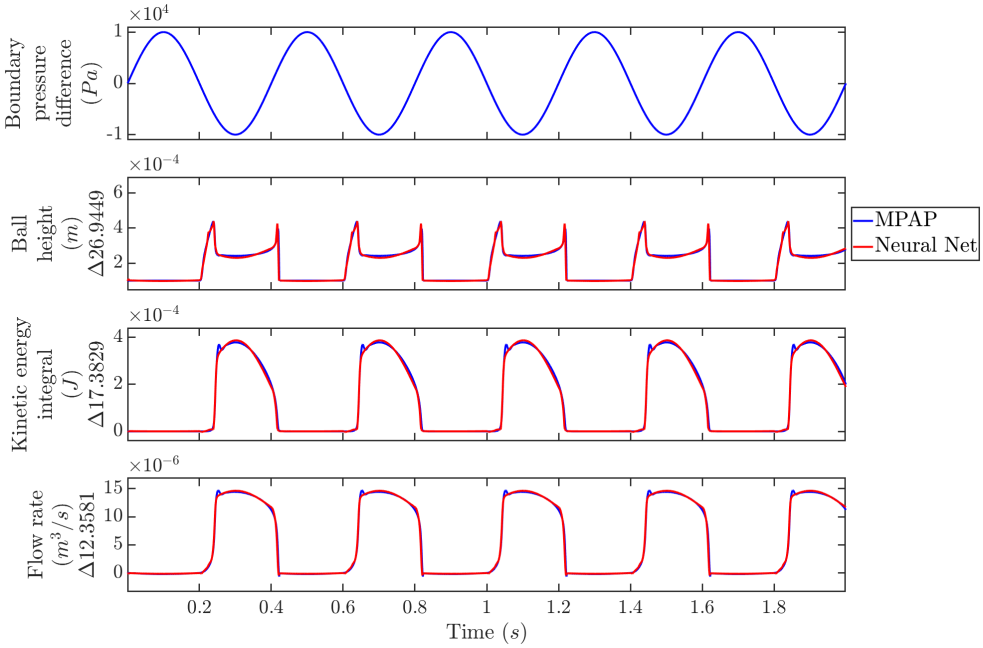


Figure 5.9: Validation simulation - 2.5Hz

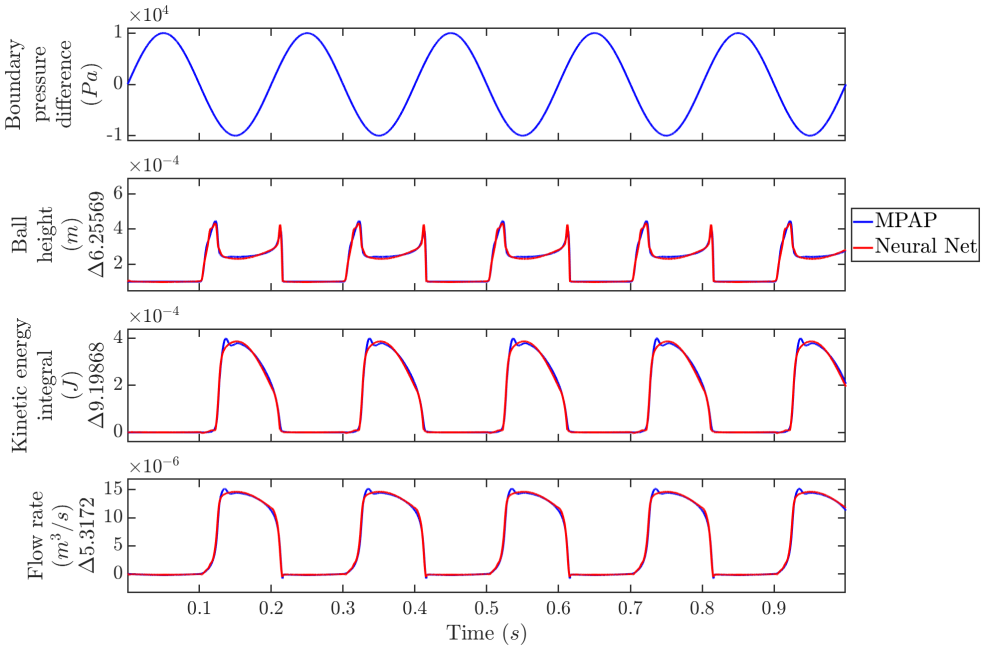


Figure 5.10: Validation full length training - $5Hz$

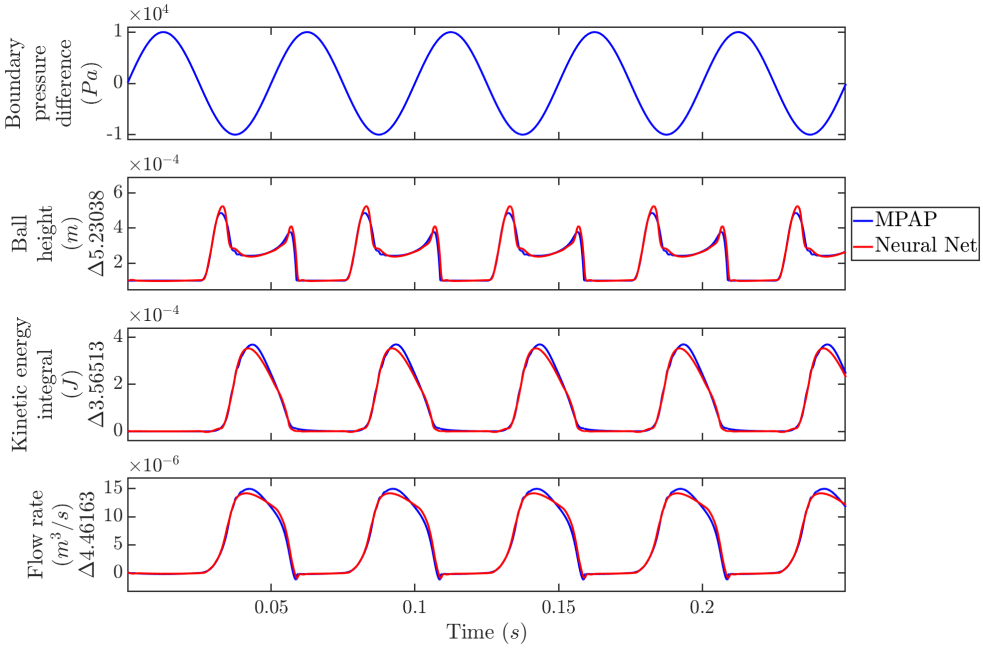


Figure 5.11: Validation simulation - $20Hz$

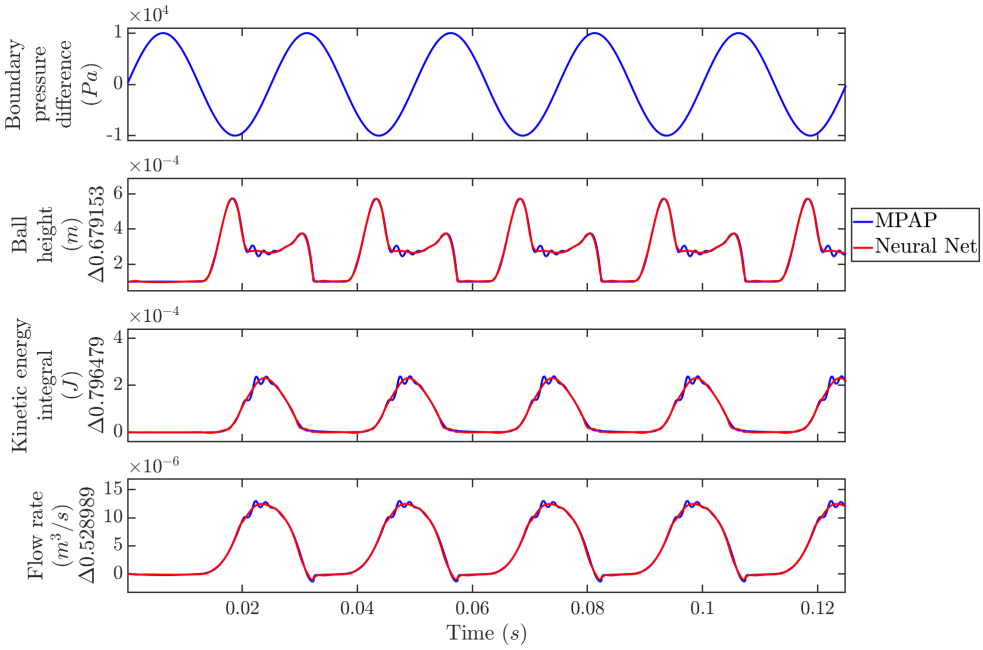


Figure 5.12: Validation full length training - $40Hz$

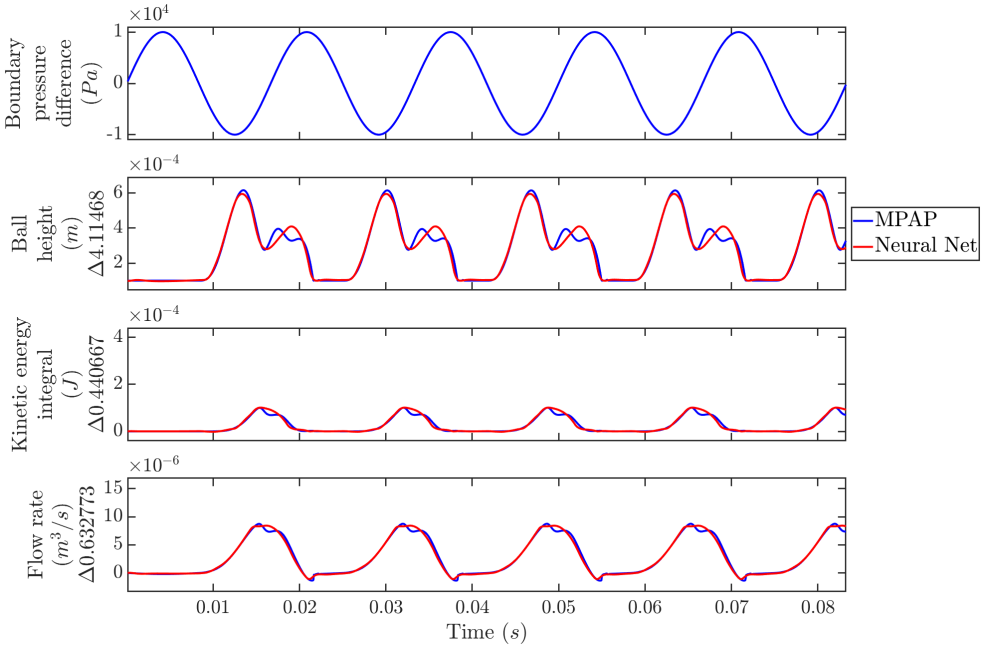


Figure 5.13: Validation simulation - $60Hz$

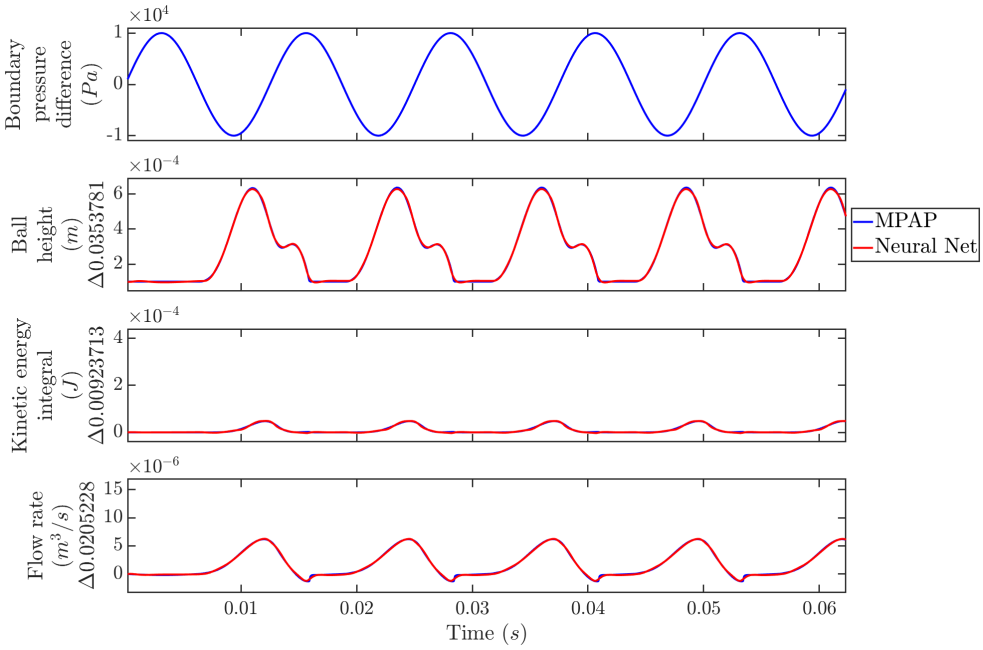


Figure 5.14: Validation full length training - $80Hz$

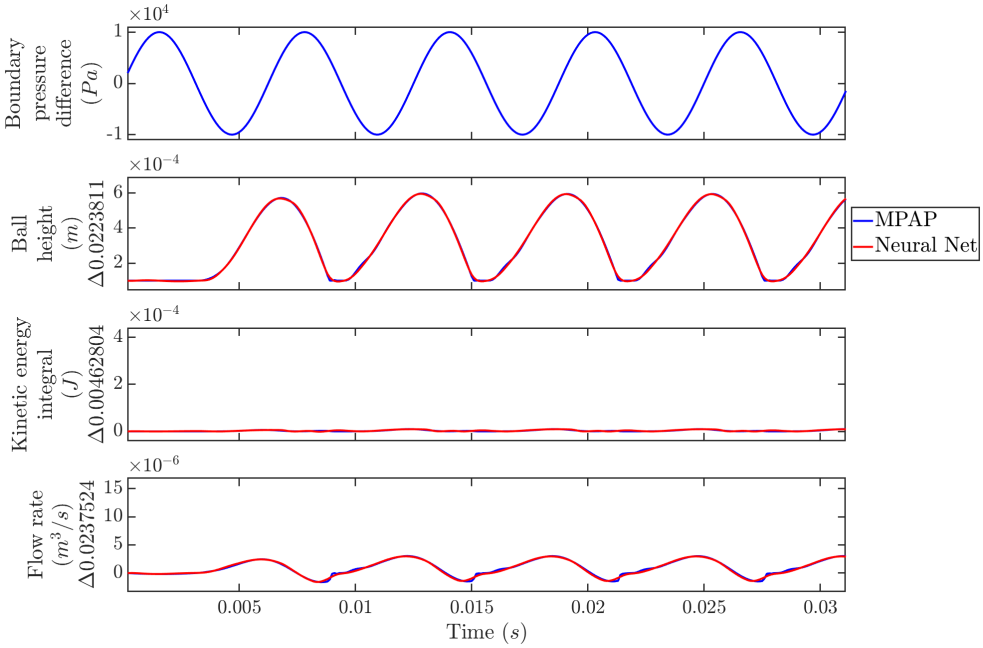


Figure 5.15: Validation full length training - $160Hz$

5.2.3 Testing

Several test simulations were run to evaluate the performance of the validated network on unseen simulations. Firstly, sinusoidal simulations at unseen frequencies. As expected from validation testing the network fits well for the unseen simulation at 107Hz, this is within the frequency range of the training data (Figure 5.16). Evaluating at a higher frequency of 321Hz shows a good fit is no longer maintained (Figure 5.17). This frequency is significantly higher than the highest frequency seen in training, it is unrealistic to expect the network to maintain a good fit to a frequency higher than that seen in the training. As shown by the validation testing the network does work well for frequencies lower than that seen in the training. If the lowest frequency in the training data set approaches steady state it is reasonable to also expect a good fit at any lower frequency.

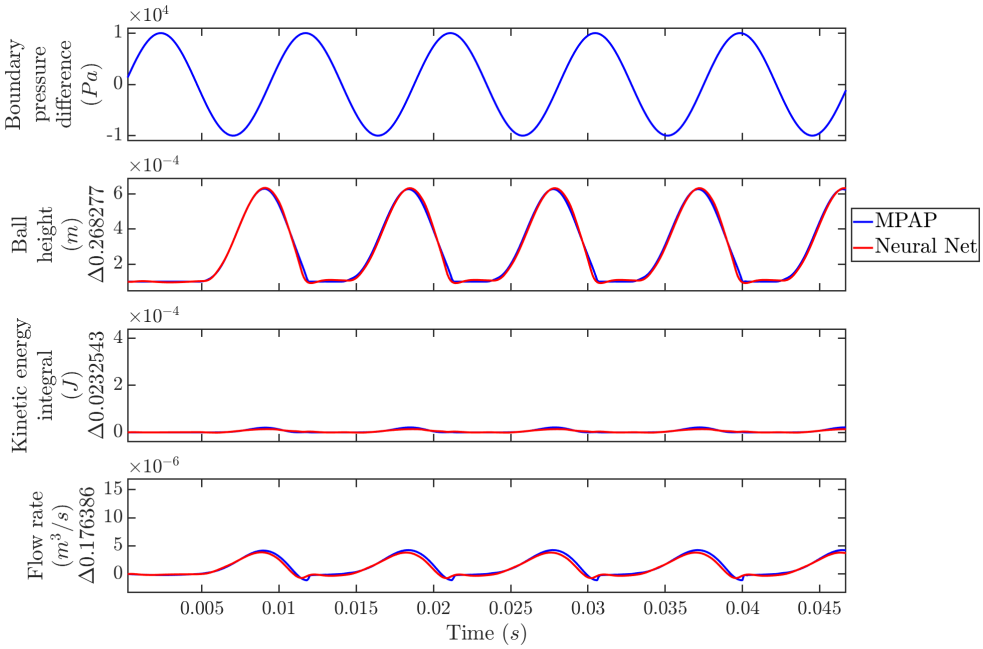


Figure 5.16: Test - 107Hz

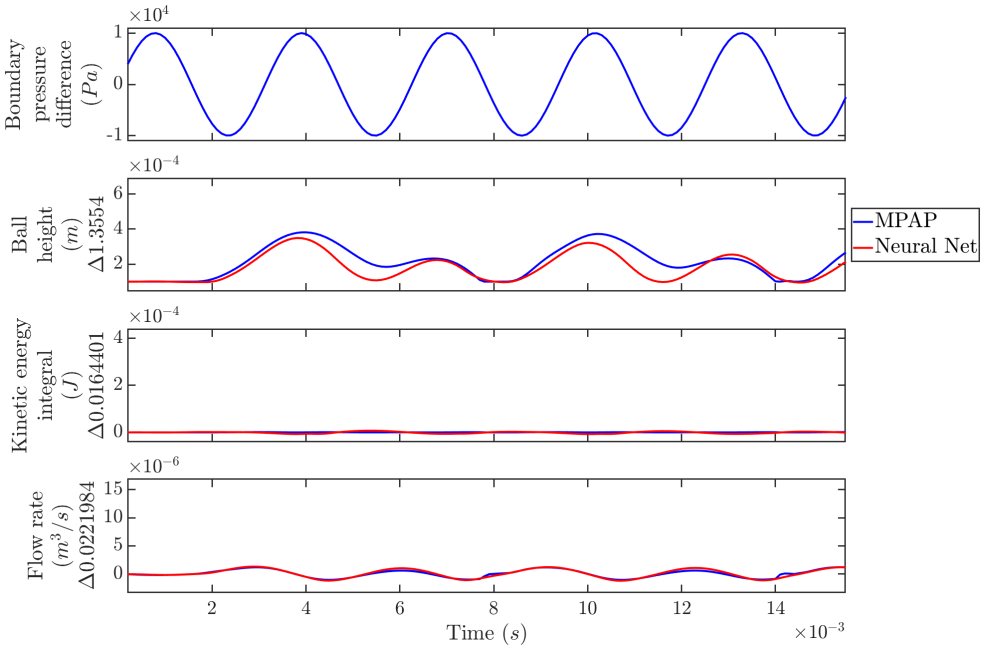


Figure 5.17: Test - $321Hz$

A superimposed sine and cosine simulation shows the network can represent more complex behaviour (Figure 5.18). Both long events holding the valve open and very brief openings are correctly represented.

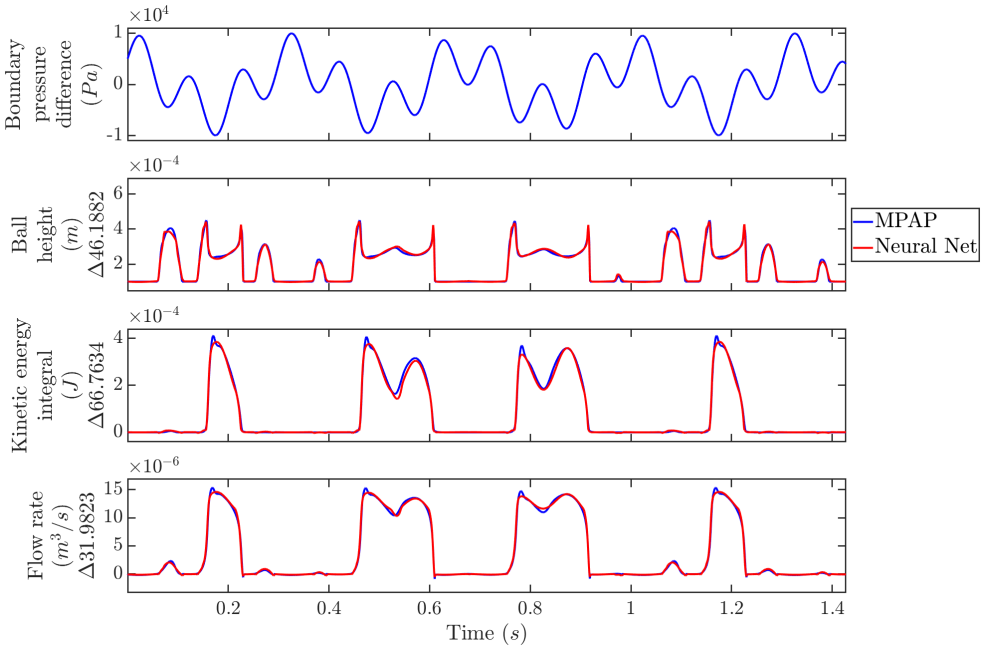


Figure 5.18: Test - superimposed sine and cosine

Five opening simulations were run using a range of constant pressures (Figure 5.19 to 5.23). These show the network responds accurately to constant pressure inputs despite being only trained on sinusoidal data. Lower pressure events are also accurately represented despite a relatively small amount of the training data being at low pressures.

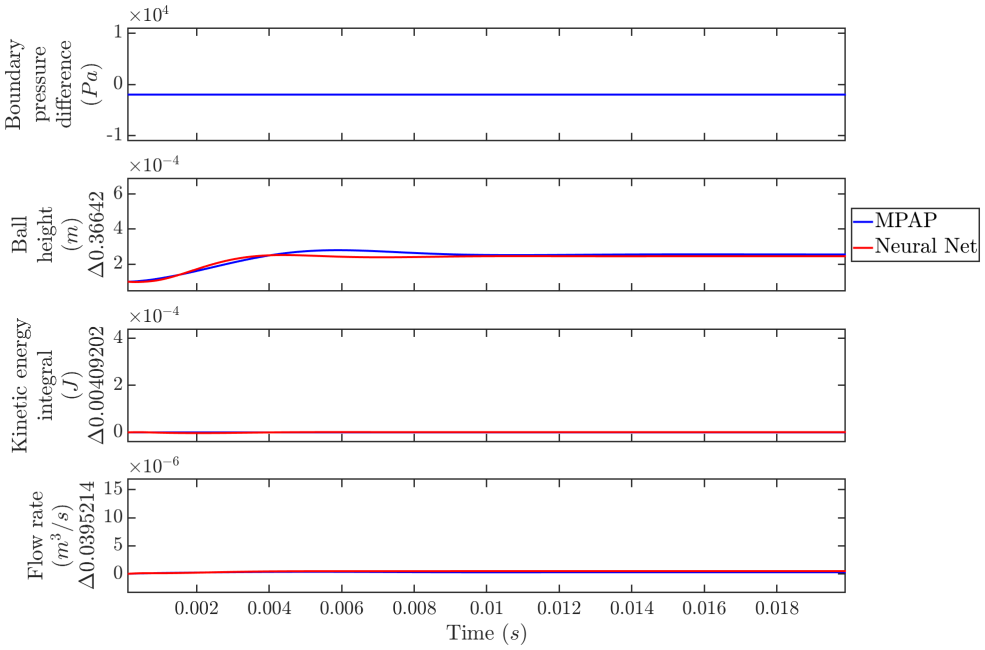


Figure 5.19: Test - opening 20% of max pressure

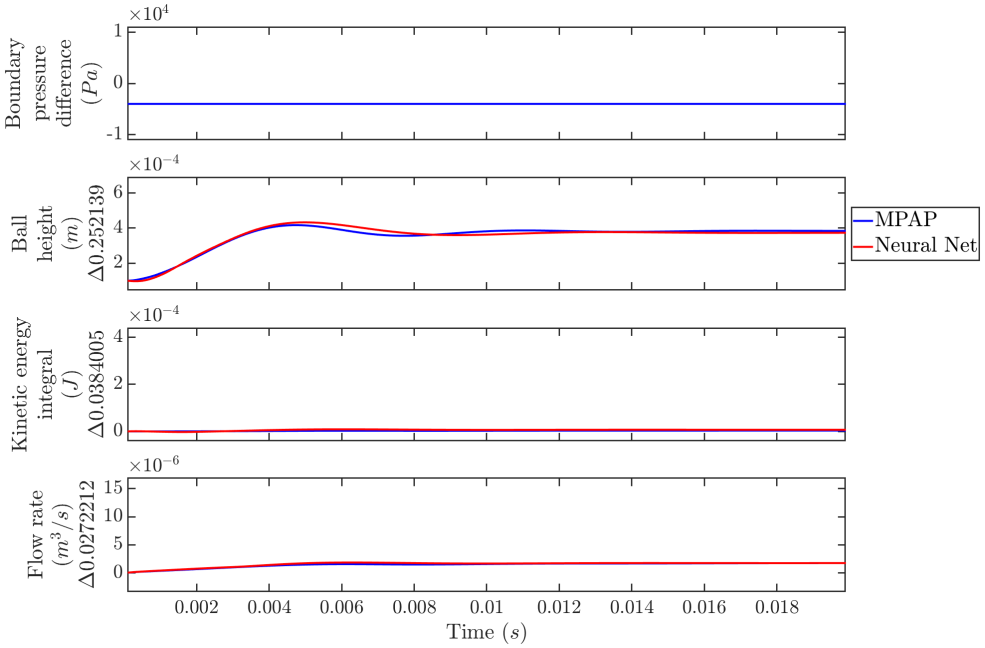


Figure 5.20: Test - opening 40% of max pressure

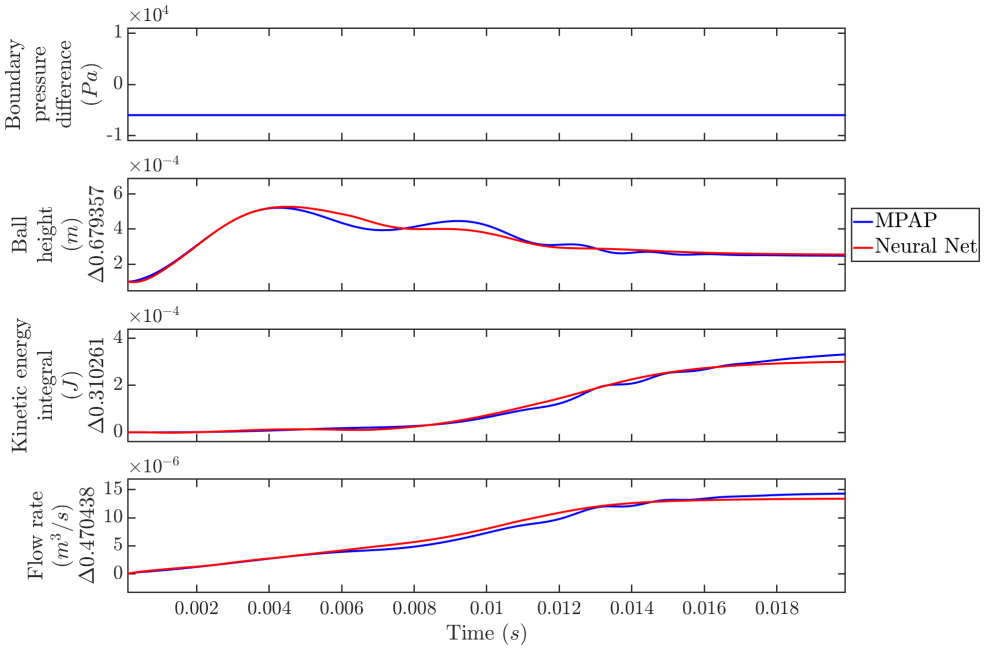


Figure 5.21: Test - opening 60% of max pressure

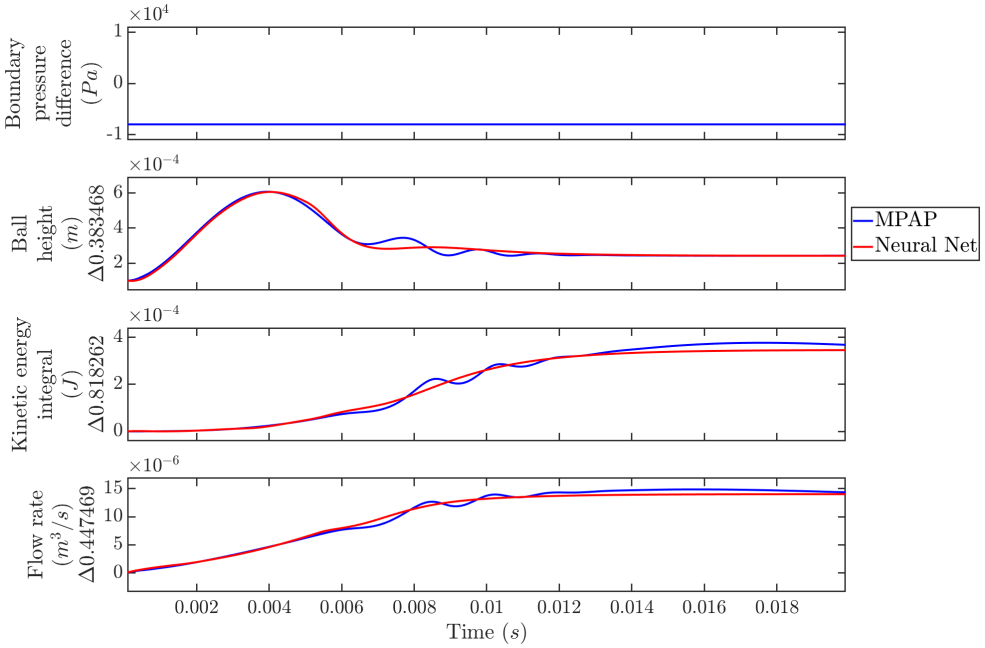


Figure 5.22: Test - opening 80% of max pressure

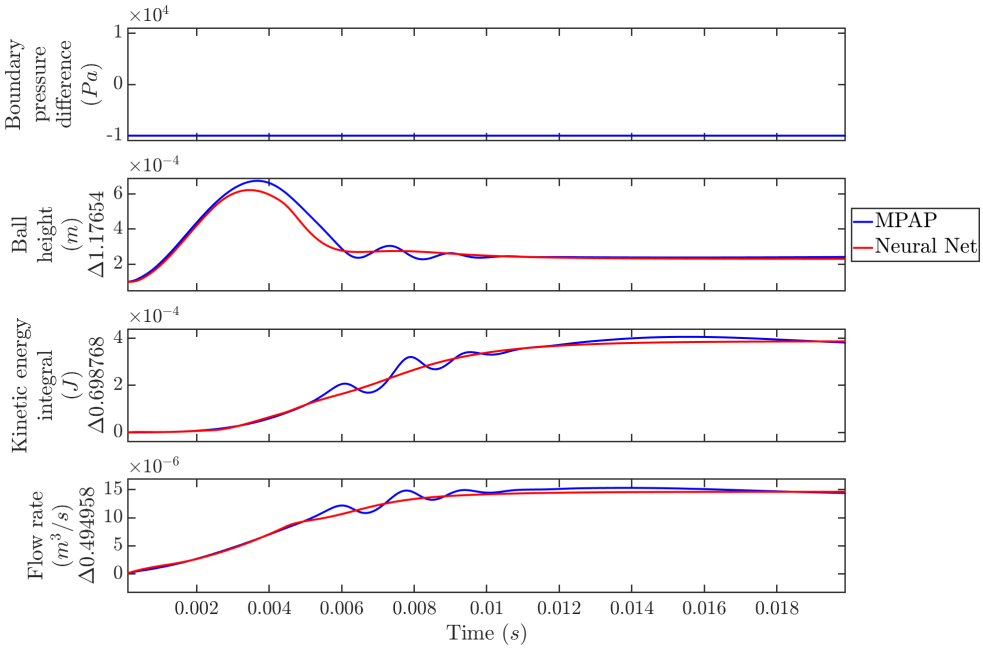


Figure 5.23: Test - opening 100% of max pressure

A limitation of unknown states is the requirement to always use the same initial conditions. This prevents testing valve closing directly with the ball starting in a raised position. Instead, some smoothed impulses were used to first open the valve and close it with a constant pressure, Figures 5.24 to 5.27. Again, these show a good fit proving the versatility of training on sinusoidal simulations.

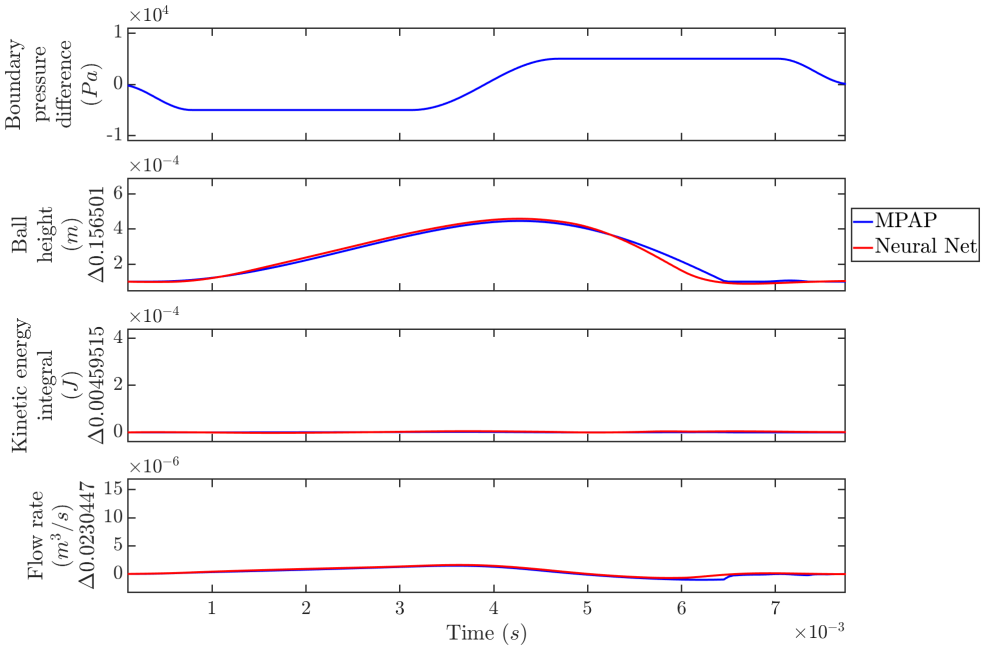


Figure 5.24: Test - fast impulse 50% of max pressure

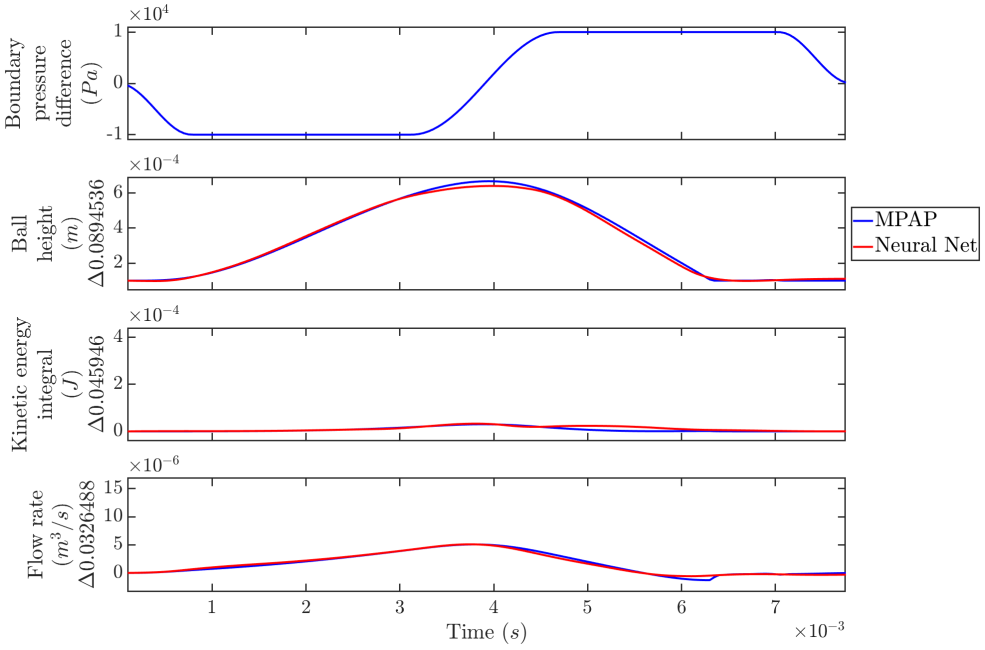


Figure 5.25: Test - fast impulse 100% of max pressure

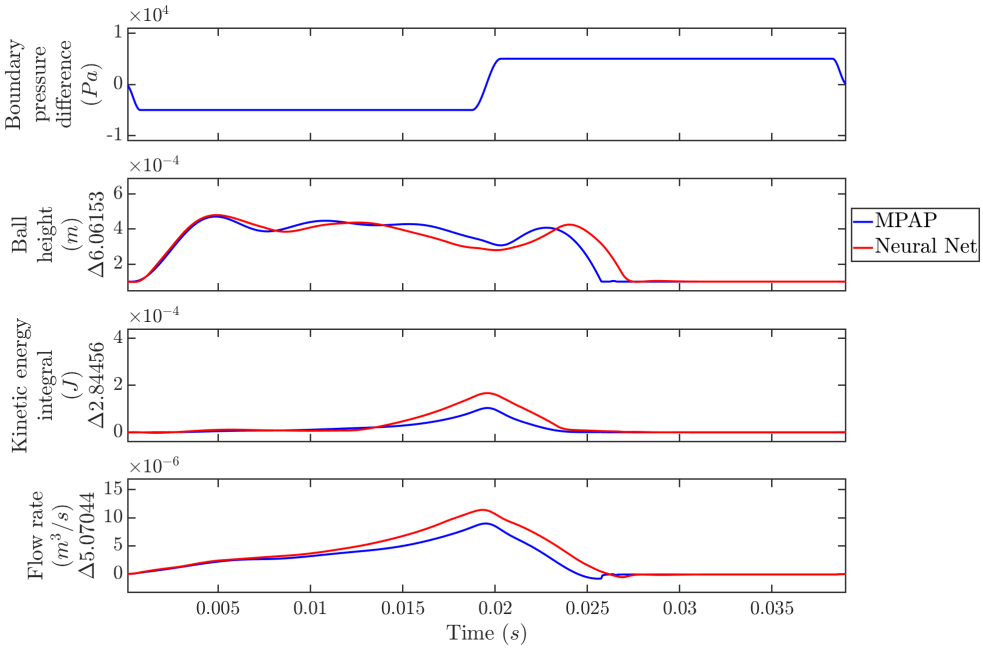


Figure 5.26: Test - slow impulse 50% of max pressure

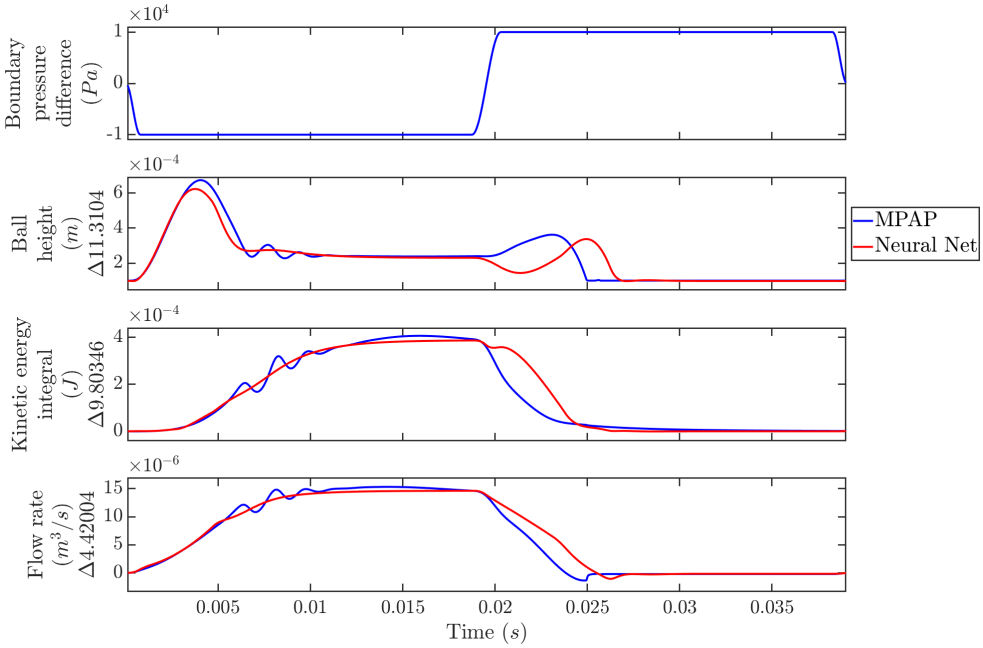


Figure 5.27: Test - slow impulse 100% of max pressure

5.2.4 Time step robustness

To evaluate the network performance at a range of time step sizes the training simulations were evaluated with a reduced time step. Note that all four training simulations already had different time step sizes. The model was not stable for all training simulations at any larger time step. With a smaller time step size the error does increase, it converges to just over thirty percent increase as time step size is reduced (Figure 5.28). It is expected that the error increases as we move away from the training data, this shows the scheme compensates somewhat for the time step size used, if this was not the case the error would reduce with time step. This increase seems quite large however this is relative to the original small cost value. Plotting all evaluations shows a good fit to the training data is still maintained (Figure 5.29).

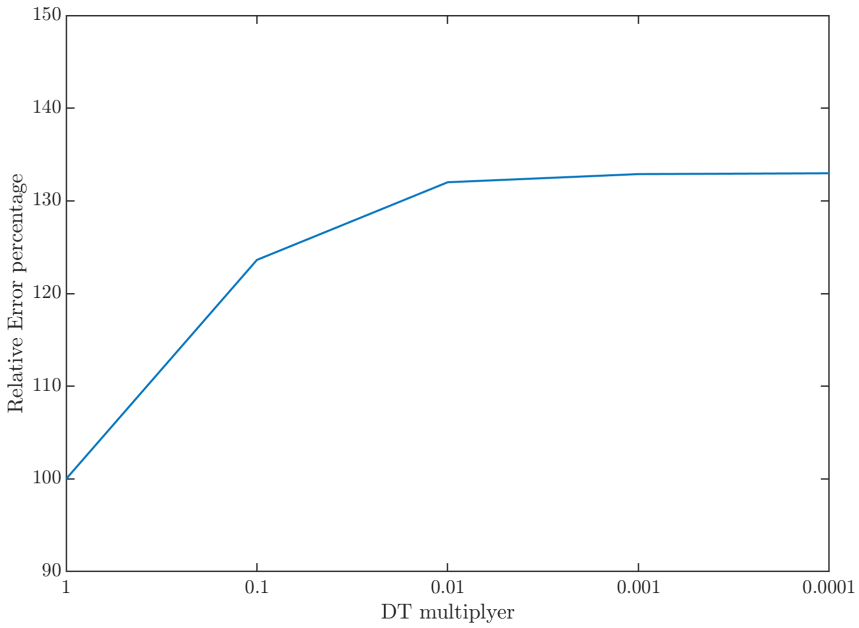


Figure 5.28: Error percentage with decreasing time step size

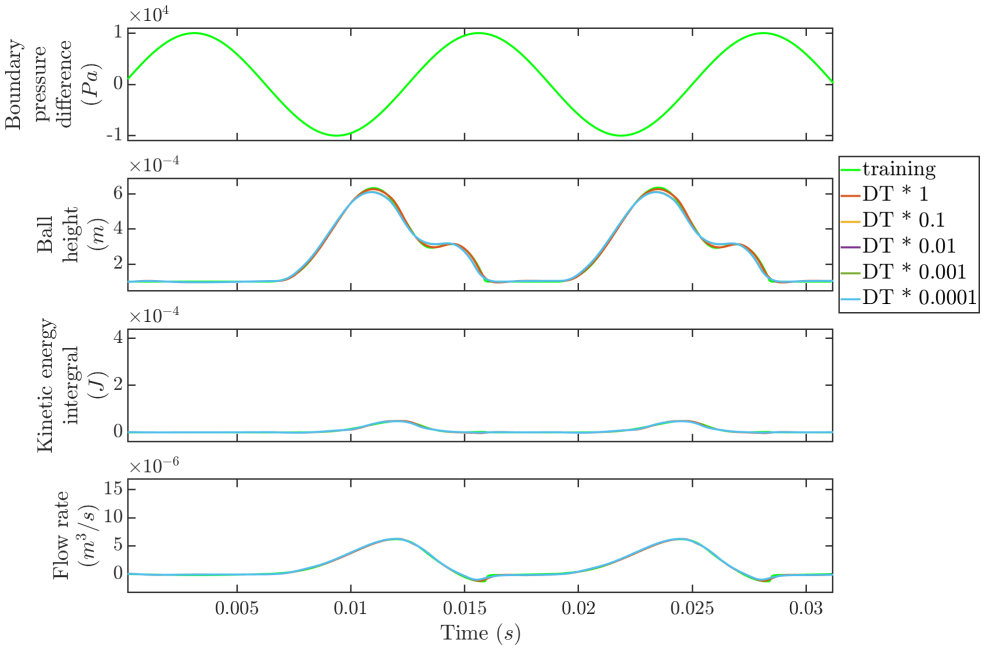


Figure 5.29: Model response with varying time step size, $80Hz$

5.3 Investigation of alternative architecture

5.3.1 Number of unknown states variables

To verify the benefit of unknown state variables several networks were trained with a varying number of unknown states. All networks were trained with the same PSO scheme for a fixed time period. A network with only a single known state, the flow rate, was tested first. With fewer known states additional unknown states should provide more benefit. This benefit is clearly shown, each additional unknown state improves the training result, until reaching four unknown states (Figure 5.30). The network with three known states, ball height, flow rate and kinetic energy integral, shows a similar improvement with one and two states but degradation at three. This degradation starts at a similar number of total states with more than five total known plus unknown states. Each additional state

adds several additional weights to the network making it harder to train. This decrease in effectiveness is most likely due to the use of a fixed training duration.

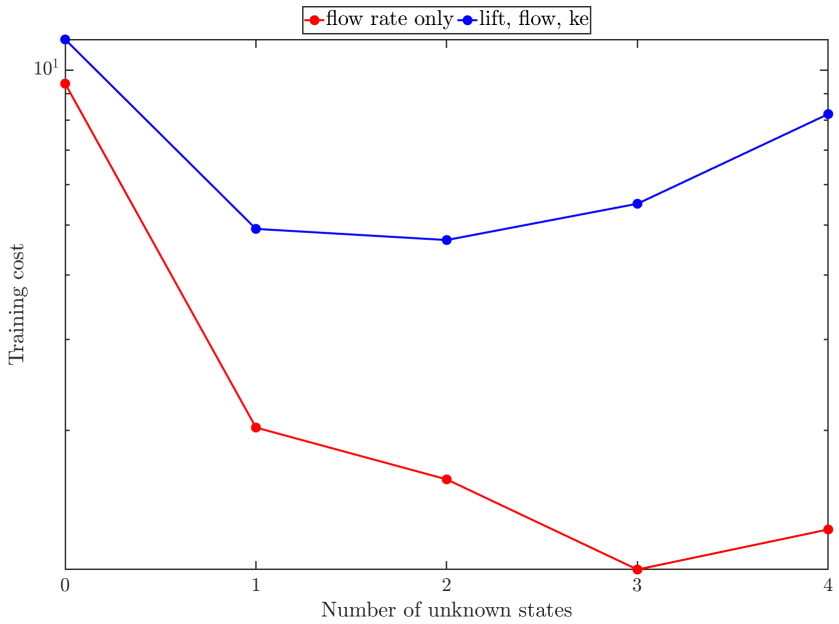


Figure 5.30: Training cost vs number of unknown state variables. Training cost is the sum of the cost for each training simulation.

5.3.2 Network size

This work concentrates on relatively small networks, to explore the effect of network size testing was done with a varying number of neurons. Networks with a single hidden layer of ten, fifteen and twenty neurons were trained using the PSO scheme for a fixed period. Multiple hidden layers were not investigated in depth as part of this work, initial testing gave poor results so the focus was kept on single layer networks. As outlined in section 4.2.2 only a single layer is needed to represent higher order solutions. Again, varying numbers of unknown states were tested. All networks give a very similar result with no unknown states, this suggests the limitation is the input data to the network (Figure 5.31). With a single unknown state both fifteen and twenty neurons show a much larger

improvement than ten neurons, this suggests the ten-neuron network is reaching the limit of what can be represented with so few neurons. Again, all networks show a degradation with higher numbers of hidden states. This is thought to be due to the fixed training time.

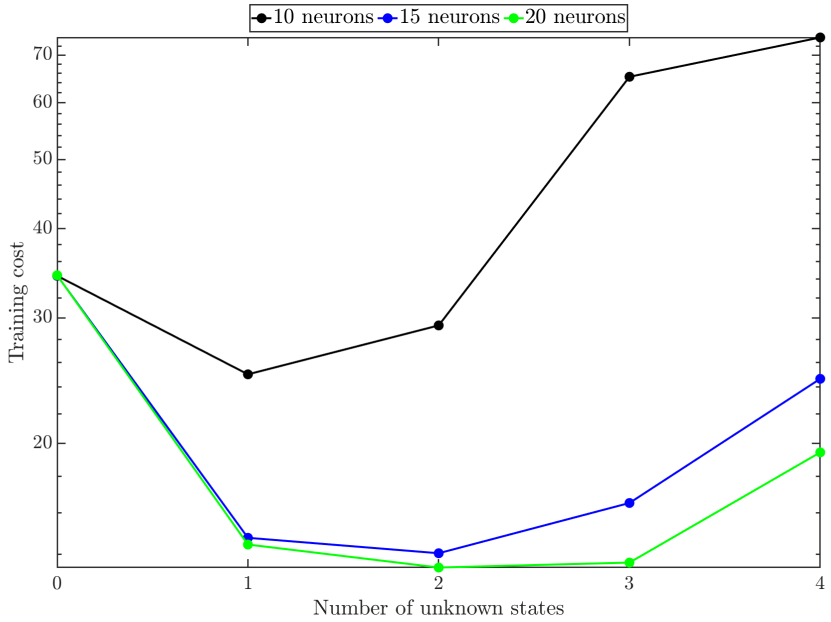


Figure 5.31: Training cost vs number of unknown state variables and network size

Chapter 6

Dynamic system simulation

Amesim is modelling and analysis software available under the Simcenter family of simulation tools from Siemens [31]. Originally developed for use in the oil and gas industry it is now also widely used in aerospace and automotive sectors [32]. Amesim uses a bond graph representation for multi domain systems, readers may be more familiar with Simscape from MathWorks, it provides a similar graphical representation of complex systems [33].

Amesim is used at Schaeffler for modelling engine lubrication systems, of which the valves considered in this work are a small part. Individual components are modelled with a relatively simplistic set of equations; however, the overall system model gives meaningful results. To improve the fidelity of models the methods explored in this work have been implemented as blocks in Amesim allowing for drop-in replacements for the existing blocks.

6.1 Workflow

MATLAB was used extensively in the development of the reduced order methods; it was a simple choice to use the integrated MATLAB and Simulink support

within Amesim. This is by no means the only way in which this functionality could have been implemented. As a first step a Simulink model was made to evaluate a trained neural network and the results of this model were checked against the existing evaluation code. This Simulink model was then exported into Amesim. Amesim 17 was used, but the workflow should be similar newer versions. For compatibility with Amesim 17 MATLAB R2018a was used.

6.2 Simulink

A combination of basic blocks and a custom function were used to evaluate a trained neural network, the same approach could also be applied to the physics based reduced order model. Basic constant, addition and scaling blocks are used to convert to and from the normalised values used by the network. A memory block returns its input value from the previous time step, they are used to provide the time history of the state variables and calculate the time step size. This removes the need for Amesim to provide a time history to the block. Amesim only supports a single input and output node, so the required variables are combined into an array using a mux block (Figure 6.1).

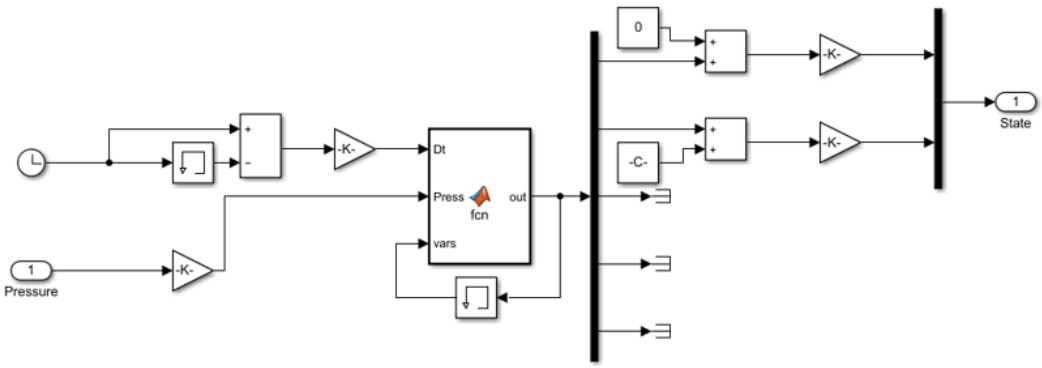


Figure 6.1: Simulink model evaluation

6.3 Software integration

Importing the Simulink block into Amesim is straight forward once the environment is setup correctly. For compatibility between Amesim and Simulink both must use the same compiler, to achieve this Microsoft Visual C++ 2013 was installed and both Amesim and Simulink configured to use it. MATLAB is then launched from the Amesim menu, this loads the Amesim libraries. The `sl2amecosim` command is then used. The exported block then must be added to the Amesim block libraries path. The block can then be used in any model.

6.4 Comparison with built in models

A comparison to a basic valve block makes a simple example of the complete workflow. A training data set was generated using a Amesim ball valve block with a sinusoidal pressure change at a range of frequencies (Figure 6.2).

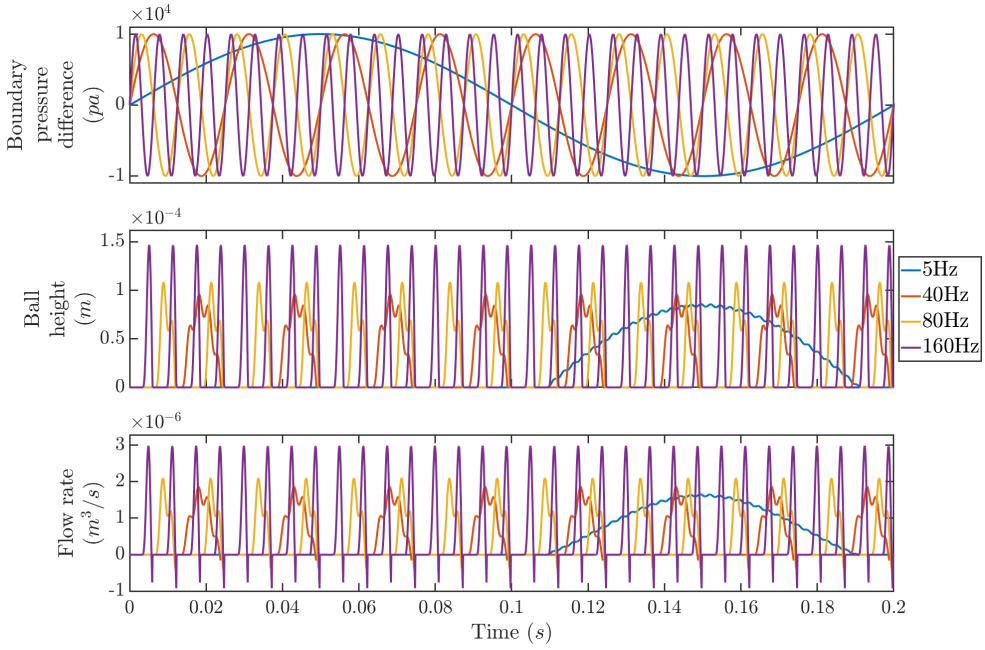


Figure 6.2: Training data set generated from Amesim

A neural network was trained on this data set using a single hidden layer of twenty-five neurons and three unknown state variables (Figure 6.3). Training completed quicker than that performed on data sets generated from CFD simulations. It is thought this is because the data is generated from a fixed set of equations within Amesim, while more complex flow behaviours seen in CFD are not captured by these equations.

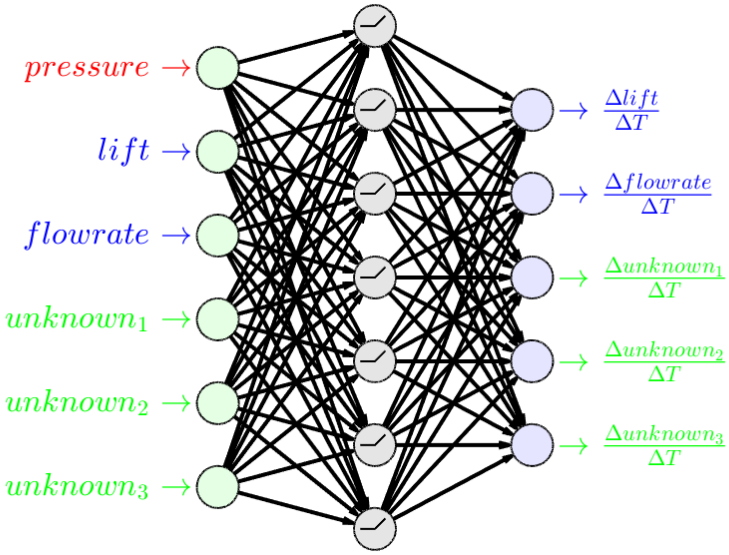


Figure 6.3: Network input and output variables, note: reduced number of neurons shown in hidden layer, true network used twenty-five neurons

This network was then included in the Simulink block and in turn imported into Amesim. This allows both the original valve block and the trained model to be compared within the same simulation (Figure 6.4).

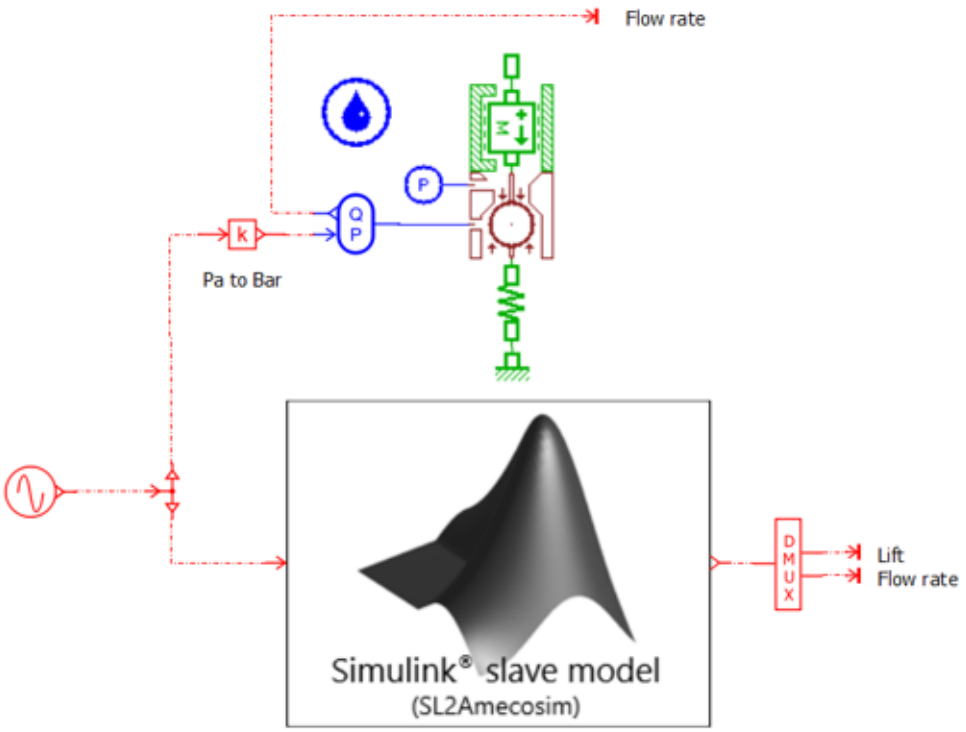


Figure 6.4: Original Amesim block and imported custom model

An evaluation of the error at different time step sizes was done to test the robustness of the scheme. Unlike the previous example the same time step size was used for all training data sets. The trained network shows very poor results for larger time steps than those used in the training data. As might be expected the lowest error is seen when the network is evaluated at the same time step size as it is trained at (Figure 6.5).

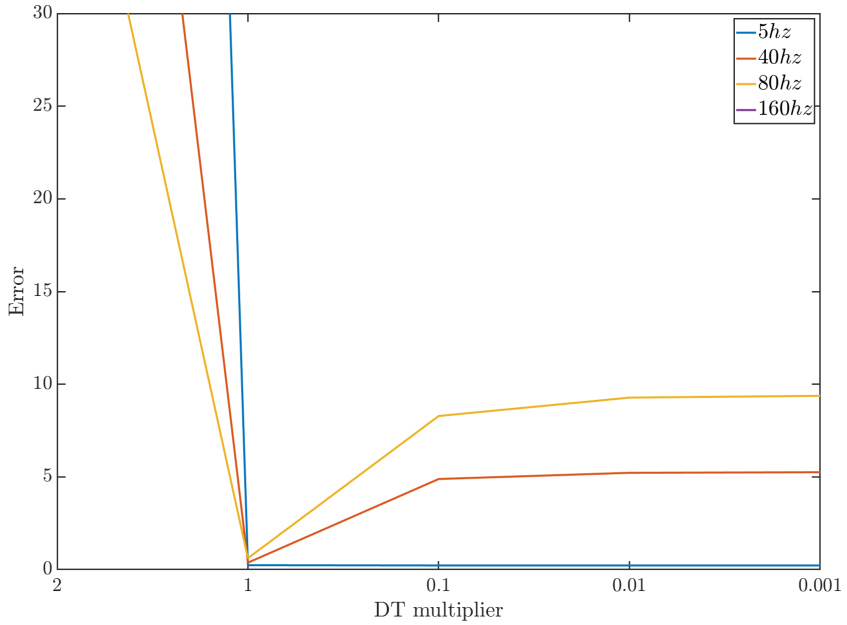


Figure 6.5: Error at a range of time steps

Faster simulations see error increasing as the time step is reduced, but in all cases this converges to a steady value. Figure 6.6 shows the degradation is seen at the point which the valve closes and at the point of flow reversal. This is the fastest simulation tested and shows the worst degradation.

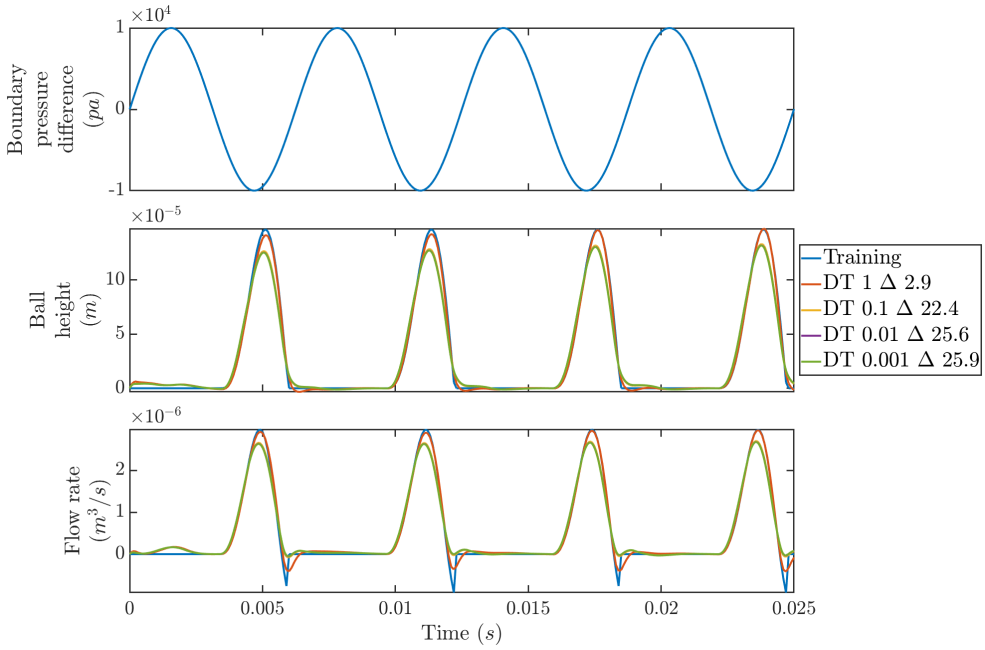


Figure 6.6: Degradation with reduced time step $160Hz$

If the training data set were to include a range of different time step sizes it is hoped that this degradation could be minimised. Including input variables scaled with time step proved not to help in this case, again it is thought this is due to the constant time step size used across all training data.

Chapter 7

Conclusions

Firstly, methods for producing high resolution models of valves were investigated. Two one-way valves were introduced: a disk valve and a ball valve. The disk valve was used to benchmark MPAP against OpenFOAM using steady state simulations. MPAP was then used to investigate the dynamic behaviour of the ball valves. As a result, high resolution models were available for benchmarking and training the reduced order models developed in later chapters. The second chapter presents a physics based reduced order model. This combines several physical phenomena to build up a model that can accurately reproduce the dynamic behaviour of the disk valve based on only steady state simulations. This removes the need for advanced simulation tools to run FSI simulations. However, this methodology proved less effective when applied to the more complex flow behaviour seen in the ball valve. In that case the steady state simulations were a poor representation of the dynamic behaviour. The third chapter describes a neural network approach. This came from the realisation that if one had to run FSI simulations to derive parameters for an accurate physics-based model then one could use those simulations to train a network with no requirement for physical insight. The proposed network architecture is described, and three training

methods compared. Chapter four applies the methodology outlined in the previous chapter to the ball valve. The training, validation and testing results are presented. An investigation into different network architectures was performed. The fifth chapter applied the neural network methodology to a new valve example problem. A workflow is outlined that can be used to train a model and include it as a component in a larger system simulation.

7.1 Achievements and observations

The main objective for this research was to develop a methodology for deriving improved reduced order models for the complex behaviours seen in real world one-way valves. A workflow to achieve this was presented in the previous chapter, it is hoped that this methodology will be put to use at Schaeffler allowing for improved modelling of larger oil systems.

7.2 Future work

The physics based neural network is very tightly linked to the one-way valves considered in this work. On the other hand, the neural network approach presented could be applicable to wide range of physical systems. Further work into the effectiveness of the presented approach to other problems with driving variables and a requirement for variable time step sizes would be valuable. Extending the methodology to other fluid systems such as pumps, and actuators could be a first step. Applying a multi objective optimisation methods to the training of the reduced order model could also show some improvement. Investigation into the selection of training data sets could be done with a view to covering the widest range of conditions with the smallest data set.

Appendices

MATLAB source code for the multi PSO algorithm developed for this work can be found at `https://github.com/IamPete1/Matlab-PSO`.

Bibliography

[1] Schaeffler Automotive. The valve train system. https://library.schaeffler-aftermarket.com/common/library/dbt/download.php?dbt_section=assets&file=enfile&dbt_id=122, 11 2018.

[2] Wikipedia contributors. Paraview — Wikipedia, the free encyclopedia, 2022. [Online; accessed 28-March-2022].

[3] Wulf G Dettmer and Djordje Perić. A new staggered scheme for fluid–structure interaction. *International Journal for Numerical Methods in Engineering*, 93(1):1–22, 2013.

[4] W.G. Dettmer, C. Kadapa, and D. Perić. A stabilised immersed boundary method on hierarchical b-spline grids. *Computer Methods in Applied Mechanics and Engineering*, 311:415–437, 2016.

[5] C. Kadapa, W.G. Dettmer, and D. Perić. A stabilised immersed boundary method on hierarchical b-spline grids for fluid–rigid body interaction with solid–solid contact. *Computer Methods in Applied Mechanics and Engineering*, 318:242–269, 2017.

[6] OpenFOAM. The openfoam foundation. https://openfoam.org/, Dec 2021.

[7] Tobias Holzmann. *Mathematics, Numerics, Derivations and OpenFOAM®*. 11 2019.

[8] OpenFOAMWiki. Openfoam guide/the pimple algorithm in openfoam. `https://openfoamwiki.net/index.php/OpenFOAM_guide/The_PIMPLE_algorithm_in_OpenFOAM`.

[9] OpenFOAM. 4.3 mesh generation with the blockmesh utility. `https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.3-mesh-generation-with-the-blockmesh-utility`.

[10] OpenFOAMWiki. Main contribexamples/axisymmetric. `https://openfoamwiki.net/index.php/Main_ContribExamples/AxiSymmetric`.

[11] J.M. Kirshner. *Design Theory of Fluidic Components*. Elsevier Science, 2012.

[12] Edgar Buckingham. On physically similar systems; illustrations of the use of dimensional equations. *Physical review*, 4(4):345, 1914.

[13] Lawrence F Shampine and Mark W Reichelt. The matlab ode suite. *SIAM journal on scientific computing*, 18(1):1–22, 1997.

[14] Govind S Krishnaswami and Sachin Phatak. The added mass effect and the higgs mechanism. *Resonance*, 25(2):191–213, 2020.

[15] Herve Abdi. A neural network primer. *Journal of Biological Systems*, 2(03):247–281, 1994.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[17] Ken ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806, 1993.

[18] David Hunter, Hao Yu, Michael S Pukish III, Janusz Kolbusz, and Bogdan M Wilamowski. Selection of proper neural network sizes and architectures—a comparative study. *IEEE Transactions on Industrial Informatics*, 8(2):228–240, 2012.

[19] Steve Lawrence, C Lee Giles, and Ah Chung Tsoi. Lessons in neural network training: Overfitting may be harder than expected. In *AAAI/IAAI*, pages 540–545. Citeseer, 1997.

[20] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[21] Martin T Hagan and Mohammad B Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE transactions on Neural Networks*, 5(6):989–993, 1994.

[22] R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[23] David E Goldberg. Genetic algorithms in search, optimization, and machine learning. addison. *Reading*, 1989.

[24] MathWorks. How the genetic algorithm works - matlab. `https://uk.mathworks.com/help/gads/how-the-genetic-algorithm-works.html`.

[25] Kenneth O Stanley. Neuroevolution: A different kind of deep learning. *Obtenido de https://www. oreilly. com/ideas/neuroevolution-a-different-kindof-deep-learning el*, 27(04):2019, 2017.

[26] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[27] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[28] Jun Sun, Bin Feng, and Wenbo Xu. Particle swarm optimization with particles having quantum behavior. In *Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753)*, volume 1, pages 325–331. IEEE, 2004.

[29] PyTorch. Pytorch. https://pytorch.org/.

[30] TensorFlow. Tensorflow. https://www.tensorflow.org/.

[31] Siemens Digital Industries Software. Simcenter amesim: Siemens software. `https://www.plm.automation.siemens.com/global/en/products/simcenter/simcenter-amesim.html`.

[32] K Sanada, CW Richards, DK Longmore, D Nigel Johnston, and Clifford R Burrows. Practical requirements for modelling the dynamics of hydraulic pipelines. In *Proceedings of the JFPS International Symposium on Fluid Power*, volume 1993, pages 657–664. The Japan Fluid Power System Society, 1993.

[33] Mathworks. Simscape. https://uk.mathworks.com/products/simscape.html.