Spectral element methods for simulating pulsatile flow in human arteries

Zhen Xiong

A THESIS

in

Mathematics

For the Graduate Group in Applied Mathematics and Computational Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Master of Arts

2019

Supervisor of Dissertation

Paris Perdikaris, Assistant Professor

Graduate Group Chairperson

Pedro Ponte Castaneda, Professor

ACKNOWLEDGEMENT

I can remember the back of Professor Perdikaris when he wrote clear notes on blackboard during his course ENM 360 Introduction to Data-driven Modeling" in Fall 2018. It's very impressive for me that Professor Perdikaris can always express things very clearly by following the order: problem set up, model and solutions. From then on, I learn from him and work under his supervision in the following semester doing my master thesis.

I was very happy to work with Professor Perdikaris during the time in Spring 2019 and Summer 2019. Although I didn't finish a very good work during this time, it's still very amazing to me that I can start from zero to finally make the simulation happens. This would not happen without the advisement and patient help from Professor Perdikaris. Under his supervision, I learned many technical skill from this project, like how to use remote Linux server to perform numerical simulation, how to use relevant software and some basic knowledge on CFD and so on. However, these are not the most important things I learn from him. One of the most invaluable thing learning from him is the philosophy of "start from small". At the very beginning, Professor Perdikaris first guided me through some basic use of related software of simulation and run simple examples but not directly expose me with more complicated blood flow simulation. I can feel that he wants me to learn something but not quickly get the job done. Also, in later experiments, when I was stuck with difficulties, he always suggested me to take a step back and understand the simpler case first. With this strategy, I accumulated small progress step by step and finally had a better understanding how things work.

From the experience working with Professor Perdikaris, I can strongly feel that what he wants me to do is not quickly achieve success but how to learn something from failure. I have made many careless mistakes and progressed very slowly when doing this project, however, Professor Perdikaris never blame me with that but always remind me what I learn from that. I believe by following this way, people can build a solid foundation and achieve longer and stable success in the future. From my perspective, Professor Perdikaris is a real teacher that can make his student not only a good researcher but also a better person. I have to say that I always owe a sincere thanks to Professor Perdikaris.

In addition to expressing thanks to my advisor Professor Perdikaris, I would also like to thank my partner, my truly friend Shanshan Luo sincerely. Without her accompany and help, I can't finish some difficult coursework here at Upenn and this thesis. Without her accompany, my life maybe be gray living in a foreign country alone.

I would also like to say sincere thanks to Spyros for guiding me to do research in application of deep learning in medical image segmentation, to Professor Monique for giving me a good course in discrete optimization, to yansong for taking me to gym and let me become a body builder, to jiaming for helping me solve many difficult homework questions in some advanced pure math courses.....

At last, I would like to thank my family. Without their general support, I can't have my journey pursuing knowledge and live a good life here at Upenn. Everything I achieved is eventually from their support.

ABSTRACT

Cardiovascular flow simulation using computational method can help medical researchers better understanding of arteries diseases. In this thesis, we will introduce the fundamental work flow of blood flow simulation using spectral element method. We will review both the basic theory and show practical examples to provide new learners a concrete foundation for understanding essential knowledge on blood flow simulation. For theory part, we will introduce basic knowledge in spectral element method for solving differential equations on complex geometry domain and framework for solving Navier-Stokes equation. For practice part, we will show how to use open source software to implement simulation and some advice for avoid potential mistakes. Hope this thesis could be a friendly tutorial for new learners who want to enter the field of blood flow simulation.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1. Introduction

1.1. Background and survey

Heart or arteries related diseases are one of the most dangerous threat to human health. Cardiovascular system is the crucial point to understand those diseases. Some work has been done to reveal and understand cardiovascular system and arteries disease from different perspective, for example, modeling cardiovascular dynamics,[21, 14] view of arteries flow from mechanical perspective[17] and clinical perspective.[15, 1, 9, 6] These studies indicate that understanding some key point of hemodynamic like velocity and pressure of blood flow and wall mechanics of arteries can help people reveal the truth of cardiovascular diseases.

Some invasive measurement method of blood flow monitoring has been proposed to understand the cardiovascular system. One of the traditional method of blood flow measurement is inserting catheters into radial-artery with guide of pulse palpation and anatomic knowledge for placement.[20, 23] Base on this, researchers develop a more advanced method using ultrasound to guide the placement of radial-artery catheters to decrease the rate of failed artery cannulation.[20, 23, 22, 2, 18] Although the invasive measurement method is getting its popularity theses year and can be both accurate and safe to some extent, in addition to be expensive, there is still a possibility to induce risk of complications.[23]

Given the limitations of invasive measurement method, with the development of medical imaging techniques, non-invasive method like X-rays, electro-cardiograms(ECG), computed tomography(CT) and magnetic resonance imaging(MRI) are routinely used in clinical environment.[5] MRI is especially popular today. Unlike X-rays or CT, MRI can get accurate information of inner human body without introducing any radiation, thus safer to patients. Recent research proposed more advanced MRI technique which is called 4D MRI.[13] The 4D MRI can combine temporal evolution of blood flow and spatial geometry of blood vessel together to represent the full three-dimension velocity flow field.[8]

1.2.1 Motivation for using CFD in blood flow simulation

Although imaging technique like 4D MRI can provide us velocity profile of the flow and accurate geometry structure of the vessel, some important diagnosis indicators like pressure, vorticity can not be measured by these non-invasive methods. In order to measure those important variables in cheaper and safer way, computer numerical simulation methods could be our choice. With combining the geometry information from MRI image, computational fluid dynamics simulation can numerically calculate those important variables out. Such method is also called Image based computational fluid dynamics(IB-CFD).[15]

The work flow of IB-CFD is as following: people first take MRI images of aorta and using geometric modeling method to build geometry model of the vessel which is the computational domain for simulation, then we apply computational fluid dynamics algorithms to perform simulation. The situations vary among different patients, however, MRI image will exactly keep the patient-specific vessel geometry information and this makes IB-CFD adapts each individual case and makes it a very practical method in hospital environment.

In addition to clinical reason, IB-CFD method can be also meaningful to pure research purpose. With the stable and reproducible numerical results in the same condition. We can change the parameters like boundary conditions or initial conditions to provide more insight of hemodynamics and physics.

1.2.2 Motivation for using spectral element method

Finite element method, was first introduced into structural mechanics and then it got success in fluid mechanics. By subdividing the computational domain and interpolating with low order functions, finite element method is good at solving equations with complex geometry domain. However, due to the low order approximation, when we deal with time-dependent simulation of Navier-Stokes equations, in order to keep the same error level, we need to increase the number of subdomains for longer time simulation, which require more computational resources.[17] On the other hand, spectral method using a high order truncated series expansion for approximation can mitigate this increasing resolution demand due to its exponential convergence property, thus is computationally more efficient. However, spectral method may lose its advantage when the computational domain is irregular.[17] By combing with finite element method, we can easily keep the advantage of the spectral method because each element is a relatively regular domain and we can apply spectral in each element. This is also called spectral element method, which provide us a better strategy for convergence.

1.3. Open Challenge

Although using computational method to perform simulation of blood flow has many advantages as mentioned previously, some challenges still remain.

The mesh quality of the geometry model is essential for a successful numerical simulation, however, it's hard to define what is a good quality mesh for the geometry model(computational domain). Although there are many geometry processing techniques been developed, there is no benchmark for judging which is better. Although, we can also notice some limitations in existing open source geometry processing software like Gmsh.[4] For example, Gmsh can not do remeshing for complex or large geometry model. Those limitations may let geometry modeling and processing laborious. The way to fully autonomous geometry processing is still long.

The numerical simulation also has its inherent weakness, that is requiring many computing resources and time consuming. For some large scale cardiovascular flow simulation which requires computing many variables, we need distributed computing systems or super computers. However, this is not always available for all the researchers. Also, if the computing resources is limited, the simulation can take very long time. This may inconvenient in real clinical setting because usually patients need diagnosis in short time.

Performing the simulation itself is not an easy thing as well. Even with existing software package, people need to accumulate practical knowledge in trouble shooting of simulation blowing up. The reasons of failed simulation can from any steps and may be difficult to debug.

Last, validation of the simulation results is still a problem. Data getting from in-vivo methods can be used for validation of the simulation results. However, this may be hindered ethical considerations of invasive data collection.[31] Without invasive data, validation of simulation result can be less reliable.

1.4. Overview of the thesis

This thesis will show entry level blood flow simulation from new learners' perspective including both the fundamental theory and practical experiment using open source software. The second part will introduce related basic theory in blood flow simulation. In the third part, we will show concrete numerical simulation examples. The last part will analyze the limitation of this thesis and discuss the future work.

In the theory part, we will introduce the basic knowledge of spectral element methods, velocity correction scheme for solving Navier-Stokes equation and related model with pulsatile flow. In the experiment part, we will use open source Nektar++[3] perform the simulation. Nektar++ is a scalable framework for solving partial differential equations using spectral element methods. Nektar++ mainly contains three kinds of utilities: pre-processing utility, numerical solver utility and post-processing utility. The pre-processing utility can deal with mesh file from geometry processing software like Gmsh and feed the geometry information into solver utility so that users can perform simulation on the geometry. Post-processing can manipulate with simulation results and let the results can be visualized by other visualization software like Paraview.[4]

Blood flow simulation domain involves a broad range of knowledge and accumulates many specific knowledge. For this reason, this thesis hopes to provide a friendly tutorial for people who wants to get into this field and see some simple but useful instances. For more advanced information, readers need to explore deeper with the reference materials.

2. Method

2.1. Introduction to spectral/hp element method

Here we will introduce Spectral/hp element method. We will start from classical finite element method, then the high-order finite element method. This will prepare the foundation for understanding Spectral/hp element method.

2.1.1 Finite element method

The two main ideas of finite element method are discretization and approximation. In many real problems like structural mechanics that require to solve (partial)differential equation on a complex geometry with boundary conditions, it's very difficult to solve directly. The finite element method will first divide computational domain into a collection of subdomains which are called elements. This step is called discretization. In practice, this step need us to represent the geometry by mesh. The elements of the mesh are connected by nodes. In 2-dimension case, people commonly use triangle element, quadrilateral element or polygonal element in mesh. In 3-dimension case, we can use tetrahedron element, hexahedron element or polyhedron element. The Mesh can be very important to finite element method. A good mesh can efficiently capture local structure, thus may help to decrease errors. This is one of the advantage of finite element method. In the second step. the approximation will happen in element level. Unlike the global case, finite element method will divide the total problem into simpler case which will be solved in each element and then recombine the solution from each element into the final answer. This can avoid directly solving the very difficult problem, which is another advantage of finite element method. A more comprehensive introduction of finite element method can be found in.[16, 53, 54, 55, 56, 57, 58, 59, 60]

2.1.2 High-order finite element method

In the section 2.1.1, we already know some of the advantage that general finite element method has. However, in the classical approach, the approximation in each element is using linear interpolation functions. When we require higher level of accuracy, we need high-order finite element method. High-order finite element method will use high order polynomials to approximate unknown solution in each element. However the name 'high-order' in the method doesn't mean using high order polynomials but refer to the order of convergence rate of approximation with respect to mesh-refinement. In other words, the solution of high-order finite element method can converge to the real solution faster comparing with the classical one. There are generally three type of high-order finite element method: h-version finite element method, p-version finite element method and hp-version finite element method.

2.1.2.1 The h-version finite element method The h-version finite element method fix the degree of polynomial interpolation functions and enhance accuracy by means of mesh refinement. The refinement of mesh doesn't need to be globally. This can happen locally when the part need Here h in the name of 'h-version' means the characteristic length of mesh. Intuitively, this method will achieve higher level accuracy when it uses a denser mesh.

2.1.2.2 The p-version finite element method The p-version finite element method only change the degree of polynomial basis functions and keeps the mesh refinement fixed to enhance accuracy. The degree or order of polynomial in different element is unnecessary the same. For some parts of the geometry which require higher accuracy may have higher order interpolation polynomials. Here the p in the name 'p-version' means the degree or order of the interpolation polynomial. When people mention p-version finite element method, we generally assume that $p > 1$.

2.1.2.3 The hp-version finite element method The hp-version finite element method is the combination of h-version and p-version finite element method. The work[10] first discover the method and found it converges exponentially. The method is very flexible. It can do the h-refinement and p-refinement at the same time. For example, the method can divide an element into smaller subelements(h-refinement) and increase the polynomial degrees inside each subelements(p-refinement). Also, there are many combinations for polynomial degrees. For example, if an element is divided into three subelements and the polynomial degrees are allowed to vary by two, then in each subelements we have three choices of polynomial degree, which give us 27 refinement candidates.

2.1.3 Spectral methods

Spectral methods[61, 62, 63, 64, 65, 66, 67] are a class of techniques to solve differential equations. The main idea of spectral method are approximating the solution using a sum of certain basis functions like Fourier series with coefficients which can be adjusted to satisfy the differential equations as well as possible. There are two main differences between spectral methods and finite element methods. First, spectral methods use a global approach(the sum of basis functions is applied on the whole computational domain) while the interpolation polynomial of finite element methods have local support(each element has its own polynomial). Second, spectral methods generally use higher order expansions than finite element methods do. In traditional finite element community, four-order elemental expansions are already considered as high order but in spectral methods community, sometimes a sixteen-order global expansion may be considered as a low-order approximation. With global high-order expansions, spectral methods can converge extremely fast(exponentially) when the solution is smooth. However, spectral methods are not good at dealing with computational domains with complex geometries.

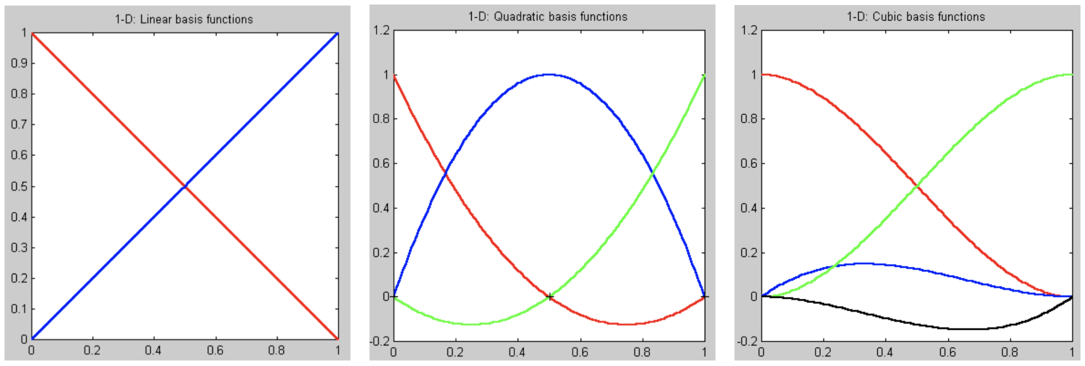


Figure 1: Frequently used basis functions for classical finite element methods.[27]

2.1.4 Spectral/hp element methods

Previous sections build the foundation for us to understanding Spectral/hp element methods by introducing hp-version finite element method and spectral methods. Roughly speaking, finite element methods are adaptive to complex geometries but may not be accurate with low-order expansions, however, spectral methods taking high-order global expansions can achieve high-order accuracy but may be inaccurate with complex geometries. Combining the advantage of both method, Spectral/hp element methods use high-order polynomial in each elements can deal with complex geometries better and converge to true solution very fast. Formally speaking, spectral/hp element method is a formulation of hp-version finite element method using high-order piecewise polynomial functions(like Lagrange polynomial) as basis functions. Comparing Figure 1 and Figure 2, we can notice the difference between basis function of classical finite element method and spectral element method and can expect that spectral element methods have better ability to fit the solution.

2.2. Introduction to weak form of differential equation

A weak solution of a differential equation is a function for which the derivatives may not all exist but can still satisfy the equation in some precisely defined sense. More intuitively, the weak solution is not as smooth as the real solution or it can not differentiate as many times as the real solution but they are still very close. In fact, this is the advantage of weak

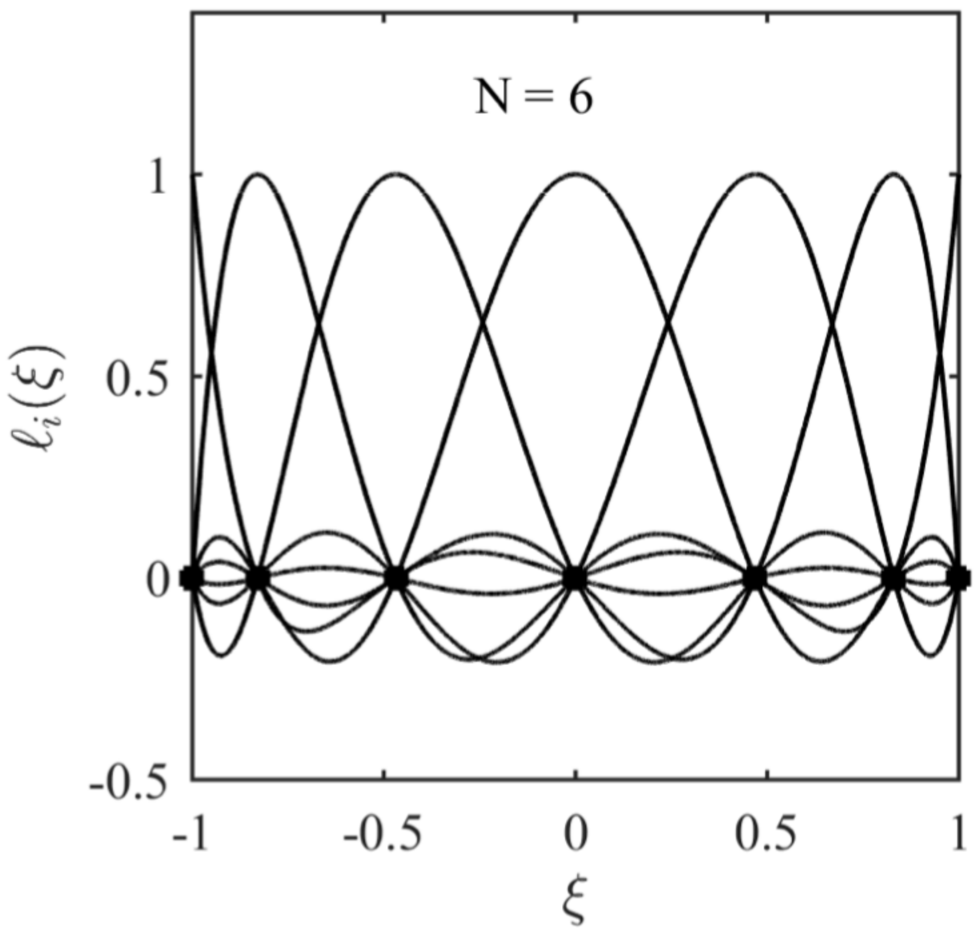


Figure 2: Lagrange polynomial basis functions of order six for spectral element methods. Although the order is six, it's still a relative low-order polynomial expansions in spectral element methods.[28]

solution because many real problems don't admit very smooth solutions and the weak form of equation will be the only way to solve the problem. In finite(spectral) element method, weak form is very commonly used. One of the most popular way to derive the weak form of differential equation in finite(spectral) element method is using Galerkin formulation which will be introduced next.

2.3. Introduction to Galerkin formulation

In section 2.2, we introduced the weak solution of differential equations. This section will introduce the most commonly used way, the Galerkin method, to derive the weak form of partial differential equations and convert continuous operator problems(differential equations) into discrete problems(matrix computation). The details of Galerkin formulation is far beyond of our scope and here we will use an example to illustrate how the Galerkin method works. More detailed information of Galerkin method is in.[12]

We divide the description of Galerkin method into three steps. The first step is deriving the weak form of the equation. Consider a steady linear differential equation in a domain Ω denoted by

$$L(u) = f \tag{1}$$

subject to appropriate boundary conditions. Here the f is a known function, L denote linear differential operator and u is the function we want to solve. To derive the weak form, the Galerkin method multiplies a test function v from test function space V on both sides of the equation and then do integration over the entire domain Ω, which arrives: Find a $u \in U$ such that

$$\int_\Omega vL(u)dx = \int_\Omega vf dx, \quad \forall v \in V \tag{2}$$

where U is trial space containing possible solution function. For now, instead of solving the original differential equation, we need to solve its weak form which is an integral equation now. In order to make things succinct, we rewrite the equation (2) as

$$a(v,u) = (v,f), \quad \forall v \in V \tag{3}$$

where $a(v,u)$ is a bilinear form representing $\int_\Omega vL(u)dx$ and (v,f) represents $\int_\Omega vf dx$ which is also called inner product of v and f.

The second step of Galerkin method is dimension reduction of trial(U) and test(V) space. Rather than using infinite dimension function space, we cut trial and test space into finite dimension function which is spanned by finite numbers of basis function $\Phi_n (n \in N)$. In the classical Galerkin method, we adopt same set of basis functions for both trial and test space. The reduced finite dimension trial and test space are denoted by U^δ and V^δ With finite dimension trial space, the approximation of solution u is denoted by u^δ which is represented as linear combination of basis functions in trial space:

$$u^\delta = \sum_{n \in N} \Phi_n \hat{u}_n \tag{4}$$

where $\hat{u}_n$ are the coefficients we need to solve. With the new presentation of equation (4), we can update equation (3) as:

$$a(v^\delta, u^\delta) = (v^\delta, f) \tag{5}$$

If we substitute u^δ in equation (5) with equation (4), then we get: Find $\hat{u}_n$such that,

$$\sum_{n \in N} a(v^\delta, \Phi_n)\hat{u}_n = (v^\delta, f), \quad \forall v^\delta \in V^\delta \tag{6}$$

Since v^δ is also the linear combination of basis function in test space, we can write equation (6) into an equivalent form: Find $\hat{u}_n$such that,

$$\sum_{n \in N} a(\Phi_m, \Phi_n)\hat{u}_n = (\Phi_m, f), \quad \forall m \in N \tag{7}$$

The third step of Galerkin method will transform (7) into matrix form so that we can get the final answer by computers. If we let $\hat{\mathbf{u}}$ be the vector of coefficients $\hat{u}_n$, $\mathbf{A}$ be a matrix with elements

$$\mathbf{A}[m][n] = a(\Phi_m, \Phi_n) = \int_\Omega \Phi_m L(\Phi_n) dx \tag{8}$$

and $\mathbf{f}$ be the vector with elements

$$\mathbf{f}[m] = (\Phi_m, f) = \int_\Omega \Phi_m f dx, \tag{9}$$

then we can just solve $\hat{\mathbf{u}}$ in the following matrix equation:

$$\mathbf{A}\hat{\mathbf{u}} = \mathbf{f}. \tag{10}$$

2.4. Numerical method for integration

Since we need to evaluate integral like (8) and (9) in the Galerkin method, we need techniques of numerical integration. There are plenty of numerical method for integration. In this thesis, we focus our attention on introducing the way that Nektar++ employs to compute integration numerically by an one-dimension example. Consider the following integral:

$$\int_{-1}^{1} u(\xi) d\xi, \tag{11}$$

here the $u(\xi)$ may be made up of products of polynomial bases just like (8) and (9). The main numerical integration method used in Nektar++ is called Gaussian Quadrature. First, it represent the integrand as a Lagrange polynomial using q points ξ_i,

$$u(\xi) = \sum_{i=0}^{q-1} u(\xi_i) h_i(\xi) + \varepsilon(u), \tag{12}$$

where $\varepsilon(u)$ is the approximation error. Second, instead of evaluating (11), we substitute (12) into (11) and get:

$$\int_{-1}^{1} u(\xi)d\xi = \sum_{i=0}^{q-1} u(\xi_i)w_i + R(u), \tag{13}$$

where

$$w_i = \int_{-1}^{1} h_i(\xi)d\xi, \tag{14}$$

$$R(u) = \int_{-1}^{1} \varepsilon(u)d\xi \tag{15}$$

In this step, we notice that $u(\xi_i)$ are known value so if we can compute (14) and (15), we can compute (13). For (14), since $h_i(\xi)$ are Lagrange basis polynomial, which means that if we know all the ξ_i then we know all the $h_i(\xi)$. For (15), if $u(\xi)$ is a polynomial of order less than $(q-1)$, we would expect $R(u) = 0$ since we use polynomial of order $(q-1)$ to approximate it. In fact, Gauss recognized that there exists a better choice of ξ_i that can permit (13) with $R(u) = 0$ when $u(\xi)$ is a polynomial higher order than $(q-1)$. So if we choose a good set of ξ_i, we can compute (13) appropriately. In Nektar++, we have three types of Gauss quadrature which choose the set of ξ_i differently. All those three types of Gauss quadrature are related with Jacobi polynomial. Jacobi polynomial[34, 35, 36, 37, 38, 39] is denoted as $P_P^{\alpha,\beta}$ where α and β are two parameters and P is the order of Jacobi polynomial. We denote $\xi_{i,P}^{\alpha,\beta}$ as P zeros of Jacobi polynomial $P_P^{\alpha,\beta}$. Based on this, we can introduce all three types of Gauss quadrature now.

The first type of Gauss quadrature is called Gauss-Legendre quadrature, where

$$\xi_i = \xi_{i,q}^{0,0} \qquad i = 0, ..., q-1 \tag{16}$$

$$w_i^{0,0} = \frac{2}{[1-(\xi_i)^2]}[\frac{d}{d\xi}(L_q(\xi))|_{\xi=\xi_i}]^{-2} \qquad i = 0, ..., q-1 \tag{17}$$

$$R(u) = 0 \qquad if\ u(\xi) \in P_{2q-1}([-1,1]) \tag{18}$$

where P_{2q-1} denotes polynomials with order not higher than 2q-1 and $L_q(\xi)$ denotes Legendre polynomial($L_q(\xi) = P_q^{0,0}(\xi)$).

The second type of Gauss quadrature is called Gauss-Radau-Legendre quadrature, where

$$\xi_i = \begin{cases} -1 & i=0 \\ \xi_{i-1,q-1}^{0,1} & i=1,...,q-1 \end{cases} \tag{19}$$

$$w_i^{0,0} = \frac{(1-\xi_i)}{q^2[L_{q-1}(\xi_i)]^2} \qquad i = 0, ..., q-1 \tag{20}$$

$$R(u) = 0 \qquad if\ u(\xi) \in P_{2q-2}([-1,1]) \tag{21}$$

The third type pf Gauss quadrature is called Gauss-Lobatto-Legendre, where

$$\xi_i = \begin{cases} -1 \quad i = 0 \\ \xi_{i-1,q-2}^{1,1} \quad i = 1, ..., q-2 \\ 1 \quad i = q-1 \end{cases} \tag{22}$$

$$w_i^{0,0} = \frac{2}{q(q-1)[L_{q-1}(\xi_i)]^2} \qquad i = 0, ..., q-1 \tag{23}$$

$$R(u) = 0 \qquad if\ u(\xi) \in P_{2q-3}([-1,1]) \tag{24}$$

Although the above equations look intimidating, they just specify the set of ξ_i to choose and also calculate w_i so we can calculate (13) numerically(we can ignore $R(u)$). In finite(spectral) element method, we can't always do integration on [-1,1](local region). The way to fix this is to map the original region of integration into local region and do the calculation in the way mentioned above and then map the result back to the original region by Jacobian mapping.

2.5. Velocity correction scheme for solving unsteady incompressible Navier-Stokes equation

Unsteady incompressible Navier-Stokes equation is the governing equation for viscous Newtonian fluids. In this thesis, we regard blood flow in aorta as incompressible viscous Newtonian fluid where the density and viscosity are constant. In this section, we will introduce a method called velocity correction scheme to solve unsteady Navier-Stokes equation, which is written as:

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} \\ \mathbf{u} = \mathbf{u}_{\mathfrak{D}} \quad \text{on}\, \Gamma_{\mathfrak{D}} \\ \dfrac{\partial \mathbf{u}}{\partial n} = \mathbf{u}_{\mathfrak{n}} \quad \text{on}\, \Gamma_{\mathfrak{n}} \end{cases} \tag{25}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{26}$$

where $\mathbf{u}$ is velocity(need to solve) of the fluid, p is pressure(this pressure is the original pressure divided by density, but we still denote this as pressure and we need to solve it) of the fluid and ν is the kinematic viscosity(a constant obtained from dividing dynamic viscosity by density). Usually we also specify the boundary conditions The solution of (25) should also satisfy (26) because (26) means divergence of fluid velocity is zero, which is the incompressible constraint(more details can be found from[40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]).

The mean idea of velocity correction scheme is first solve pressure by solving a weak form of Poisson equation and then use the information of pressure to solve velocity by solving Helmholtz equation. We will divide the whole process in four subsections.

2.5.1 Temporal discretization of the equation

In order to derive weak form pf Poisson equation to solve pressure, we need to first discretize equation (25). In order to write things conveniently, we denote $\mathbf{u} \cdot \nabla \mathbf{u}$ as $N(\mathbf{u})$(also called as convective term) and denote $\nabla^2 \mathbf{u}$ as $L(\mathbf{u})$. So the equation (25) becomes:

$$\frac{\partial \mathbf{u}}{\partial t} + N(\mathbf{u}) = -\nabla p + \nu L(\mathbf{u}) \tag{27}$$

Following the way that[19] discretize equation (27) using backward differentiation formula[51,52] and by representing the convective term explicitly using a polynomial extrapolation from previous time-steps,[19] we can represent equation (27) in time step n+1 as:

$$\frac{\gamma_0 \mathbf{u}^{n+1} - \mathbf{u}^{+}}{\Delta t} + N^* = -\nabla p^{n+1} + \nu L(\mathbf{u}) \tag{28}$$

Here the $\mathbf{u}^{+}$ is defined as:

$$\mathbf{u}^{+} = \sum_{q=0}^{J_i - 1} \alpha_q \mathbf{u}^{n-q} \tag{29}$$

and N^* is defined as:

$$N^* = \sum_{q=0}^{J_e - 1} \beta_q N(\mathbf{u}^{n-q}) \tag{30}$$

where J_i, J_e are integration orders and $\alpha.\beta, \gamma$ are coefficients.

Now we rewrite the equation (28) as:

$$\frac{\gamma_0 \mathbf{u}^{n+1} - \gamma_0 \bar{\mathbf{u}}^{n+1} + \gamma_0 \bar{\mathbf{u}}^{n+1} - \mathbf{u}^{+}}{\Delta t} + N^* + \nabla p^{n+1} - \nu L(\mathbf{u}) + \nu(\nabla \times \nabla \times \mathbf{u})^* - \nu(\nabla \times \nabla \times \mathbf{u})^* = 0 \tag{31}$$

where $(\nabla \times \nabla \times \mathbf{u})^*$ is approximation of $(\nabla \times \nabla \times \mathbf{u})$ using the same style of N^*. Then we split the equation (31) into two parts:

$$\frac{\gamma_0 \bar{\mathbf{u}}^{n+1} - \mathbf{u}^{+}}{\Delta t} + N^* + \nabla p^{n+1} + \nu(\nabla \times \nabla \times \mathbf{u})^* = 0 \tag{32}$$

$$\frac{\gamma_0 \mathbf{u}^{n+1} - \gamma_0 \bar{\mathbf{u}}^{n+1}}{\Delta t} - \nu L(\mathbf{u}) - \nu(\nabla \times \nabla \times \mathbf{u})^* = 0 \tag{33}$$

where we will solve equation (32) to get pressure and then solve equation (33) to get velocity. In the next subsection, we will introduce how to derive weak form of Poisson equation and then formulate the weak form of equation (32) in the third subsection.

2.5.2 Derivation of weak form of Poisson equation.

The Poisson equation is written as the form:

$$-\nabla^2 u = f \tag{34}$$

where f is a known function and we need to solve u. Like the Galerkin method, to derive the weak form of this equation, we require the inner product of terms in the equation with test function:

$$\int_\Omega (\nabla^2 u + f)v = 0 \tag{35}$$

where Ω is the integration field. Here we need a trick to change the form of equation (35). According to rule of derivative of product, we have:

$$\nabla \cdot (v\nabla u) = \nabla v \cdot \nabla u + v\nabla^2 u \tag{36}$$

If we do the integration of equation (36) on both sides, then we obtain:

$$-\int_\Omega v\nabla^2 u = \int_\Omega \nabla v \cdot \nabla u - \int_\Omega \nabla \cdot (v\nabla u) \tag{37}$$

Recall the divergence theorem which is stated as:

$$\int_\Omega (\nabla \cdot F) = \int_{\partial\Omega} (F \cdot \mathbf{n}) \tag{38}$$

where $\partial\Omega$ denote the boundary of integration field Ω and $\mathbf{n}$ denote outward pointing unit normal field of the boundary $\partial\Omega$. Now we can apply divergence theorem to the term $\int_\Omega \nabla \cdot (v\nabla u)$ and equation (37) becomes:

$$-\int_\Omega v\nabla^2 u = \int_\Omega \nabla v \cdot \nabla u - \int_{\partial\Omega} v\nabla u \cdot \mathbf{n} \tag{39}$$

In fact, according to directional derivative, $\nabla u \cdot \mathbf{n}$ is the derivative of u in direction of vector $\mathbf{n}$. So we can rewrite the equation (39) as:

$$-\int_\Omega v\nabla^2 u = \int_\Omega \nabla v \cdot \nabla u - \int_{\partial\Omega} v\frac{\partial u}{\partial \mathbf{n}} \tag{40}$$

Combining the equation (35), we can substitute the left side of equation (40) and obtain:

$$\int_\Omega vf = \int_\Omega \nabla v \cdot \nabla u - \int_{\partial\Omega} v\frac{\partial u}{\partial \mathbf{n}} \tag{41}$$

which can be written in classical form:

$$\int_\Omega \nabla v \cdot \nabla u = \int_\Omega vf + \int_{\partial\Omega} v\frac{\partial u}{\partial \mathbf{n}} \tag{42}$$

The equation (42) is regarded as weak form of Poisson equation. When we have Neumann boundary conditions, $\frac{\partial u}{\partial \mathbf{n}}$ is known so that we can solve equation (42).

2.5.3 Solve pressure from weak form of Poisson equation

The algorithms to solve weak form of Poisson equation already exist. The problem is we need to show that we can derive weak form of Poisson equation from equation (32). This issue will be solved in this section.

Now we look back equation (32) and rewrite it as:

$$\nabla p^{n+1} = \frac{\mathbf{u}^+ - \gamma_0\bar{\mathbf{u}}^{n+1}}{\Delta t} - N^* - \nu(\nabla\times\nabla\times\mathbf{u})^* \tag{43}$$

Following the way in section 2.5.2, we multiply the gradient of some test function and do the integration on both sides:

$$\int_\Omega \nabla\phi\cdot\nabla p^{n+1} d\Omega = \int_\Omega \Big[\frac{\mathbf{u}^+ - \gamma_0\bar{\mathbf{u}}^{n+1}}{\Delta t} - N^* - \nu(\nabla\times\nabla\times\mathbf{u})^*\Big]\cdot\nabla\phi d\Omega \tag{44}$$

We can apply divergence theorem to the right hand side of this equation to obtain a more useful form. The original divergence theorem states the equation (38). However, if we substitute F with Fg, we obtain:

$$\int_\Omega \nabla \cdot Fg = \int_\Omega F \cdot (\nabla g) + g(\nabla \cdot F) = \int_{\partial\Omega} gF \cdot \mathbf{n} \tag{45}$$

With a little operation of equation (45), we have:

$$\int_\Omega F \cdot (\nabla g) = -\int_\Omega g(\nabla \cdot F) + \int_{\partial\Omega} gF \cdot \mathbf{n} \tag{46}$$

Comparing the right hand side of equation (44) and the left hand side of equation (46), if we regard the test function ϕ and $\left[\frac{\mathbf{u}^+ - \gamma_0 \bar{\mathbf{u}}^{n+1}}{\Delta t} - N^* - (\nabla \times \nabla \times \mathbf{u})^*\right]$ and in (44) as g and F in (46) respectively, we have:

$$\int_\Omega \left[\frac{\mathbf{u}^+ - \gamma_0 \bar{\mathbf{u}}^{n+1}}{\Delta t} - N^* - (\nabla \times \nabla \times \mathbf{u})^*\right] \cdot \nabla\phi d\Omega = -\int_\Omega \phi(\nabla \cdot F) d\Omega + \int_{\partial\Omega} \phi F \cdot \mathbf{n} dS \tag{47}$$

where

$$F = \left[\frac{\mathbf{u}^+ - \gamma_0 \bar{\mathbf{u}}^{n+1}}{\Delta t} - N^* - (\nabla \times \nabla \times \mathbf{u})^*\right] \tag{48}$$

However, since we require $\bar{\mathbf{u}}^{n+1}$ should follow the incompressible restriction($\nabla \cdot \bar{\mathbf{u}}^{n+1} = 0$ and $\nabla \cdot (\nabla \times \nabla \times \mathbf{u}) = 0$(so $\nabla \cdot (\nabla \times \nabla \times \mathbf{u})^* = 0$), the equation (47) can be reduced to:

$$\int_\Omega \left[\frac{\mathbf{u}^+ - \gamma_0 \bar{\mathbf{u}}^{n+1}}{\Delta t} - N^* - \nu(\nabla \times \nabla \times \mathbf{u})^*\right] \cdot \nabla\phi d\Omega = \int_\Omega \frac{N^* \Delta t - \mathbf{u}^+}{\Delta t} \cdot \phi + \int_{\partial\Omega} \phi F \cdot \mathbf{n} dS \tag{49}$$

Combine equations (44) and (49) we obtain:

$$\int_\Omega \nabla\phi \cdot \nabla p^{n+1} d\Omega = \int_\Omega \frac{N^* \Delta t - \mathbf{u}^+}{\Delta t} \cdot \phi + \int_{\partial\Omega} \phi \left[\frac{\mathbf{u}^+ - \gamma_0 \bar{\mathbf{u}}^{n+1}}{\Delta t} - N^* - \nu(\nabla \times \nabla \times \mathbf{u})^*\right] \cdot \mathbf{n} dS \tag{50}$$

From equation (50), we notice that in fact it is in the same form as equation (42). The $u, v, f, \frac{\partial u}{\partial \mathbf{n}}$ in equation (42) is corresponding to $p^{n+1}, \phi, \frac{N^* \Delta t - \mathbf{u}^+}{\Delta t}, \left[\frac{\mathbf{u}^+ - \gamma_0 \bar{\mathbf{u}}^{n+1}}{\Delta t} - N^* - (\nabla \times \nabla \times \mathbf{u})^*\right]$ in equation (50). This means equation (50) is the weak form of Poisson equation. Usually we will specify Neumann boundary condition for pressure, we can just let $\left[\frac{\mathbf{u}^{+}-\gamma_0\bar{\mathbf{u}}^{n+1}}{\Delta t} - N^* - (\nabla \times \nabla \times \mathbf{u})^*\right]$ be equal to $\frac{\partial u}{\partial \mathbf{n}}$ which is given. In this way, solving the weak form of Poisson equation can give the information of p at $n+1$ time step.

2.5.4 Solving Helmholtz equation for computing velocity field

In this section, we transform the equation (33) into Helmholtz equation which can solve velocity. The Helmholtz equation is written as:

$$\nabla^2 \mathbf{u} - \lambda \mathbf{u} = f \tag{51}$$

where λ is coefficient and f is a function. Now we discuss how to derive Helmholtz equation from equation (33).

From equation (32), we have:

$$\frac{\gamma_0 \bar{\mathbf{u}}^{n+1}}{\Delta t} = \frac{\mathbf{u}^{+}}{\Delta t} - \nabla p^{n+1} - \nu(\nabla \times \nabla \times \mathbf{u})^* - N^* \tag{52}$$

So for equation (33), we can substitute $\frac{\gamma_0 \bar{\mathbf{u}}^{n+1}}{\Delta t}$ from equation (52) and obtain:

$$\frac{\gamma_0 \mathbf{u}^{n+1} - \mathbf{u}^{+}}{\Delta t} + \nabla p^{n+1} + N^* - \nu L(\mathbf{u}^{n+1}) = 0 \tag{53}$$

Recalling the definition of $L(\mathbf{u})$, this equation can be rewrite as:

$$\nabla^2 \mathbf{u} - \frac{\gamma_0 \mathbf{u}^{n+1}}{\nu \Delta t} = \nabla p^{n+1} - \frac{\mathbf{u}^{+}}{\Delta t} + N^* \tag{54}$$

which is in fact the Helmholtz equation where ∇p^{n+1} has already been solved from equation

(32). So by solving the problem :

$$
\begin{cases}
\nabla^2 \mathbf{u} - \dfrac{\gamma_0 \mathbf{u}^{n+1}}{\nu \Delta t} = \nabla p^{n+1} - \dfrac{\mathbf{u}^+}{\Delta t} + N^* \\
\mathbf{u}^{n+1} = \mathbf{u}_{\mathfrak{D}} \quad \text{on } \Gamma_{\mathfrak{D}} \\
\dfrac{\partial \mathbf{u}^{n+1}}{\partial n} = \mathbf{u}_{\mathfrak{n}} \quad \text{on } \Gamma_{\mathfrak{n}}
\end{cases}
\tag{55}
$$

we can get velocity field at time step $n + 1$. From section 2.5.1 to section 2.5.4, we present the whole procedure of velocity correction scheme for solving governing equation of blood flow simulation. More detailed information on velocity correction scheme can be found in.[19, 7, 11]

2.5.5 Space discretization using Spectral elements methods

In previous sections, we described the framework and derivation of velocity correction scheme. However, we didn't talk about how to solve the weak Poisson form(32) and Helmholtz equation(33) with spectral element methods(space discretization). People may find the page 127-128 in[33] and page 74-75 in[32] very useful.

2.6. Numerical experiments of two dimension Kovasznay flow using Nektar++

In this section, we will show some numerical experiments that using the open source software Nektar++ to perform incompressible fluid simulation. The general work flow of doing such numerical experiments is writing mesh files which can define the geometry and Nektar++ condition file which can define the setup of the simulation and then pass these two files into Nektar++ solver which will perform the numerical computing(simulation). After the simulation is down, we can use Nektar++ post-processing utility to transform the result file into the file that can be read in Paraview(an open source visualization software) to let people actually see the result.

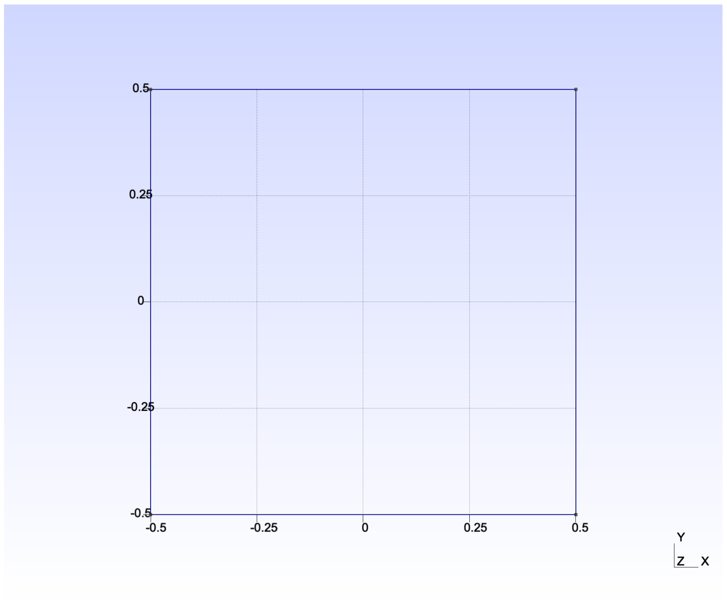


Figure 3: The 2D geometry for Kovasznay flow simulation.

2.6.1 Geometry and mesh of the simulation

In order to do numerical simulation, the first thing is to define computational domain(geometry). One of the mesh generation software for Nektar++ is called Gmsh. The more detailed information on Gmsh can be found at http://gmsh.info/. The usage of Gmsh is beyond the scope of this thesis.

The Figure 3 shows a two dimensional geometry model in the graphical user interface of Gmsh. Base on this geometry model, Gmsh can create mesh on it. The Figure 4 shows the mesh for our computational domain. This mesh file is in format .msh which can not be read directly in Nektar++. By using the preprocessing utility Nekmesh of Nektar++:

```
Nekmesh mesh_name.msh mesh_name.xml
```

in command line, we can transform the .msh file into .xml which can be passed to Nektar++ solver. Here the Nekmesh is an executable file and its location in Nektar++ folder is:

```
nektar++/build/dist/bin/Nekmesh
```

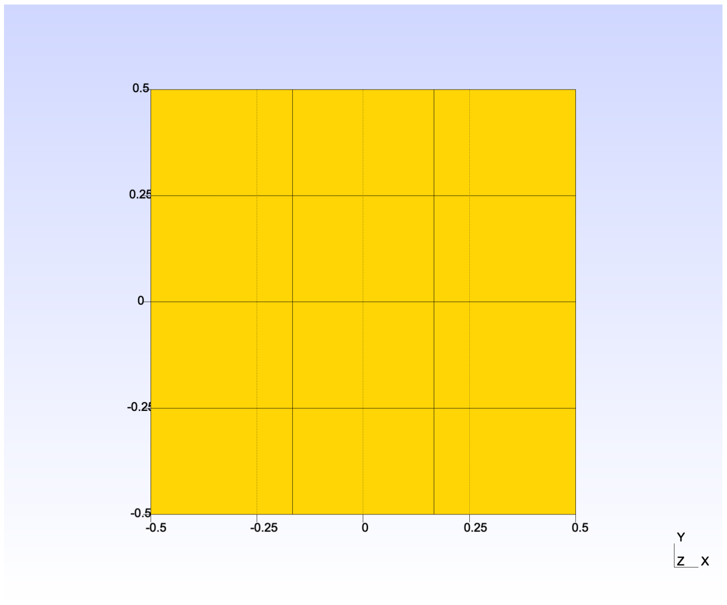


Figure 4: The 2D mesh for Kovasznay flow simulation.

so user should add the path to Nekmesh when using this utility.

This .xml format mesh file consist four parts of information. First, it defines the location of vertex by specifying their coordinates:

```
<VERTEX>
            <V ID="0">-5.00000000e-01 5.00000000e-01 0.00000000e+00</V>
            <V ID="1">-5.00000000e-01 2.50000000e-01 0.00000000e+00</V>
            <V ID="2">-1.66666667e-01 2.50000000e-01 0.00000000e+00</V>
            <V ID="3">-1.66666667e-01 5.00000000e-01 0.00000000e+00</V>
            <V ID="4">1.66666667e-01 2.50000000e-01 0.00000000e+00</V>
            <V ID="5">1.66666667e-01 5.00000000e-01 0.00000000e+00</V>
            <V ID="6">5.00000000e-01 2.50000000e-01 0.00000000e+00</V>
            <V ID="7">5.00000000e-01 5.00000000e-01 0.00000000e+00</V>
            <V ID="8">-5.00000000e-01 1.37512224e-12 0.00000000e+00</V>
            <V ID="9">-1.66666667e-01 4.58346324e-13 0.00000000e+00</V>
            <V ID="10">1.66666667e-01 -4.58401835e-13 0.00000000e+00</V>
            <V ID="11">5.00000000e-01 -1.37512224e-12 0.00000000e+00</V>
            <V ID="12">-5.00000000e-01 -2.50000000e-01 0.00000000e+00</V>
            <V ID="13">-1.66666667e-01 -2.50000000e-01 0.00000000e+00</V>
            <V ID="14">1.66666667e-01 -2.50000000e-01 0.00000000e+00</V>
            <V ID="15">5.00000000e-01 -2.50000000e-01 0.00000000e+00</V>
            <V ID="16">-5.00000000e-01 -5.00000000e-01 0.00000000e+00</V>
            <V ID="17">-1.66666667e-01 -5.00000000e-01 0.00000000e+00</V>
            <V ID="18">1.66666667e-01 -5.00000000e-01 0.00000000e+00</V>
            <V ID="19">5.00000000e-01 -5.00000000e-01 0.00000000e+00</V>
</VERTEX>
```

Second, it defines edges base on vertex:

```
<EDGE>
            <E ID="0">    0  1   </E>
            <E ID="1">    1  2   </E>
            <E ID="2">    2  3   </E>
            <E ID="3">    3  0   </E>
            <E ID="4">    2  4   </E>
            <E ID="5">    4  5   </E>
            <E ID="6">    5  3   </E>
            <E ID="7">    4  6   </E>
            <E ID="8">    6  7   </E>
            <E ID="9">    7  5   </E>
            <E ID="10">    1  8   </E>
            <E ID="11">    8  9   </E>
            <E ID="12">    9  2   </E>
            <E ID="13">    9  10   </E>
            <E ID="14">   10  4   </E>
            <E ID="15">   10  11   </E>
            <E ID="16">   11  6   </E>
            <E ID="17">    8  12   </E>
            <E ID="18">   12  13   </E>
            <E ID="19">   13  9   </E>
            <E ID="20">   13  14   </E>
            <E ID="21">   14  10   </E>
            <E ID="22">   14  15   </E>
            <E ID="23">   15  11   </E>
            <E ID="24">   12  16   </E>
            <E ID="25">   16  17   </E>
            <E ID="26">   17  13   </E>
            <E ID="27">   17  18   </E>
            <E ID="28">   18  14   </E>
            <E ID="29">   18  19   </E>
            <E ID="30">   19  15   </E>
 </EDGE>
```

To understand this script, for example, consider the edge with "ID=30":

```
        <E ID="30">   19  15   </E>
```

the number "19" and "15" are the IDs of vertex. In other words, this means the two ending points of edge "30" is vertex "19" and vertex "15". The third part is elements:

```
<ELEMENT>
            <Q ID="0">    0     1     2     3 </Q>
            <Q ID="1">    2     4     5     6 </Q>
            <Q ID="2">    5     7     8     9 </Q>
            <Q ID="3">   10    11    12     1 </Q>
            <Q ID="4">   12    13    14     4 </Q>
            <Q ID="5">   14    15    16     7 </Q>
            <Q ID="6">   17    18    19    11 </Q>
            <Q ID="7">   19    20    21    13 </Q>
```

```
        <Q ID="8">    21    22    23    15 </Q>
        <Q ID="9">    24    25    26    18 </Q>
        <Q ID="10">    26    27    28    20 </Q>
        <Q ID="11">    28    29    30    22 </Q>
</ELEMENT>
```

Similar to the way we define edges, the four number in each line are IDs of edges. Since from Figure 4, we see each element is quadrilateralwe need four edges to form one element.

The last part is composite. Composite can help to define the boundary region:

```
<COMPOSITE>
        <C ID="1"> E[0,10,17,24] </C><!-- inflow -->
        <C ID="2"> E[30,23,16,8] </C><!-- outflow -->
        <C ID="3"> E[25,27,29,9,6,3] </C><!-- walls -->
        <C ID="4"> Q[0-11] </C><!-- Domain -->
</COMPOSITE>
```

The basic blocks of composite is edge. For example, edges "0,10,17,24" corresponding the leftmost boundary in Figure 4 define the inflow region and edges "30,23,16,8" corresponding to the rightmost boundary in Figure 4 define the outflow region so that people know in this simulation, the fluid flow into the computational domain from the left and flow out from the right. The walls in this case are consist with uppermost and lowest boundary which are edges "25,27,29,9,6,3". The composite with "ID=4" is the whole computational domain which including all the elements(from element 1 to element 11).

2.6.2 Nektar++ condition file

Having the geometry and mesh of computaional domain is not enough for us to do a simulation, we still need to define how we do the simulation and this is why we need Nektar++ condition file. The condition file is also in .xml format. It specifies every aspects we need to know for a simulation.

In this simulation, we use 7th-order polynomial expansions using the modified Legendre basis:

```
<EXPANSIONS>
    <E COMPOSITE="C[4]" NUMMODES="8" FIELDS="u,v,p" TYPE="MODIFIED" />
```

```
</EXPANSIONS>
```

Recall the composite C[4] is the whole domain and this means the 7th-order polynomial expansions are applied in very element in the computational domain. Since we use 7th-order polynomial expansions, the number of modes is 8(we have NUMMODES=8). The "FIELDS" means what variable we want to solve, which will be introduced later. The TYPE="MODIFIED" means we use modified Legendre basis function.

The solver information in condition file is defined as:

```
<SOLVERINFO>
        <I PROPERTY="SolverType" VALUE="VelocityCorrectionScheme" />
        <I PROPERTY="EQTYPE" VALUE="UnsteadyNavierStokes" />
        <I PROPERTY="AdvectionForm" VALUE="Convective" />
        <I PROPERTY="Projection" VALUE="Galerkin" />
        <I PROPERTY="TimeIntegrationMethod" VALUE="IMEXOrder2" />
</SOLVERINFO>
```

By defining "SolverType", we use velocity correction scheme to solve the governing equation. The "EQTYPE" means equation type, which is unsteady Navier-Stokes eqaution. The "Projection" specifies that we use Galerkin method for the weak form of equation.

The parameters are defined as:

```
<PARAMETERS>
        <P> TimeStep       = 0.001    </P>
        <P> NumSteps       = 10000     </P>
        <P> IO_CheckSteps  = 1000      </P>
        <P> IO_InfoSteps   = 1        </P>
        <P> Re             = 40       </P>
        <P> Kinvis         = 1/Re     </P>
        <P> LAMBDA         = 0.5*Re-sqrt(0.25*Re*Re+4*PI*PI) </P>
</PARAMETERS>
```

We set the time step be 0.001 and total number of time steps be 10000 so we perform totally 10 dimensionl time unit.

The variables are defined as:

```
<VARIABLES>
        <V ID="0"> u </V>
        <V ID="1"> v </V>
        <V ID="2"> p </V>
</VARIABLES>
```

These are the three variables we need to solve. The u, v are the quantity of velocity in X-axis and Y-axis direction respectively and p is pressure.

The boundary regions are defined as:

```
<BOUNDARYREGIONS>
        <B ID="0"> C[1] </B>
        <B ID="1"> C[2] </B>
        <B ID="2"> C[3] </B>
</BOUNDARYREGIONS>
```

From this we can see that the boundary regions are actually defined from Composite from the mesh file. For example, the boundary region with ID=0 is C[1] which is the composite with ID=1(inflow).

After defining the boundary region, we can define boundary conditions now:

```
<BOUNDARYCONDITIONS>
        <REGION REF="0">
            <D VAR="u" VALUE="1-exp(LAMBDA*x)*cos(2*PI*y)" />
            <D VAR="v" VALUE="(LAMBDA/2/PI)*exp(LAMBDA*x)*sin(2*PI*y)" />
            <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
        </REGION>
        <REGION REF="1">
            <D VAR="u" VALUE="1-exp(LAMBDA*x)*cos(2*PI*y)" />
            <D VAR="v" VALUE="(LAMBDA/2/PI)*exp(LAMBDA*x)*sin(2*PI*y)" />
            <D VAR="p" VALUE="0.5*(1-exp(2*LAMBDA*x))" />
        </REGION>
        <REGION REF="2">
            <N VAR="u" VALUE="0" />
            <D VAR="v" VALUE="0" />
            <N VAR="p" VALUE="0" />
        </REGION>
</BOUNDARYCONDITIONS>
```

The Region with REF=X refers to boundary region with ID=X. The letter D and N refer to Dirichlet and Neumann boundary condition. For example, for boundary region with ID=0, the variable u, v are specified with Dirichlet boundary condition and variable p is specified with Neumann boundary condition.

In Nektar++ condition file, we can define our own function. Here we define the initial condition:

```
<FUNCTION NAME="InitialConditions">
            <E VAR="u" VALUE="0" />
            <E VAR="v" VALUE="0" />
            <E VAR="p" VALUE="0" />
</FUNCTION>
```

We specifies the three variables are all zero in initial conditions.

We can also define the exact solution function:

```
<FUNCTION NAME="ExactSolution">
            <E VAR="u" VALUE="(1-exp(LAMBDA*x)*cos(2*PI*y))" />
            <E VAR="v" VALUE="(LAMBDA/2/PI)*exp(LAMBDA*x)*sin(2*PI*y)" />
            <E VAR="p" VALUE="0.5*(1-exp(2*LAMBDA*x))" />
</FUNCTION>
```

With this solution, Nektar++ will compute the relative error with simulation results. We can also perform convergence analysis using those errors, which will be discussed later.

The information above introduce all the blocks of Nektar++ condition file. The next step is pass the mesh file and condition file into Nektar++ solver for numerical simulation.

2.6.3 Numerical simulation and result

To perform the numerical simulation, we can simply type the following command in command line:

```
IncNavierStokesSolver mesh.xml condition.xml
```

The executable file "IncNavierStokesSolver" locates at:

```
nektar++/build/solvers/IncNavierStokesSolver/IncNavierStokesSolver
```

users should add path when typing this command.

After the simulation, Nektar++ will output the final result which is in .fld format. In this case, in addition to .fld file, Nektar++ will also output .chk file which represent the

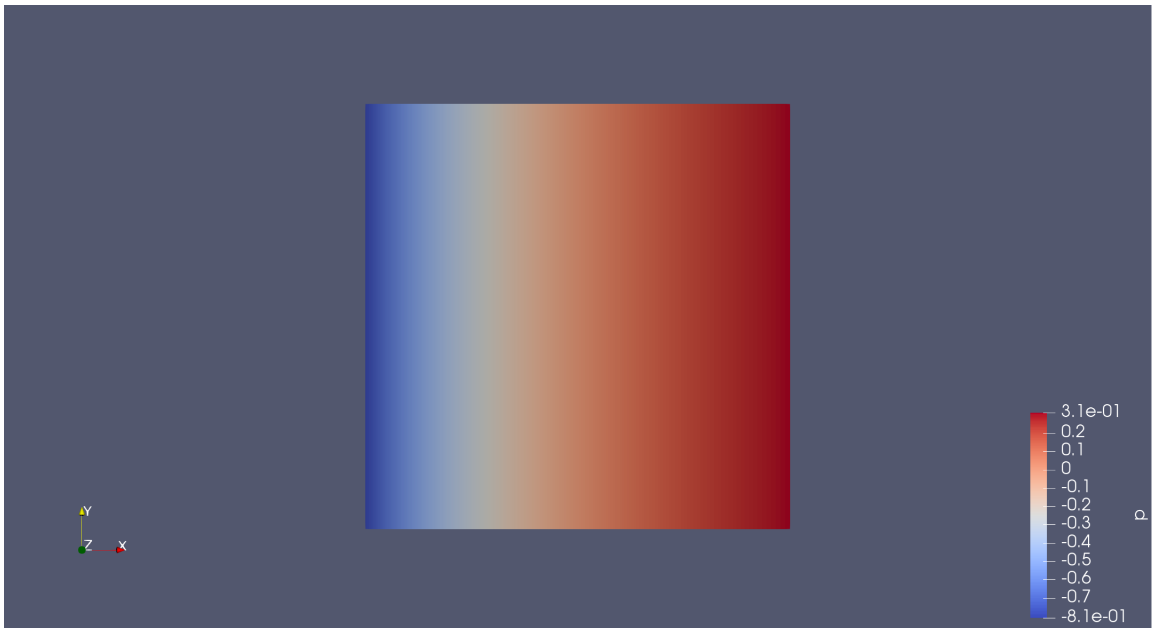


Figure 5: Pressure.

intermediate results due to IO_CheckSteps in condition file. To visualize the result in Paraview, we can use Nektar++ post-processing utility by typing the command:

```
FieldConvert result.fld mesh.xml result.vtu
```

The executable file FieldConvert will get the input result.fld and mesh.xml and output result.vtu which is a format can be opened in Paraview. The location of the executable file FieldConvert is at:

```
nektar++/build/dist/bin/FieldConvert
```

The visualization of the results in Paraview are in Figure 5,6 and 7

2.6.4 H-type convergence experiments

In this section we will plot relative errors to show that denser mesh will help improve results. The relative error is L2 error defined as:

$$\|\varepsilon\|_2 = \Big[\int_\Omega |result - real(solution)|^2 d\Omega\Big]^{\frac{1}{2}} \tag{56}$$

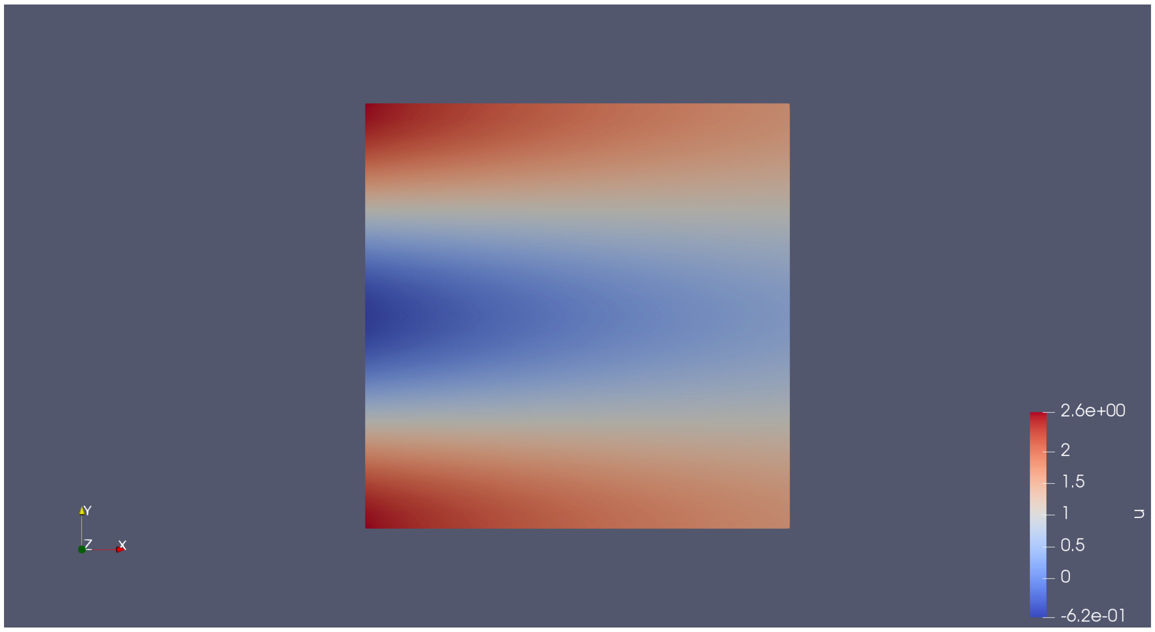


Figure 6: Size of velocity in X-axis direction.

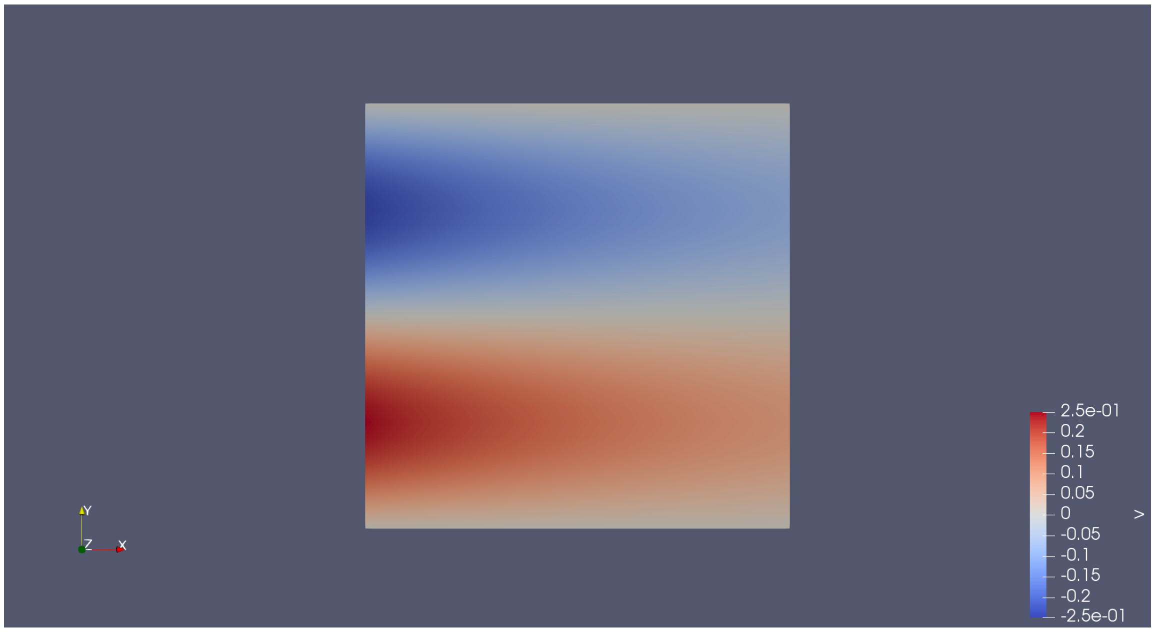


Figure 7: Size of velocity in Y-axis direction.

Here we do five experiments and record errors of the results. For each experiment, we fix the order of polynomial expansions be 2(number of modes is 3) and we change the density of the mesh.Figure 8 shows how the density of mesh change. Using these mesh for simulation, we plot relative errors for the size velocity in X-axis and Y-axis directions and pressure. The Figure 9 shows the relative error is going down with denser mesh. In order to show the decrease of error, we take logarithm of the error in Figure 9.

2.6.5 P-type convergence

In this section, we fix the refinement of mesh and change the orders of polynomial expansions and see the convergence of the result. We use polynomial expansions with order 1,3,5,7,9 and the fourth mesh in section 2.6.4, the plot of convergence of errors(logarithm of errors) is shown in Figure 10 where the X-axis represent the order of polynomial expansions. In fact, the p-type convergence is much faster than h-type convergence. For example, the five errors for variable u in section 2.6.4 are:[0.00972794, 0.00141868, 0.000445087, 3.35383e-5, 2.51558e-6], while the five errors for variable u in p-type convergence are:[0.20766, 0.00184112, 8.90652e-6, 2.41333e-8, 4.14239e-11].

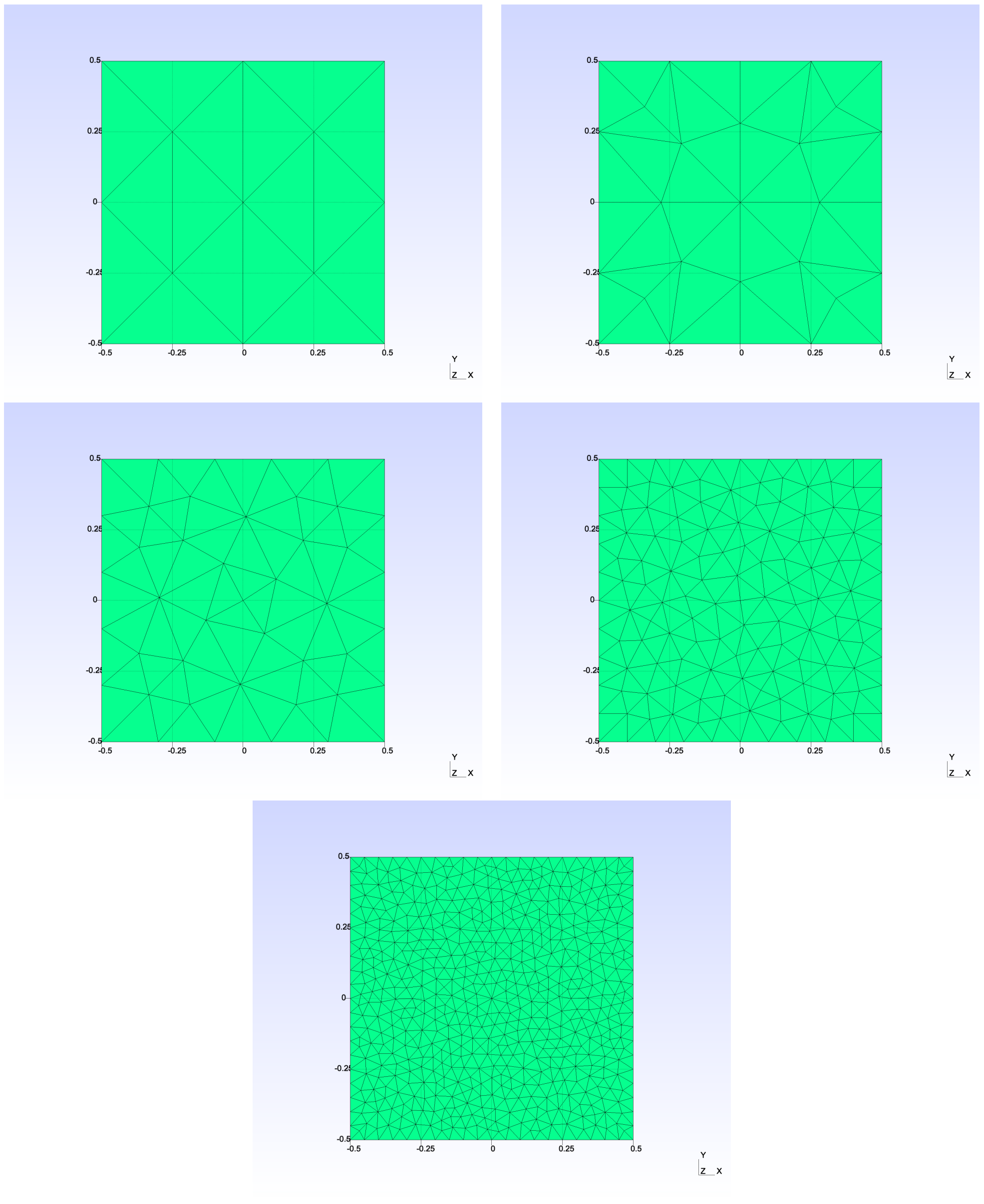


Figure 8: Five meshes with increasing density.

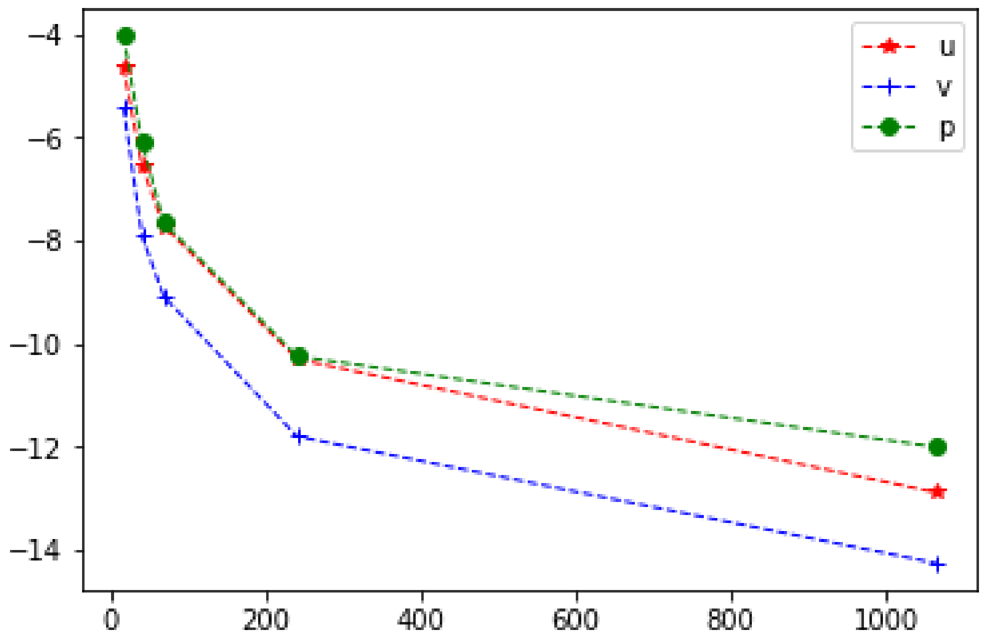


Figure 9: This plot shows convergence of u,v,p with denser mesh. The X-axis represents number of elements in mesh. The Y-axis is logarithm relative error.

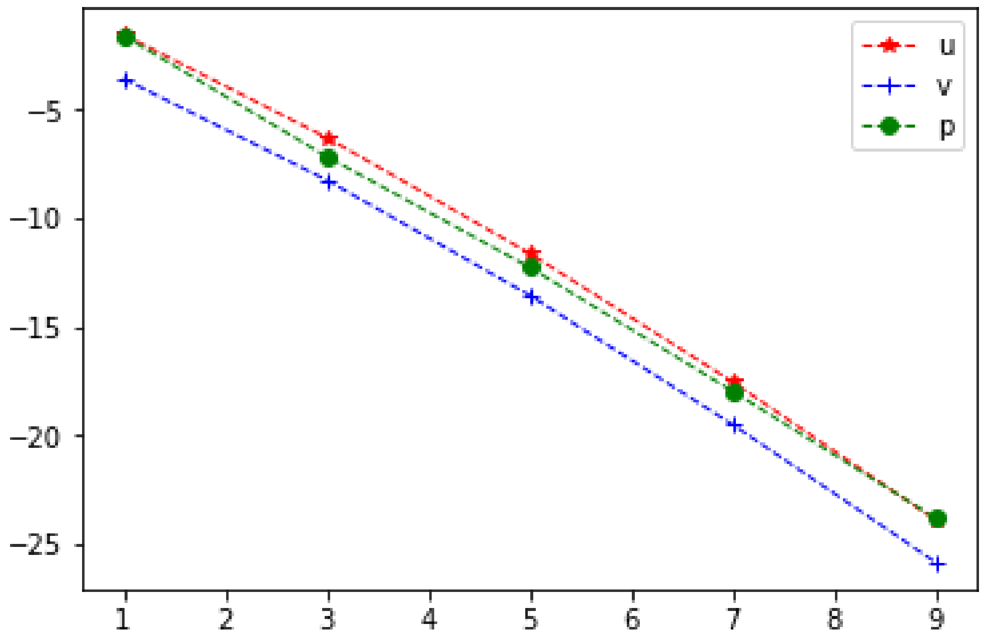


Figure 10: P-type convergence.

3. Realization

In this part we will show concrete numerical experiments in blood flow simulation using Nektar++. By considering the effect of heart, we model the blood flow as pulsatile flow. In order to do so, we need to specify Womersley boundary condition at inlet of the vessel. Also, in order to deal with boundary condition of pressure at the outlet of the vessel, we will employ Windkessel model which uses idea from circuit to model the rest of the arterial system. Womersley boundary condition and Windkessel model will be introduced in the following part. After understanding those, we can see how to set up the numerical experiments.

3.1. Womersley boundary condition

The problem of Womersley boundary condition is also called Womersley flow problem. The Womersley flow problem was first introduced in paper.[24] The goal of Womersley boundary condition is to specify the velocity of blood flow at the inlet of the vessel by solving incompressible Navier-Stocks equation. However, this time the equation can be easier to solve. In the setting of,[24] the pressure gradient is known. By decomposing the pressure gradient in Fourier series, we obtain:

$$\frac{\partial p}{\partial x}(t) = \sum_{n=0}^{N} P_n' e^{in\omega t} \tag{57}$$

where the i is imaginary number, ω is angular frequency of the first harmonic and P_n' are the amplitudes of each harmonic n. People may notice the pressure gradient $\frac{\partial p}{\partial x}(t)$ is just one dimension(x direction). That's because the computational domain for Womersley flow is only the inlet of vessel(that's why it is also called boundary condition), which in usual case is a circle and we only care about the pressure gradient in direction of the one perpendicular to inlet of the vessel. So here the x direction should be perpendicular to the plane of vessel's inlet but may not be the x-axis.

To derive the velocity, we also decompose it into Fourier series:

$$u(r,t) = \sum_{n=0}^{N} U_n e^{in\omega t} \tag{58}$$

and use each known harmonic of pressure gradient to solve the corresponding harmonic of velocity(more detailed derivation can be found from.[24,29] In equation (58), the velocity profile is represented using two variables: r and t, where r is the radial coordinate on the inlet of the vessel(since we assume the inlet is a circle, we can use radial coordinate) and t is time. So in different time, we impose different velocity on the inlet of the vessel to simulate the effect of pulsatile flow and this is an intuition to understand mechanism of Womersley boundary condition. The solution of amplitudes in each harmonic of velocity is:

$$U_0(r) = -\frac{P_0'}{4\nu}(R^2 - r^2) \tag{59}$$

$$U_n(r) = \frac{iP_n'}{n\omega}\Big[1 - \frac{J_0(\Lambda_n \frac{r}{R})}{J_0(\Lambda_n)}\Big] \qquad n = 1, ..., N \tag{60}$$

Combining with equation (58), the solution of velocity is:

$$u(r,t) = -\frac{P_0'}{4\nu}(R^2 - r^2) + Re\{\frac{iP_n'}{n\omega}\Big[1 - \frac{J_0(\Lambda_n \frac{r}{R})}{J_0(\Lambda_n)}\Big]e^{in\omega t}\} \tag{61}$$

where R is radius of vessel's inlet,ν is kinematic viscosity,$J_0()$ is the Bessel function of first kind and order zero and $\Lambda_n = \alpha n^{\frac{1}{2}} i^{\frac{3}{2}}$ where α is womersley number defined as $\alpha = R(\frac{\omega}{\nu})^{\frac{1}{2}}$

3.2. Windkessel model

Since the limitation of computing resources, people usually do the blood flow simulation just on a truncated aorta. In order to represent the rest part of arterial network, we can set an effective boundary condition by using Windkessel model. Windkessel model borrows idea from circuit and use electronic component like capacitor and resistance to model arterial system. The Figure 11 show us a illustration of how Windkessel model works. In the following sections, we will introduce two-element and three-element Windkessel model.

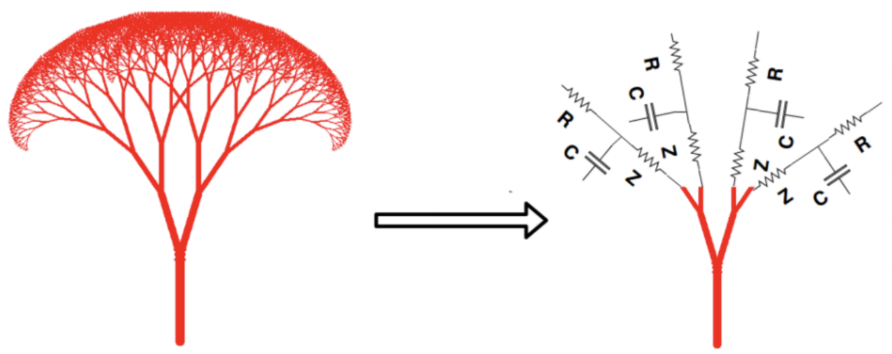


Figure 11: Windkessel model use electronic component to model rest part of arterial network.

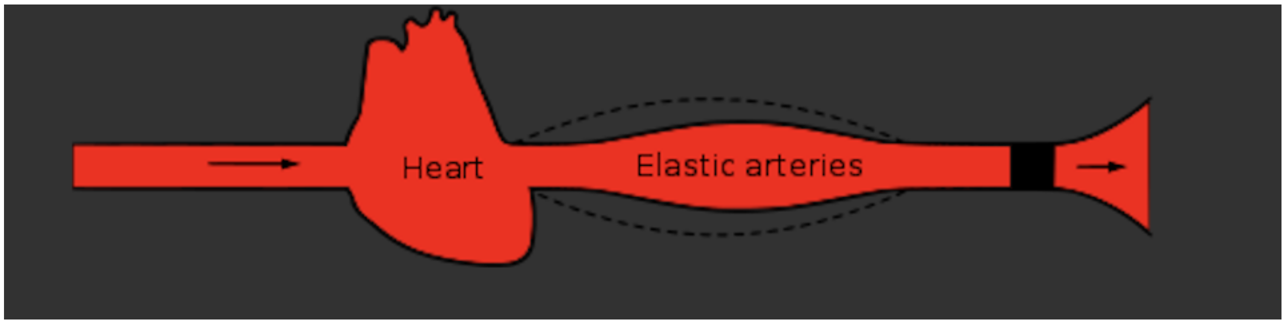


Figure 12: Windkessel model analogy.[30]

3.2.1 Two-element Windkessel model

The two-element Windkessel model involves two electronic components: one capacitor and one resistance. During the systole, blood will flow into elastic arteries as shown in Figure 12. However, due to the peripheral resistance, only part of blood can leave so there is a net storage of blood in elastic arteries, which discharge during diastole[wiki]. Given this mechanism, we can model the elastic arteries as capacitor(charge and discharge when heart is pumping). The Figure 13 also shows that the circuit analogy of two-element windkessel model.

The governing equation of two-element Windkessel model is the following:

$$I(t) = \frac{P(t)}{R} + C\frac{dP(t)}{dt} \tag{62}$$

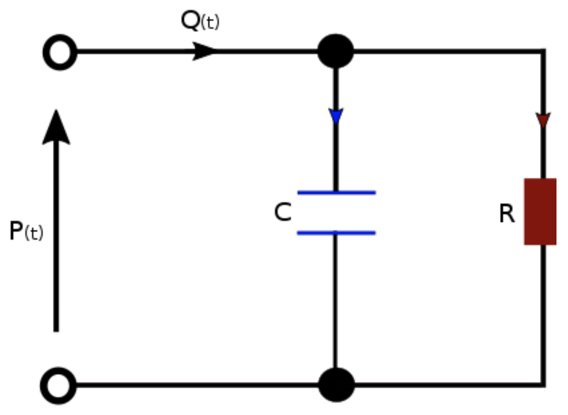


Figure 13: Circuit analogy of two-element Windkessel model. P(t) represents the pressure, Q(t) is the flow, C is capacitor and R is resistance.[30]

where $I(t)$ is volumetric inflow due to the heart pump, $P(t)$ is pressure, C is the ratio of volume to pressure and R is resistance relating outflow to fluid pressure. More detailed information on derivation of this equation is beyond our scope, people can refer to [wiki,...]. Solving the equation 62, we can get the pressure and set it to the boundary condition(change with time) of outlet of the vessel.

3.2.2 Three-element Windkessel model

Comparing with two-element Windkessel model, the three-element Windkessel model not only consider the peripheral resistance but also the resistance of large elastic arteries. Figure 14 shows the circuit analogy of three-element Windkessel model where Z_c represent the resistance of elastic arteries.

The governing equation of three-element Windkessel model is :

$$(1 + \frac{R_1}{R_2} I(t) + CR_1 \frac{dI(t)}{dt} = \frac{P(t)}{R_2} + C\frac{dP(t)}{dt} \tag{63}$$

where R_1 is resistance of large elastic arteries, R_2 is peripheral resistance. $I(t)$, $P(t)$ and C keep the same meaning with equation 62.

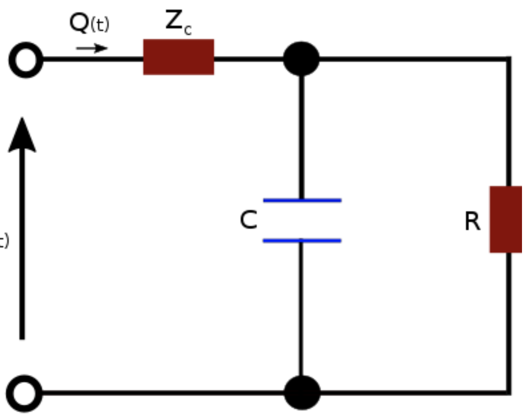


Figure 14: Cuircuit analogy of three-element Windkessel model.[30]

3.3. Simulation on cylinder with parabolic velocity

In this section, we will show an example on how to do blood flow simulation on 3D cylinder, which can be a foundation for simulation on geometry of vessels.

3.3.1 Geometry and mesh of computational domain

In this section, we assume that the inflow is parabolic(not a pulsatile flow). The geometry and mesh of cylinder is shown in Figure 15. The height of this cylinder is 2 and the radius of the inlet is 1 in Gmsh. The (x,y,z) coordinate of inlet center is (0,0,0) and direction of inflow is (0,0,1) which is also the normal direction of inlet pointing inside of the cylinder.

Again, following the same procedure as shown in section 2.6.1, we use Nektar++ pre-processing utility ”Nekmesh” to convert mesh.msh file into mesh.xml file. Since the .xml is too big, we don’t show it here.

3.3.2 Nektar++ condition file

After setting up the computational domain, the next step is preparing Nektar++ condition file. The whole file is as following:

```
<?xml version="1.0" encoding="utf-8"?>
<NEKTAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

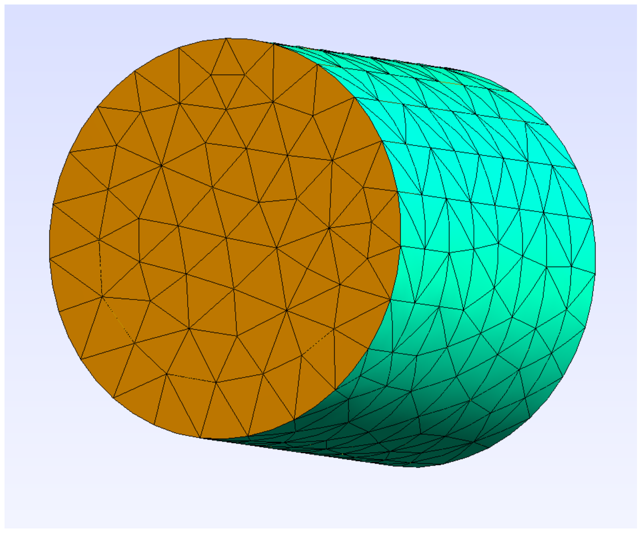


Figure 15: 3D cylinder in Gmsh.

```
 xsi:noNamespaceSchemaLocation="http://www.nektar.info/schema/nektar.xsd">
<EXPANSIONS>
    <E COMPOSITE="C[4]" NUMMODES="5" FIELDS="u,v,w,p" TYPE="MODIFIED" />
</EXPANSIONS>
<CONDITIONS>
    <SOLVERINFO>
        <I PROPERTY="SolverType" VALUE="VelocityCorrectionScheme" />
        <I PROPERTY="EQTYPE" VALUE="UnsteadyNavierStokes" />
        <I PROPERTY="AdvectionForm" VALUE="Convective" />
        <I PROPERTY="Projection" VALUE="Galerkin" />
        <I PROPERTY="TimeIntegrationMethod" VALUE="IMEXOrder1" />
    </SOLVERINFO>

    <PARAMETERS>
        <P> TimeStep = 0.005          </P>
        <P> NumSteps = 3400           </P>
        <P> IO_CheckSteps = 50        </P>
        <P> IO_InfoSteps = 50          </P>
        <P> Kinvis = 0.00167      </P>
    </PARAMETERS>

    <VARIABLES>
        <V ID="0"> u </V>
        <V ID="1"> v </V>
        <V ID="2"> w </V>
        <V ID="3"> p </V>
    </VARIABLES>

    <BOUNDARYREGIONS>
        <B ID="0"> C[1] </B>
        <B ID="1"> C[2] </B>
        <B ID="2"> C[3] </B>
    </BOUNDARYREGIONS>
```

```
        <BOUNDARYCONDITIONS>

            <REGION REF="0">
                <D VAR="u"    VALUE="0" />
                <D VAR="v"    VALUE="0" />
                <D VAR="w"    VALUE="1-x*x-y*y" />
                <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
            </REGION>

            <REGION REF="1">
                <N VAR="u" VALUE="0" />
                <N VAR="v" VALUE="0" />
                <N VAR="w" VALUE="0" />
                <D VAR="p" VALUE="0" />
            </REGION>

            <REGION REF="2">
                <D VAR="u" VALUE="0" />
                <D VAR="v" VALUE="0" />
                <D VAR="w" VALUE="0" />
                <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
            </REGION>

        </BOUNDARYCONDITIONS>

        <FUNCTION NAME="InitialConditions">
            <E VAR="u" VALUE="0" />
            <E VAR="v" VALUE="0" />
            <E VAR="w" VALUE="0" />
            <E VAR="p" VALUE="0" />
        </FUNCTION>
    </CONDITIONS>
</NEKTAR>
```

The condition file here has many similarities with the one we have introduced in section 2.6.2. Here we only explain some special information. In "VARIABLES" part, the u, v, w(scalers) are the size of velocity in direction of X-axis, Y-axis and Z-axis, respectively. The p is pressure. They are four variables that we want to solve from this simulation. In "BOUNDARYREGIONS" part: c[1](ID=0) is the inlet, c[2](ID=1) is the outlet and c[3](ID=2) is the wall. In "¡REGION REF="0"¿" of "BOUNDARYCONDITIONS" part, we set the size of velocity(w) in direction of Z-axis be $1 - x^2 - y^2$ which is a parabolic function open towards minus Z-axis. For other direction, the size of velocity are all zeros. This sets the parabolic velocity boundary condition.

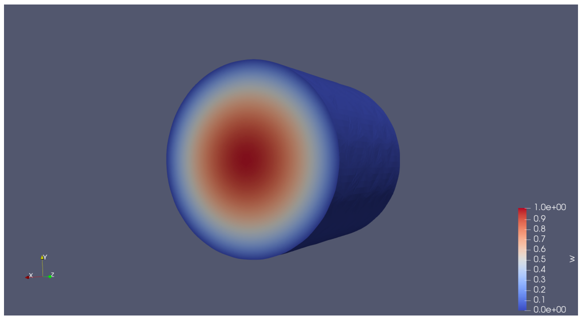


Figure 16: Visualization of velocity of inlet in Z-axis direction.

3.3.3 Simulation result

By pass the mesh.xml and condition.xml file into Nektar++ solver as shown in section 2.6.3, we can do the simulation and get a bunch of .chk files and one final .fld file as results. Using the post-processing utility as shown in section 2.6.3, we transform all the .chk and .fld files into .vtu files which can be visualized in Paraview.

Figure 16 visualize w in the inlet and this will keep the same through out the whole simulation because it is fixed as boundary condition. This initial velocity of flow will push fluid into the cylinder and finally flow out of the cylinder. The change of velocity of fluid of outlet is shown in Figure 17. We can see that at first there is nearly no outflow at the outlet but later the "red region" appears and gets bigger.

The u, v, w from simulation are scalers but not vectors. In order to visualize the velocity but not the size of velocity, we need to compute $u\mathbf{i} + v\mathbf{j} + w\mathbf{k}$ to get the real velocity which is a vector. To do so, we can click the "calculator" button in Paraview and type $u * iHat + v * jHat + w * kHat$ as shown in Figure 18. After calculating the vector, we can just type the "Glyph" button and set like the way shown in Figure 19(The scale factor can be set as users' wish or just click the "reset" button). Figure shows how the velocity looks like at the inlet and Figure how they evolve with time at the outlet.

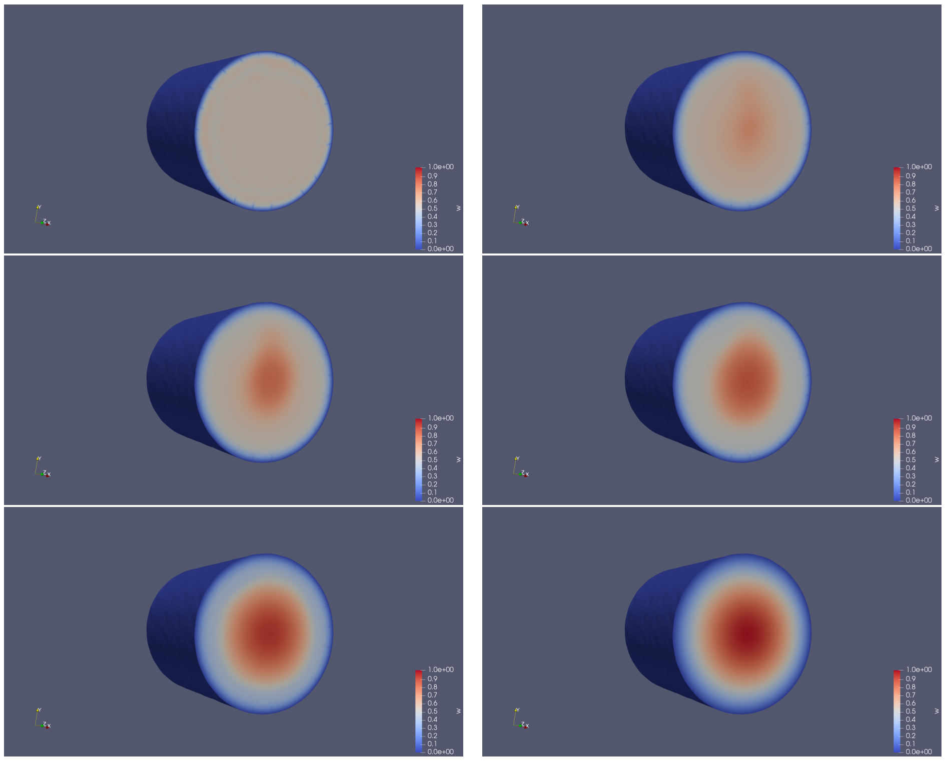
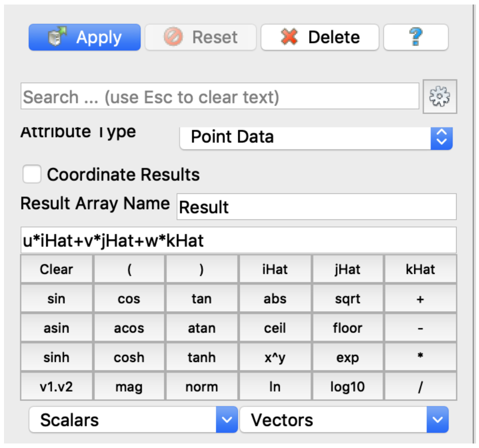


Figure 17: Evolution of velocity at outlet in direction of Z-axis.

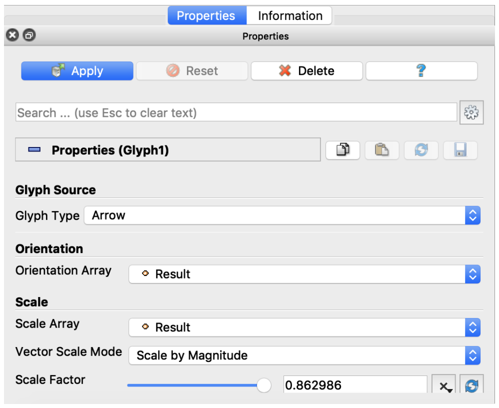

Attribute Type: Point Data

Coordinate Results: [ ]

Result Array Name: Result

```
u*iHat+v*jHat+w*kHat
```

Figure 18: Calculator in Paraview.


Properties | Information

Properties

Apply | Reset | Delete | ?

Search ... (use Esc to clear text)

Properties (Glyph1)

Glyph Source

Glyph Type: Arrow

Orientation

Orientation Array: Result

Scale

Scale Array: Result

Vector Scale Mode: Scale by Magnitude

Scale Factor: 0.862986

Figure 19: Setting parameters with Glyph filter of Paraview.

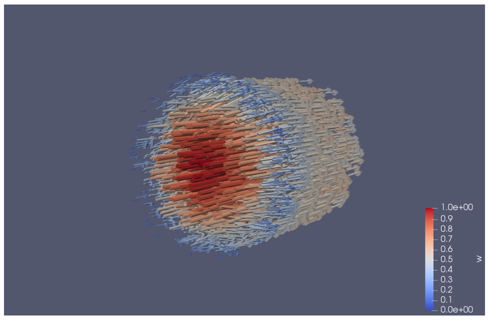


Figure 20: Visualization of velocity at inlet.

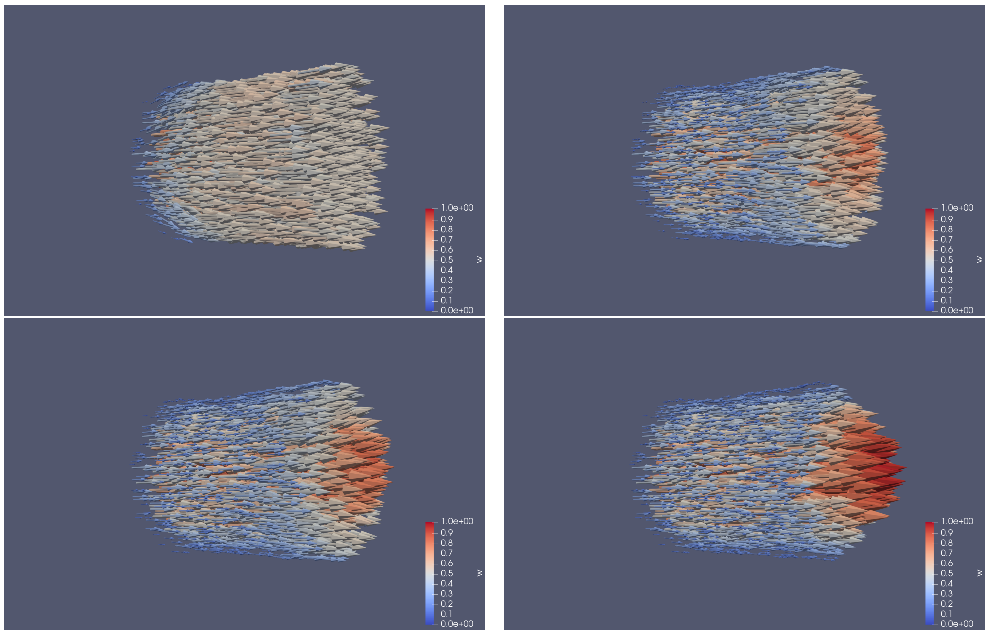


Figure 21: Evolution of velocity at outlet.

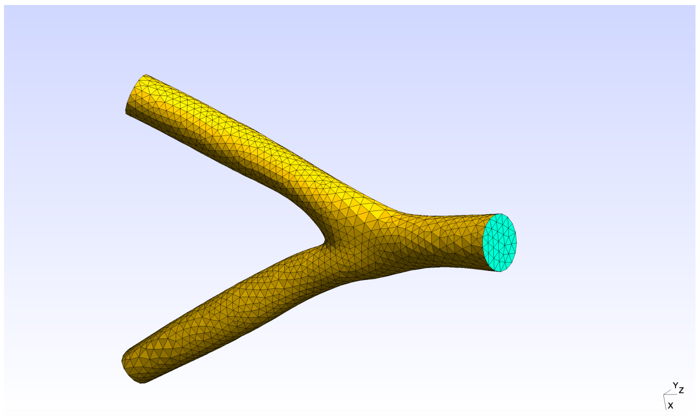


Figure 22: 3D vessel model in Gmsh.

3.4. Simulation on vessel model with Womersley boundary condition

Having a basic understanding of section 3.3, we can go further by taking two changes:(i)change the computational domain to vessel model. (ii) change the parabolic velocity boundary condition to Womersley boundary condition. We will see how to deal with these two changes of simulation in this section.

3.4.1 Geometry and mesh of computational domain

The computational domain this time is a small vessel model, which has one inlet and two outlet and is shown in Figure 22.

3.4.2 Setting the Womersley boundary condition

In section 3.1, we introduced basic knowledge on Womersley boundary condition. The intuition of Womersley boundary condition is using the known pressure gradient to solve unknown velocity at the inlet. However, in practice, people may already know the velocity or flow rate getting from sensor equipment. In this case, we don't need to do any computing because we already have what we want. In Nektar++, the way we represent the known velocity is by Fourier coefficients. In addition to this, we need to let Nektar++ know where the boundary is, how big the boundary is, what is the period of the pulsatile flow and which direction will the fluid flow. The whole Nektar++ file Womersley.xml setting Womersley boundary condition shows below:

```
<NEKTAR>

    <WOMERSLEYBC>
        <WOMPARAMS>
            <W PROPERTY="Radius" VALUE="0.545" />
            <W PROPERTY="Period" VALUE="64.0362" />
            <W PROPERTY="axisnormal" VALUE="0.01352417,0.08791284,-0.99603636" />
            <W PROPERTY="axispoint" VALUE="7.499648284912109,2.308580017089844,20.80792846679688" />
        </WOMPARAMS>

        <FOURIERCOEFFS>
           <F ID=0> 0.7175784462781089, -0.0 </F>
           <F ID=1> -0.04506297092144961, 0.301589400703441 </F>
           <F ID=2> -0.12619053984027664, -0.20612334538907198 </F>
           <F ID=3> 0.052919990545354074, -0.044518075477739616 </F>
           <F ID=4> 0.03645203646647879, 0.016945891821456532 </F>
           <F ID=5> -0.01997436040436404, 0.016798376246354704 </F>
           <F ID=6> -0.0035240694840311315, -0.0006192970034698905 </F>
           <F ID=7> -0.00035103905617903466, -0.0001758973968067276 </F>
           <F ID=8> -0.00025053437652451015, -0.0001844555162757482 </F>
           <F ID=9> -0.0001753319421690523, -0.00021009265824309655 </F>
           <F ID=10> -0.00013482250717433635, -0.00021148586300272343 </F>
           <F ID=11> -0.00011126578659507418, -0.00020479897415254025 </F>
           <F ID=12> -9.620628051327472e-05, -0.00019524194582150282 </F>
           <F ID=13> -8.590388615896073e-05, -0.00018494019625953675 </F>
           <F ID=14> -7.849366917098696e-05, -0.00017479111610011285 </F>
           <F ID=15> -7.295565197942886e-05, -0.0001651640490917334 </F>
           <F ID=16> -6.869033624927881e-05, -0.0001561910101342643 </F>
           <F ID=17> -6.53246296687906e-05, -0.00014789537407356546 </F>
           <F ID=18> -6.261534348662868e-05, -0.0001402515462979492 </F>
           <F ID=19> -6.039780552001897e-05, -0.00013321348010101016 </F>
        </FOURIERCOEFFS>
    </WOMERSLEYBC>

</NEKTAR>
```

In the "WOMPARAMA" part, we set radius of the inlet as 0.545. This radius is read from Gmsh. Users need to find the center of the inlet(sometimes approximately) and try to measure the radius if the position of the inlet is not in a very standard place(like the inlet centers at (0,0,0)). The period of pulsatile flow here is 64.0362. This number is non-dimensional. How can we understand this number? Usually, in simulation, in order to align different unit, people tend to transform each quantity into non-dimensional quantity, that is , get rid of units. To do so, we need benchmark. For length, we can choose a characteristic length and let the real length divided by this characteristic length to get non-dimensional length. For example, if a vessel has length 2cm and we choose 1cm as characteristic length, then the non-dimensional length of this vessel is 2 and we can say that the length of this vessel is twice as the benchmark. Using non-dimensional quantity can let us ignore the real quantity but emphasis the relationship with benchmark(transform the 'absolute' into 'relative'). After choosing a characteristic length L_c, we can choose a characteristic velocity V_c and $T_c = L_c/V_c$ should be characteristic time. Dividing the real period by this characteristic time will give us characteristic period which is 64.0362 in this case.

Theoretically, people can choose characteristic quantity as they wish. However, in this case, the characteristic length has to be computed. The vessel model is build from MRI images and in those images, we can measure the real radius of inlet of vessel. However, when we put this model in Gmsh, Gmsh will assign coordinates to each points, which means Gmsh will also define radius of inlet(0.545 in our case). This radius may different with the real radius in MRI image and for now we don't know how Gmsh assign those coordinates. We can view the radius of inlet in Gmsh as non-dimensional length and derive what is the characteristic length by dividing real length by length in Gmsh.

To set direction of the flow, we can see 'axisnormal'. This is a vector that points the direction of the flow and should be computed from vessel in Gmsh. The 'axispoint' denotes the center of the inlet which is a coordinate from Gmsh.

3.4.3 Nektar++ condition file

The whole file of condition file is as follows:

```
<NEKTAR xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.nektar.info/schema/nektar.xsd">
    <EXPANSIONS>
        <E COMPOSITE="C[5]" NUMMODES="2" FIELDS="u,v,w,p" TYPE="MODIFIED" />
    </EXPANSIONS>

    <CONDITIONS>
        <SOLVERINFO>
            <I PROPERTY="SolverType" VALUE="VelocityCorrectionScheme" />
            <I PROPERTY="EQTYPE" VALUE="UnsteadyNavierStokes" />
            <I PROPERTY="AdvectionForm" VALUE="Convective" />
            <I PROPERTY="Projection" VALUE="Galerkin" />
            <I PROPERTY="TimeIntegrationMethod" VALUE="IMEXOrder1" />
        </SOLVERINFO>

        <PARAMETERS>
            <P> TimeStep = 0.005          </P>
            <P> NumSteps = 12820            </P>
            <P> IO_CheckSteps = 200         </P>
            <P> IO_InfoSteps = 50           </P>
            <P> Kinvis =   1.0/2428.82         </P>
        </PARAMETERS>

        <VARIABLES>
            <V ID="0"> u </V>
            <V ID="1"> v </V>
            <V ID="2"> w </V>
            <V ID="3"> p </V>
        </VARIABLES>

        <BOUNDARYREGIONS>
            <B ID="0"> C[1] </B>
            <B ID="1"> C[2] </B>
            <B ID="2"> C[3] </B>
            <B ID="3"> C[4] </B>

        </BOUNDARYREGIONS>

        <BOUNDARYCONDITIONS>
            <REGION REF="0">
                <N VAR="u" VALUE="0" />
                <N VAR="v" VALUE="0" />
                <N VAR="w" VALUE="0" />
                <D VAR="p" VALUE="0" />
            </REGION>
                  <REGION REF="1">
                <N VAR="u" VALUE="0" />
                <N VAR="v" VALUE="0" />
                <N VAR="w" VALUE="0" />
                <D VAR="p" VALUE="0" />
            </REGION>
            <REGION REF="2">
                <D VAR="u"    USERDEFINEDTYPE="Womersley:WomParams.xml"   VALUE="0" />
                <D VAR="v"    USERDEFINEDTYPE="Womersley:WomParams.xml"   VALUE="0" />
                <D VAR="w"    USERDEFINEDTYPE="Womersley:WomParams.xml"   VALUE="0" />
                <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
            </REGION>
            <REGION REF="3">
                <D VAR="u" VALUE="0" />
                <D VAR="v" VALUE="0" />
                <D VAR="w" VALUE="0" />
                <N VAR="p" USERDEFINEDTYPE="H" VALUE="0" />
```

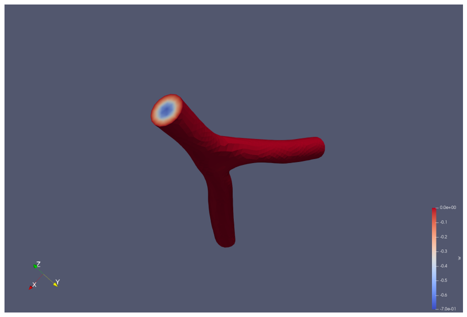


Figure 23: Visualization for 'w' variable at a certain time step when using Womersley boundary condition.

```
            </REGION>
        </BOUNDARYCONDITIONS>

        <FUNCTION NAME="InitialConditions">
            <E VAR="u" VALUE="0" />
            <E VAR="v" VALUE="0" />
            <E VAR="w" VALUE="0" />
            <E VAR="p" VALUE="0" />
        </FUNCTION>
    </CONDITIONS>
</NEKTAR>
```

The most special thing in this file is "REGION REF=2" in "BOUNDARYCONDITIONS" part. Here the "REGION REF=2" denotes the inlet and this is how we call Womersley boundary condition from Wompara.xml defined in section 3.4.2. By doing so, Nektar++ will assign size of velocity to u, v, w automatically.

3.4.4 Simulation result

To give people an idea on what may the effect of Womersley boundary condition look like, the Figure 23 shows the size of velocity in direction of Z-axis. We can see in the center of the inlet, the color is blue. This is because the flow will go into nearly the direction of minus Z-axis(see WomPara.xml in section 3.4.2 'axisnormal') so the value should be minus.

To see the evolution of simulation result in time, we visualize the velocity of the flow in Figure 24. Again, the blue color denotes bigger size of velocity than red color because of the direction of the flow. From Figure 24 we can see that the size of velocity first becomes

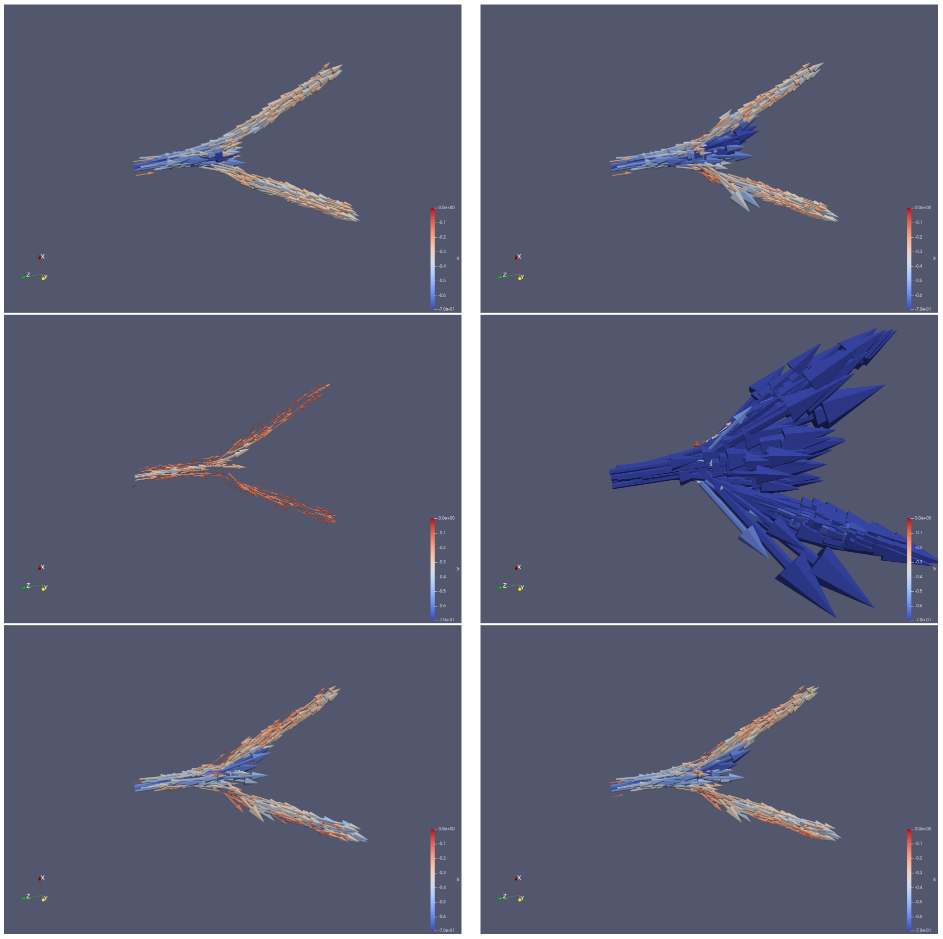


Figure 24: Evolution of velocity on vessel model with Womersley boundary condition.

bigger, then smaller, then bigger, then smaller and finally bigger. However, if we look at Figure 26, we will find this trend is not right. However, if we look the velocity wave from 'right' to 'left', we will find the evolution of velocity in Figure 24 is right. In fact, Nektar++ will process the velocity in the way from 'right' to 'left', which means the wave of velocity will be flipped in time. If we plot the size of velocity at a point of inlet basing on the simulation results(.chk files), we obtain Figure 25. The velocity indeed flips in time.

There is also an important thing to know about setting up Womersley boundary condition in Nektar++. We need to scale velocity for Womersley boundary condition to make sure that there is no change of sign of velocity. For example, we should not have such case: part of size of velocity is above zero is part of it is negative. If such thing happens, the

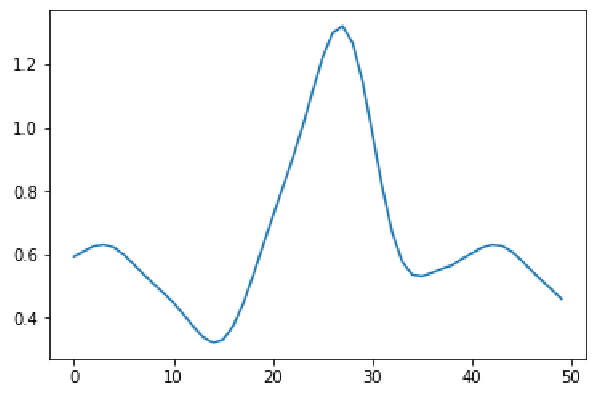


Figure 25: Plot of velocity at a point of inlet from simulation results.

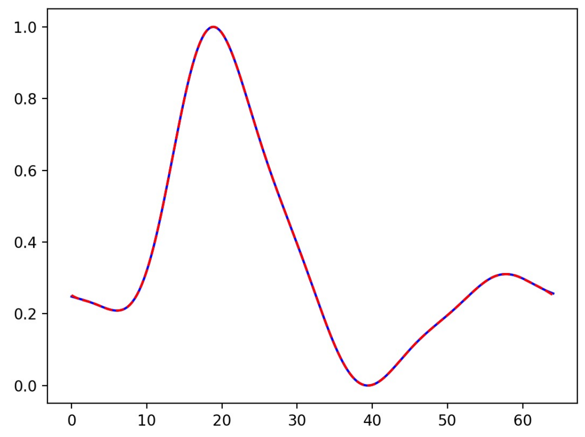


Figure 26: Velocity of the Womersley boundary condition at inlet.

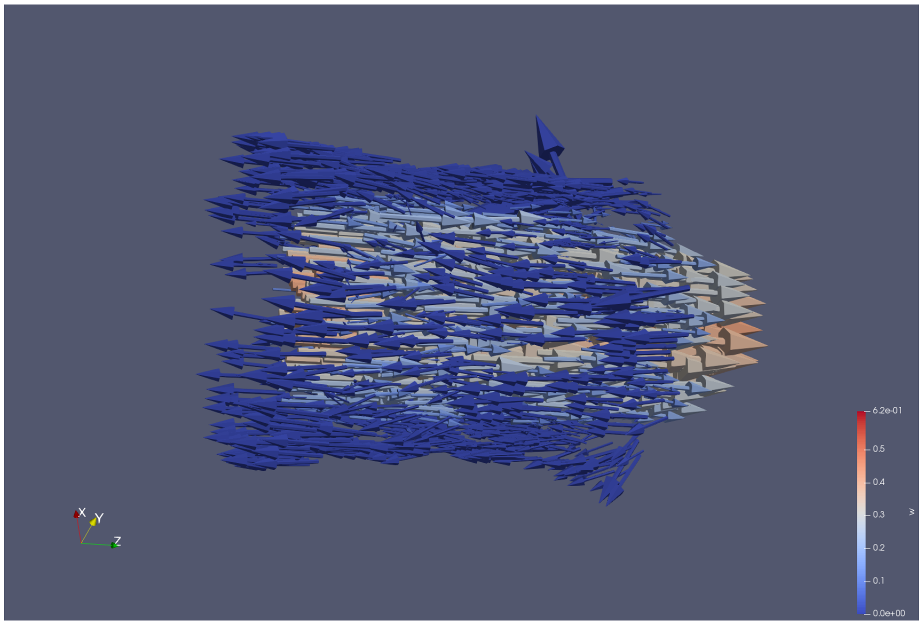


Figure 27: A failed case on cylinder.

simulation will fail. If we visualize the last output file(.chk file) before failure of simulation, we may see situation like Figure 27. From this image, we can see that some fluid is flowing back to inlet(see blue part). So if someone doing the similar simulation and see such case happens, maybe we need to check if the velocity changes sign.

4. Future work

In this section, we will discuss about what to do next base on the work. We will mainly talk about two parts.

4.1. Unfinished work

We have introduced Windkessel model in section 3.2 but we didn't do the simulation with Windkessel model to set pressure boundary condition for outlet in this thesis. The reason for this is that the Windkessel model is under development for Nektar++ version4.4.1(current version). So in our simulation in this thesis, the pressure should not attain physiologically correct values because the boundary condition is wrong.

4.2. Looking towards new paradigm of scientific computing

In this thesis, we discussed about traditional method in computational fluid dynamics:(i)generate the mesh for computational domain. (ii)set boundary condition for simulation. (iii)using traditional numerical methods to solve partial differential equation. However, recently, with development of machine learning(deep learning) techniques, people can totally change the work flow. The high-level idea of this new paradigm is using the extremely powerful fitting capacity of deep neural networks to fit the solution of differential equations. An obvious advantage for this methodology is the neural network we trained is essentially a function which can predict value of variables for each position of computational domain, which can allow us avoid generating mesh and setting boundary conditions. In addition to this, people can also use machine learning method to infer or estimate parameters of differential equations. People may see[8] as a very good example which is also dealing with blood flow simulation.

Bibliography

[1] Bermejo J, Martnez-Legazpi P, del lamo JC. The clinical assessment of intraventricular flows. Annu Rev Fluid Mech. 2015;47(1):31542.

[2] Bobbia X , Grandpierre RG , Claret P-G et al. . Ultrasound guidance for radial arterial puncture: a randomized controlled trial . Am J Emerg Med 2013 ; 31 : 810 5

[3] C. D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R. M. Kirby, and S. J. Sherwin, Nektar++: An open-source spectral/hp element framework, Computer physics communications, vol. 192, pp. 205-219, 2015.

[4] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering 79(11), pp. 1309-1331, 2009.

[5] Chan BT, Lim E, Chee KH, Abu Osman NA. Review on CFD simulation in heart with dilated cardiomyopathy and myocardial infarction. Comput Biol Med. 2013;43(4):37785.

[6] Cohn JN, Ferrari R, Sharpe N. 2000. Cardiac remodelingconcepts and clinical implications: a consensus paper from an international forum on cardiac remodeling. J. Am. Coll. Cardiol. 35:56982.

[7] G.E. Karniadakis, M. Israeli, S.A. Orszag High-order splitting methods for the incompressible Navier-Stokes equations J.Comput. Phys.,97 (2) (1991), pp. 414-443

[8] Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R Witschey, John A Detre, Paris Perdikaris Machine learning in cardiovascular flows modeling: Predicting pulse wave propagation from non-invasive clinical measurements using physics-informed deep learning arXiv:1905.04817

[9] Hornberger LK, Sanders SP, Rein AJ, Spevak PJ, Parness IA, Colan SD. 1995. Left heart obstructive lesions and left ventricular growth in the midtrimester fetus: a longitudinal study. Circulation 92:153138.

[10] I. Babuka, B.Q. Guo: The h, p and h-p version of the finite element method: basis theory and applications, Advances in Engineering Software, Volume 15, Issue 3-4, 1992.

[11] J.L. Guermond, J. Shen Velocity-correction projection methods for incompressible flowsSIAM J. Numer. Anal., 41 (1) (2003), pp. 112-134

[12] Karniadakis, G. and Sherwin, S. (2013).Spectral/hp element methods for CFD. Oxford: Oxford University Press.

[13] Markl, M., Frydrychowicz, A., Kozerke, S., Hope, M., Wieben, O., 2012. 4d flow mri. Journal of Magnetic Resonance Imaging 36 (5),10151036.

[14] Ma,Y.,Choi,J.,HourlierFargette,A.,Xue,Y.,Chung,H.U.,Lee,J.Y.,Wang,X.,Xie,Z.,Kang,D.,Wang,H.,eta pressure and pulse wave velocity for human arteries. Proceedings of the National Academy of Sciences 115 (44), 1114411149.

[15] N Doost, Siamak Ghista, Dhanjoo Su, Boyang Zhong, Liang Morsi, Yosry (Yos. (2016). Heart blood flow simulation: A perspective review. BioMedical Engineering OnLine. 15. 10.1186/s12938-016-0224-8.

[16] Olgierd Cecil Zienkiewicz and Robert Leroy Taylor. Basic formulation and linear problems. McGraw-Hill, 1989

[17] ORourke,M.,1995.Mechanical principles in arterial disease.Hypertension26(1),29.

[18] Schwemmer U , Arzet HA , Trautner H , Rauch S , Roewer N , Greim C-A . Ultrasound-guided arterial cannulation in infants improves success rate . Eur J Anaesthesiol 2006 ; 23 : 476 80

[19] Serson D., Meneghini J. R., Sherwin S. J. Velocity-correction schemes for the incompressible navier-stokes equations in general coordinate systems [J]. Journal of Computational Physics, 2016, 316: 243254.

[20] Shiver S , Blaivas M , Lyon M . A prospective comparison of ultrasound-guided and blindly placed radial arterial catheters . Acad Emerg Med 2006 ; 13 : 1275 9

[21] Stefanovska,A.,1999.Physics of the human cardiovascular system.Contemporary Physics40(1),3155.

[22] Tan TYS , Petersen JAK , Zhao X , Taylor KL . Randomized controlled trial of ultrasound versus palpation method for arterial cannulation in infants less than 24 months of age . SOJ Anesthesiol Pain Manag 2015 ; 2 : 1 3

[23] White, L; Halpin, A; Turner, M; Wallace, L (22 April 2016). "Ultrasound-guided radial artery cannulation in adult and paediatric populations: a systematic review and meta-analysis". British Journal of Anaesthesia. 116 (5): 610617.

[24] Womersley, J.R. (March 1955). "Method for the calculation of velocity, rate of flow and viscous drag in arteries when the pressure gradient is known". J. Physiol. 127 (3): 553563.

[25] `https://www.paraview.org`

[26] `https://en.wikipedia.org/wiki/Continuity_equation`

[27] `https://www.geophysik.uni-muenchen.de/~igel/Lectures/NMG/08_finite_elements_basisfunctions.pdf`

[28] https://www.skience.de/2017/mess-2017-introduction-spectral-element-method

[29] https://en.wikipedia.org/wiki/Pulsatile_flow#References

[30] https://en.wikipedia.org/wiki/Windkessel_effect

[31] https://sinews.siam.org/Details-Page/cardiovascular-blood-flow-simulation

[32] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.716.8008&rep=rep1&type=pdf

[33] https://www.brown.edu/research/projects/crunch/sites/brown.edu.research.projects.crunch/files/uploads/Paris\%20Perdikaris\%20Thesis.pdf

[34] Szeg, Gbor (1939). "IV. Jacobi polynomials.". Orthogonal Polynomials. Colloquium Publications. XXIII. American Mathematical Society.

[35] Abramowitz, Milton; Stegun, Irene Ann, eds. (1983) [June 1964]. "Chapter 22". Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. Applied Mathematics Series. 55 (Ninth reprint with additional corrections of tenth original printing with corrections (December 1972); first ed.). Washington D.C.; New York: United States Department of Commerce, National Bureau of Standards; Dover Publications. p. 561. ISBN 978-0-486-61272-0. LCCN 64-60036. MR 0167642. LCCN 65-12253.

[36] P.K. Suetin (2001) [1994], "Jacobi polynomials", in Hazewinkel, Michiel (ed.), Encyclopedia of Mathematics, Springer Science+Business Media B.V. Kluwer Academic Publishers

[37] Biedenharn, L.C.; Louck, J.D. (1981). Angular Momentum in Quantum Physics. Reading: Addison-Wesley.

[38] Andrews, George E.; Askey, Richard; Roy, Ranjan (1999), Special functions, Encyclopedia of Mathematics and its Applications, 71, Cambridge University Press, ISBN 978-0-521-62321-6, MR 1688958, ISBN 978-0-521-78988-2

[39] Koornwinder, Tom H.; Wong, Roderick S. C.; Koekoek, Roelof; Swarttouw, Ren F. (2010), "Orthogonal Polynomials", in Olver, Frank W. J.; Lozier, Daniel M.; Boisvert, Ronald F.; Clark, Charles W. (eds.), NIST Handbook of Mathematical Functions, Cambridge University Press, ISBN 978-0521192255, MR 2723248

[40] Pedlosky, Joseph (1987). Geophysical fluid dynamics. Springer. pp. 1013.

[41] Clancy, L.J.(1975), Aerodynamics, Section 3.3, Pitman Publishing Limited, London

[42] Quantum Mechanics Demystified, D. McMahon, McGraw Hill (USA), 2006.

[43] D. McMahon (2006). Relativity DeMystified. McGraw Hill (USA).

[44] C.W. Misner; K.S. Thorne; J.A. Wheeler (1973). Gravitation. W.H. Freeman Co.

[45] Michael Weiss; John Baez. "Is Energy Conserved in General Relativity?". Retrieved 2014-04-25.

[46] J.A. Wheeler; C. Misner; K.S. Thorne (1973). Gravitation. W.H. Freeman Co. pp. 558559. ISBN 0-7167-0344-0.

[47] Hydrodynamics, H. Lamb, Cambridge University Press, (2006 digitalization of 1932 6th edition)

[48] Introduction to Electrodynamics (3rd Edition), D.J. Griffiths, Pearson Education Inc, 1999, ISBN 81-7758-293-3

[49] Electromagnetism (2nd edition), I.S. Grant, W.R. Phillips, Manchester Physics Series, 2008 ISBN 0-471-92712-0

[50] Gravitation, J.A. Wheeler, C. Misner, K.S. Thorne, W.H. Freeman Co, 1973, ISBN 0-7167-0344-0

[51] Curtiss, C. F., Hirschfelder, J. O. (1952). Integration of stiff equations. Proceedings of the National Academy of Sciences, 38(3), 235-243.

[52] Ascher, U. M.; Petzold, L. R. (1998), Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations, SIAM, Philadelphia, ISBN 0-89871-412-5.

[53] Daryl L. Logan (2011). A first course in the finite element method. Cengage Learning.

[54] Reddy, J. N. (2006). An Introduction to the Finite Element Method (Third ed.). McGraw-Hill.

[55] Hrennikoff, Alexander (1941). "Solution of problems of elasticity by the framework method". Journal of Applied Mechanics. 8.4: 169175.

[56] Courant, R. (1943). "Variational methods for the solution of problems of equilibrium and vibrations". Bulletin of the American Mathematical Society. 49: 123.

[57] Olek C Zienkiewicz; Robert L Taylor; J.Z. Zhu (31 August 2013). The Finite Element Method: Its Basis and Fundamentals. Butterworth-Heinemann.

[58] Bathe, K.J. (2006). Finite Element Procedures. Cambridge, MA: Klaus-Jrgen Bathe.

[59] Smith, I.M.; Griffiths, D.V.; Margetts, L. (2014). Programming the Finite Element Method (Fifth ed.). Wiley.

[60] P. Solin, K. Segeth, I. Dolezel: Higher-Order Finite Element Methods, Chapman Hall/CRC Press, 2003

[61] Bengt Fornberg (1996) A Practical Guide to Pseudospectral Methods. Cambridge University Press, Cambridge, UK

[62] Chebyshev and Fourier Spectral Methods by John P. Boyd.

[63] Canuto C., Hussaini M. Y., Quarteroni A., and Zang T.A. (2006) Spectral Methods. Fundamentals in Single Domains. Springer-Verlag, Berlin Heidelberg

[64] Polynomial Approximation of Differential Equations, by Daniele Funaro, Lecture Notes in Physics, Volume 8, Springer-Verlag, Heidelberg 1992

[65] D. Gottlieb and S. Orzag (1977) "Numerical Analysis of Spectral Methods : Theory and Applications", SIAM, Philadelphia, PA

[66] J. Hesthaven, S. Gottlieb and D. Gottlieb (2007) "Spectral methods for time-dependent problems", Cambridge UP, Cambridge, UK

[67] Lloyd N. Trefethen (2000) Spectral Methods in MATLAB. SIAM, Philadelphia, PA

[68] Javier de Frutos, Julia Novo: A Spectral Element Method for the Navier–Stokes Equations with Improved Accuracy