POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Tesi di Laurea Magistrale

SubGrid Scale Models for Variational Germano Method

Luca Marre' Brunenghi

POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Tesi di Laurea Magistrale

SubGrid Scale Models for Variational Germano Method

Relatore:
Prof. Domenic. D'Ambrosio

Supervisor:
Prof. Steven .J. Hulshoff

Candidato:
Luca Marre' Brunenghi

Dicembre 2018

INDEX

CHAPTER 1

1. INTRODUCTION

There are three types of simulation to deal with the numerical computation of a turbulent flow: DNS (direct numerical simulation), LES (Large eddy simulation) and RANS (Reynolds averaged Navier Stokes).

In general, the first one numerically resolve the three dimensional Navier Stokes Equation without involving approximation, except for those due to discretization; the second, after decomposing the flow into large scales contribution and small scales contribution, explicitly calculate the large scale contribution whereas the small scale contribution is described as a model; the third one after decomposing the physical quantities in a mean part and a fluctuating part resolve the NS equation with the help of a closure model to describe Reynolds stress (k ε model is one of most used).

We focus on the LES calculating the solution using the Variational Multiscale Method (VMM). In fact, a turbulent flow contains a large range of scales (the range is proportional to the physical Reynolds number) so the calculator needs too many operations to obtain a DNS.

Instead using the VMM the solution is decomposed into

$$u = u^h + u' \quad (1)$$

Where u^h represents the numerically resolved scales and u' the unresolved scales whose effect on the resolved scales are not calculated but estimated, thanks to a model term that will be inserted in the initial equation. This is useful because it reduces the CPU time needed to obtain the solution. So the main target is to choose a model that brings a solution as close as possible to the numerical exact one .The method used to adapt the model term to the analyzed problem is the Variational Germano Method that can be used to obtain dynamically the parameters on which it depends the model term. This method doesn't need external input but only the numerically resolved scales solution (called coarse solution) so it is more general and more fitted to the specific problem analyzed.

CHAPTER 2

2. LITERARY REVIEW

The variational multiscale method is a procedure for deriving models and numerical methods capable to dealing with multiscale phenomena.

Let the strong formulation of a PDE problem

$Lu = f$ on a domain Ω (2)

u= g on the boundary Γ

Where *L* is a general differential operator, $g: \Gamma \to \mathbb{R}$ and $f: \Omega \to \mathbb{R}$ given functions

Let $S \subset H^1(\Omega)$ and Let $V \subset H^1(\Omega)$ the trial solution space and the weighting function space where

- u = g on Γ for every $u \in S$
- w = 0 on Γ for every $w \in V$

So the weak variational form of the PDE become

B(w,u)=(w,f) (3)

Where (,) is the L_2 inner product and B(,) is a bilinear form that satisfy

B(w,u)= (w,*L*u) (4)

The purpose of VMM consists in decomposing the solution into

$$u = u^h + u'$$

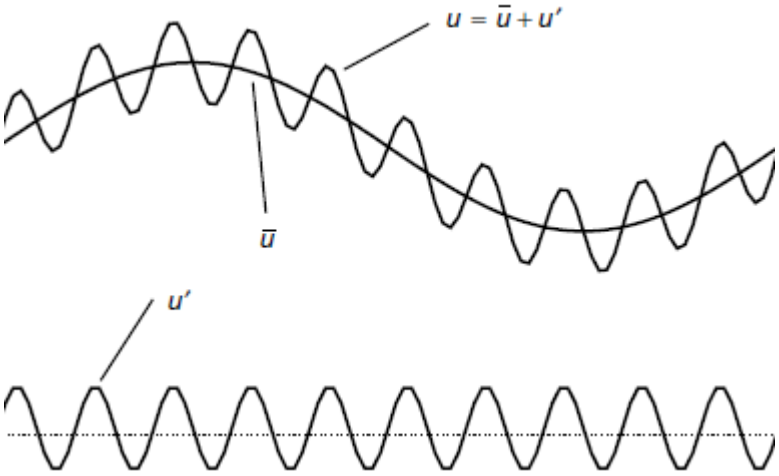


Figure 1 Coarse and fine scales separation

And numerically solving u^h finding an analytically expression for u'.

It is possible to decompose u

1) thanks to a filter operation, like it has been developed in [1]
2) thanks to a Projection operator [2]

In the first case it's possible to use a lot of different filters (for example a Fourier cut-off) characterized by their filter-width. In this way a spatial average of the PDE is employed and the so called "filtered equation" takes the place of the original PDE. To compute the numerical solution of the filtered equations is necessary solve a closure problem (in the case of NS equation it consist in computing $\overline{u \otimes u}$) that entails some form of approximation; one possible solution is introducing the so called "sub grid scale stress" that can be estimated for example by the Smagorinsky eddy viscosity model (1963) as explained in [1]. The introduction of the sub grid stress in necessary to take into account of the unresolved scales u'.

2) can be considered the evolution of 1) because, avoiding filters, it will eliminate a lot of numerical and analytical difficulties dealing with inhomogeneous or non-commutative filters, necessary for complex problems. Selecting the projector $\mathbb{P}$ (a lot of choices are possible for example L^2 projector H^1 projector, projector nodal interpolant and so on) means decomposing *V* in a finite dimensional coarse scale subsystem $\bar{V}$ and an infinite dimensional fine scale subsystem *V'*. ($V = \bar{V} + V'$) so

$$\bar{u} = \overline{\mathbb{P}}\, u \quad (5)$$

$$u' = u - \overline{\mathbb{P}}\, u \quad (6)$$

Similarly can be done for w.

Then it's possible to decompose the original variational equation into a coarse scale equation and a fine scale one. The first will be numerical solved inserting in it the approximate value of u' estimated thanks to the second equation [2]. In fact, it is possible to demonstrate that u' can be approximated by

$$u' = \tilde{F}'\left(\bar{u}, Res\ (\bar{u})\right) \quad (7)$$

Where $Res\ (\bar{u})$ is the residual of the first equation and $\widetilde{F}'$ is the approximation of the exact differential functional *F '*which would make the equation exact.

It is clear that both decomposing solution need a closure model to calculate the solution. As written before, the first solution was the Smagorinsky Eddy Viscosity model; however it results always too diffusive and the coefficients on which it depends are difficult to find a priori.

So it has been developed the so called "Germano identity" and its variational counterpart ([3] [4]-[5]) and its filtered one [6] to dynamically evaluate the coefficients without any a priori information about the PDE.

In [4] a generalization of the variational Germano Identity (VGI) has been presented and has been exploited to evaluate the correction model coefficients using a last square method and dissipation one .The 1d linear advection equation and the decay of homogenous isotropic turbulence has been analyzed to underline how this solution results more accurate than the traditional static Smagorinsky approach.

In[7] different kinds of projectors are analyzed because, in order to obtain a solution as close as possible in a user defined optimal metric, different projectors are needed. So the numerical solution depends on the metric chosen.

In [6]it is proved that VMS formulation provides a recipe for constructing sub grid models that will produce the desired numerical solution while VGI provides a method by which it is possible to determine the value of the VMS parameters to obtain a solution to the wished one.

In [8] the equation

$$u' = \tilde{F}'\left(\bar{u}, Res\ (\bar{u})\right)$$

Is approximated by the first term of Taylor series expansion; so u' is proportional to $Res\ (\bar{u})$. This formulation called RBVM (residual based variational multiscale) is simple and seems yielding good results. However the author's formulation can't take into account the Reynolds stress term so he combines RBVM with a Smagorinsky model to take into account that part too.

In [9] a general Germano identity is delivered; it is similar to VGI but it can't be just for variational numerical methods . An algorithm to solve the system of equation deriving from the new Germano identity is proposed. Several approaches to compute the course stabilization parameter are presented pointing out possible solutions to introduce non homogenous boundary conditions and to develop a last squares method basis independent. At the end it has been studied the relaxation algorithm to compute the fine scale stabilization parameter.
In [5] a variational Germano identity was compared to a filtered form one both using a last square method and a dissipation method to compute the coefficients of the corrective model. The differences and the analogies are analyzed through the numerical study of the decay of homogenous turbulence.
In this thesis different subgrid scales model are analyzed in order to find the one which can best approximate the solution of the Burger equation forced so that it can be possible to simulate a sort of 1D turbulence. The structure of the thesis is the following: in the section (3) are analyzed all the numerical models and methods used to find the solution; in the section (4) are described all the software feature of the application and the detail of the application structure; in (5) are described the analysis cases, so the different force terms and the different subgrid models used; in (6) the numerical analysis is performed and in (7) are presented the conclusion of this work.

CHAPTER 3

3. ANALYSIS FEATURE

3.1 Equation

In this thesis we will consider the viscid Burgers equation.

So we try to solve numerically the nonlinear 1D parabolic PDE (Partial Differential Equation)

$$u_t + uu_x - vu_{xx} = f \qquad (8)$$

Where

- $v > 0$ is the constant viscosity
- f is the forcing term
- $(\cdot)_x = \frac{d(\cdot)}{dx}$
- $(\cdot)_t = \frac{d(\cdot)}{dt}$

Burgers equations can be considered a simplification of a more complex model. So it is usually thought as a toy model, namely, a tool that is used to understand some of the inside behavior of the general problem. For example the Navier Stokes equation has the same structure of (1).

As the matter of fact for the momentum incompressible Navier Stokes Equation is

$$\rho u_t + \rho uu_x + \rho vu_y + \rho wu_z + p_x - \mu\left(u_{xx} + u_{yy} + u_{zz}\right) = f \qquad (9)$$

If we consider its 1D version and we neglect the pressure gradient it becomes

$$\rho u_t + \rho uu_x - \mu u_{xx} = f \qquad (10)$$

Which is the above viscid Burger equation.

We will focus on this equation because of the nonlinear term which is the main obstacle to solve both analytically both numerically the Navier Stokes equation [10]. So we try to find a stable and consistent numerical method which can solve the Burger equation in order to apply it to the fluid dynamic equations.

There are a lot of mathematical and numerical method to solve the Burger equation PDE. Among the mathematical ones, for the inviscid case, we can mention the characteristic method which try to reduce a PDE in a ODE (Ordinary Differential Equation) along the so called “characteristic curves” .So first we have to find the curves then solve the ODE at the end transform the solution to adapt it to the original PDE. On the contrary two of the most used numerical method are

- The upwind method, which try to discretize the hyperbolic partial equation in the direction of the propagation of the signal in the flow field (two example are the Lax Wendroff and the Lax Friedrichs schemes)
- The Godunov methods whose feature is to solve the Riemann problems at the interfaces of the finite volume elements. It is the base to high order methods.

However we decide to use a VMM to reduce the computational effort required to take into account a lot of different scales that is the main feature of a turbulent flow.

3.2 Finite-element Method (FEM)

As suggested in [11] finite-difference methods are easy to apply using structured meshes, for which gently-curved domains can be effectively treated using generalized transformations. However, it can be difficult to generate high-quality structured meshes in very complex domains. In these cases the use of unstructured meshes (including e.g. triangles/tetrahedrons and other shapes) is often favored. Finite-difference methods can be derived for such meshes, but they are complex to deal with. This is not a problem for the finite-element method, however, which instead considers an integral form of the problem which allows for elements with arbitrary shapes and orientations. Finite-element methods approximate the solution with combination of known functions.

The main feature of a finite element method for the solution of a differential problem are

- The weak formulation of the problem
- The approximate solution of the variational equation through the FEM functions

The first step to define a FEM is to approximate the solution as linear combination of known function called basis. We can write as example for a 1D problem

$$\hat{u} = \sum_{i=1}^{N} a_i \, \Phi_i(x) \qquad (11)$$

Where

- $\hat{u}$ is the approximate solution of the problem
- $\Phi_i(x)$ basis functions. The basis function has to be linear independent. One of the most used basis function are the Lagrange linear basis which are 0 in all domain except for one point.
- a_i are unknown coefficients. In the case of the linear Lagrange basis a_i represent the value of u in a certain point of the domain. In other cases these coefficients has only a mathematical value.
- N is the dimension of the space of Φ_i

So the problem becomes how to evaluate the a_i coefficients. It can be done in different ways. In the structural mechanics, it is used the Rayleigh-Ritz approach which is based on the minimization of the stationary potential energy of the system. So the procedure is to obtain a_i as the solution of an algebraic system obtained minimizing a functional associated to the global potential energy of the system.
Another important method which lead to FEM is the Galerkin method or the weighted residual method. It is based on the use of the weak formulation of the differential problem
If $Lu = f$ is the equation of the strong formulation of the problem his weak counterpart will be

$$\int_{\Omega} w(Lu - f)d\Omega = 0 \qquad (12)$$

For all the suitable w. In the Bobunov Galerkin method w is equal to the weighting functions. In the discrete case the weak formulation results in a system of N equations whose unoknow are the a_i coefficients. In the Galerkin method is important choosing Φ_i so that they are square-integrable.

The finite element method is similar both to Galerkin both Rayleigh-Ritz ones. The main difference is that the basis function are define only on a finite element of the mesh. The 5 steps, that a FEM procedure requires, are [12]

- Divide the domain into elements. Is is note necessary if we have got a complete mesh
- Define basis functions which span a small number of elements. Define the basis function means state the typology of the element. Different kind of finite element has different basis functions.
- Compute per-element contributions to the weak form following a procedure that is similar to the Galerkin method
- Assemble the element contributions into a global matrix. In the steady case the global matrix are called stiffness matrix. Since each element of the matrix is associated to a node of the mesh, the value that will been computed, will be the sum of the contribute of each elements to which the node belongs.

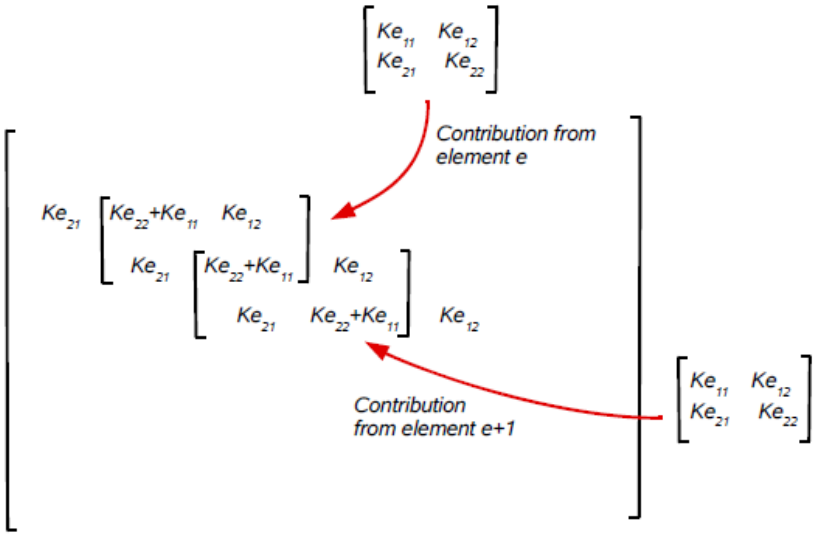


Figure 2 Stiffness matrix assembling

- Apply the boundary conditions and solve the resulting system. Applying the boundary condition means adding or removing equation in basing on the type of the boundary condition chosen.

Fem can be used in complex geometry analysis because, once the mesh is defined, all other step are easily implemented since they are not influenced by the form of the domain.
There are2 approach to deal with an unsteady discrete finite element problem. The first one is the semi discrete approach where the time derivative is initially left as continuous variable of the formulation. his results in a system of ordinary differential equations which can be integrated using conventional time-marching techniques. In this case a matrix, called mass matrix, is associated to the time dependent term of the equation. So the procedure has to assemble not only the stiffness matrix but also another one. The other one is the fully discrete approach consist in considering the time as one of the dimension of the domain. So it is necessary to define finite elements with one more discretization. For example in 1D problem we need to use 2D finite elements. They are very useful in case of changing domains since, usually, because the time step are evaluate in a 2D problem one by one. So also in this case the previous time step can be used as initial condition for the following one. The big trouble is that it is very difficult cope with the stability of the computation.

The code used for this thesis works thank to a semi discrete approach: in the following chapters the complete structure of the app will be described.

3.3 Variational Multiscale Method (VMM)

As far as computation fluid dynamic is concerned there are several different kind of simulation that can be done to solve a PDE so to decide the structure of the numerical method, as it was already written in the introduction: DNS, LES and RANS.

The big difference between a DNS and a LES of a multiscale flow is that the DNS has to take into account all of them (from the biggest to the Kolmogorov ones); a LES splits the scales in two parts: the resolved scales $\bar{u}$ and the unresolved scales, which need a model to be described. So the goal of a LES is to find $\bar{u}$ and not *u* finding the best approximation for u' in order to obtain a $\bar{u}$ as close as possible to u.

The first attempt to define a model which can close the problem of the subgrid scale was done by Smagorinsky [13].

He considered the filtered Navier Stokes equations

$$\overline{u_t} + \nabla \cdot (\overline{u \otimes u}) + \nabla \bar{p} = \nu \Delta \bar{u} + \bar{f} \qquad (13)$$

because of the fact that $\overline{u \otimes u}$ involves non filtered velocity he defined the subgrid stress

$$T = \bar{u} \otimes \bar{u} - \overline{u \otimes u} \qquad (14)$$

He state that the dilational part of T can be subsumed by $\bar{p}$ while the deviatoric part can be model as

$$T_s = 2\nu_t \nabla^s \bar{u} \qquad (15)$$

This equation is the most important part of the Smagorinsky eddy viscosity model. We will describe better the terms contained in the previous formulation

- $\nu_t = (C_s \Delta)^2 |\nabla^s \bar{u}|$ is the eddy viscosity
- $\nabla^s \bar{u} = \frac{1}{2}(\nabla \bar{u} + (\nabla^s \bar{u})^T)$
- $|\nabla^s \bar{u}| = (2\nabla^s \bar{u} \cdot \nabla^s \bar{u})^{1/2}$
- $C_s$is the so called " Smagorinsky constant"

This model presents a lot of problems for example

- T_s is not asymptotic close to the wall; in particular T_s doesn't vanish at walls
- The value of C_s , chosen for the study of the homogeneous isotropic turbulence, tends to be too big if used in other simulation (like the turbulent channel)
- The Smagorinsky model produces excessive numerical dumpingof the resolved structure in transition, resulting in incorrect growth rate of perturbations

As a consequence has been developed the Germano method (explained in the following chapter) and the VMM that we are going to describe.

First of all we will define mathematically the problem

Let Ω an open bounded domain and Γ his boundary. We want to find $u: \Omega \to \mathbb{R}$ such that

$Lu = f$ in Ω

u= g on Γ

Where (as explained in the previous chapter)

- *L* is a general differential operator
- $g: \Gamma \to \mathbb{R}$ and f: $\Omega \to \mathbb{R}$ given functions

Moreover *f* is the so called forcing function and it will be one of the main parameter of the analysis.

We define again

Let $S \subset H^1(\Omega)$ and Let $V \subset H^1(\Omega)$ the trial solution space and the weighting function space where

- u = g on Γ for every $u \in S$
- w = 0 on Γ for every $w \in V$

In fact we try to solve the problem in the so called weak formulation or the weighted residual method. The variational form of the problem will be the following

$$\int_{\Omega} w(Lu - f)d\Omega = 0 \qquad (16)$$

Which can be written in the compact notation find $u \in V$ so that

$$B(w,u) = (w, Lu)_{\Omega} = (w, f)$$

Where $(\cdot,\cdot)$ is the standard L_2 inner product and $B(a,b) = (a, Lb)$

The VMM proposed split u in 2 parts the resolved scales $\bar{u}$ and the unresolved scales *u'*. Similarly we split the weighting function into $\bar{w}$ and *w'*. So we try to find $\bar{u}$ in the finite dimensional subspace $\bar{S}$ ($\bar{V}$ is finite dimensional too) where S' and V' are infinite dimensional. We have to define projectors or filters to define adequately $\bar{u}$ and $\bar{w}$ but we discuss this aspect later.

Since

$$u = \bar{u} + u'$$

And

$$w = \bar{w} + w' \quad (17)$$

The problem becomes

$$B(\bar{w} + w', \bar{u} + u') = (\bar{w} + w', f) \quad (18)$$

Since $\bar{w}$ and w' are independent we can split the problem in 2 parts

$$B(\bar{w}, \bar{u}) + B(\bar{w}, u') = (\bar{w}, f) \quad (19)$$

$$B(w', \bar{u}) + B(w', u') = (w', f) \quad (20)$$

The first one is the resolved equation (coarse scale equation)whose solution is the exact one if u'=0 (DNS case). However in LES u' is not 0 but it has to be estimated solving approximately the second equation (fine scale equation). If we know u' we can substitute it in the first one and find the solution of the problem.

As can be seen in [1],thanks to the definition of a Green function g', we can rewrite the finer scale equation to obtain a definition of u'(this is true only because we have linearized the second equation: the biggest part of the energy is contained in the coarse scales)

$$u' = \int_{\Omega} -\mathrm{g}'(L\bar{u} - f)\mathrm{d}\Omega = \mathrm{M}'(L\bar{u} - f) \quad (21)$$

Where M' is an integral operator.

We have to underline that $L\bar{u} - f$ is the coarse scale residual sol u' derives from the coarse scale solution and the g' is the Green function associated to the problem. If we knew the exact formulation of g' we would exactly solve the problem (the discretization error is the unresolved scales modeling error).

Generally if we use, like it happens in this thesis, a mesh based method (FEM) $\bar{u}$ are called resolved scales and u' are the subgrid scales. So the coarse scale equation need a subgrid scale model because we don't know the exact formulation of g'. We have to remark also that the previous formulation has to be lightly modified to take into account the differences in slope in case of linear finite element mesh.

If we want to best describe the problem we have to reduce the subgrid scale influence since we can only estimate them, so we want that the biggest part of the turbulent energy is contained in the coarse scale. However we use LES to reduce the computational effort not solving all the scale of the problem. So we have to find a compromise.

If now we focus on our problem so on the burger equation

$$u_t + uu_x - \nu u_{xx} = f$$

We can write

$$Lu = u_t + uu_x - \nu u_{xx} \qquad (22)$$

So the weak formulation of the problem is

$$(w, u_t) + (w, uu_x) - (w, \nu u_{xx}) = (w, f) \qquad (23)$$

We can split u and w into coarse and fine scales

$$(\bar{w} + w', \bar{u} + u'_t) + \left(\bar{w} + w', \overline{(\bar{u} + u')}(\bar{u} + u'_x)_x\right) - (\bar{w} + w', \nu(\bar{u} + u')_{xx}) = (\bar{w} + w', f) \qquad (24)$$

Then we can split the equations

The coarse scale one

$$(\bar{w}, \bar{u} + u'_t) + (\bar{w}, (\bar{u} + u')(\bar{u} + u')_x) - (\bar{w}, \nu(\bar{u} + u')_{xx}) = (\bar{w}, f) \qquad (25)$$

The subgrid scale one

$$(w', \bar{u} + u'_t) + (w', (\bar{u} + u')(\bar{u} + u'_x)_x) - (w', \nu(\bar{u} + u')_{xx}) = (w', f) \qquad (26)$$

Now we will modify the first equation in this way

- We assume that $(\bar{w}, u'_t) = 0$ because we assume that the unresolved scales respond almost immediately to perturbation
- Integrating by part $(\bar{w}, \nu(\bar{u} + u')_{xx}) = -(\bar{w}_x, \nu(\bar{u} + u')_x)$ (w is 0 on Γ)
- We can consider negligible $(\bar{w}_x, \nu u'_x)$ because the scale at where the viscous effect are important are far from $\bar{w}$
- The advective nonlinear terms can be modify to best fit the problem to a numerical method integrating by part term by term

$$(\bar{w}, (\bar{u} + u')(\bar{u} + u'_x)_x) = -\left(\bar{w}_x, (\bar{u} + u')\overline{(\bar{u} + u')}\right) = -(\bar{w}_x, (\bar{u})^2) - 2(\bar{w}_x, u'\bar{u}) \qquad (27)$$

We have to remark that we have considered a problem characterize by homogenous boundary condition and we have neglected the term with the term u'^{2}.

The result is

$$(\bar{w}, u_t) - (\bar{w}_x, (\bar{u})^2) - 2(\bar{w}_x, u'\bar{u}) + (\bar{w}_x, \nu\bar{u}_x) = (\bar{w}, f) \qquad (28)$$

As far as the second equation is concerned we model the fine scale in this way

$$u' = \tau R(\bar{u}) \qquad (29)$$

Where $R(\bar{u}) = L\bar{u} - f$ so the coarse scale residual.

So τ is a functional which has to best approximate the exact integral operator M. Since the equation is 1 D τ will be a real number. The value of τ depends on a vector of parameter c which has to be evaluate to obtain the best value of u'. τ is known in very few simple cases:

for example for the advection diffusion equation [14]. These cases are useful because thanks to the it is possible to test new numerical methods which try to best represent τ.

So introducing the u' formulation in the first equation we are adding numerical diffusivity to the coarse scale problem in order to taking into account the subgrid scales too.

Finally the variational problem results

Find $\bar{u} \in \bar{V}$ so that

- $B(\bar{w}, \bar{u}) + M(\bar{w}, \bar{u}, c, f, h) = (\bar{w}, f)$ (30)
- $\bar{u}(0,t) = \bar{u}(L,t) = 0$ (L is the end of the domain t is time)
- $\bar{u}(x,0) = 0$
- $B(\bar{w}, \bar{u}) = (\bar{w}, u_t) - (\bar{w}_x, (\bar{u})^2) + (\bar{w}_x, \nu \bar{u}_x)$ (31)
- $M(\bar{w}, \bar{u}, c, f, h) = -2(\bar{w}_x, u'\bar{u}) = -2(\bar{w}_x, \tau R \bar{u})$ (32)
- h is a characteristic dimension of the mesh used
- $R = (\bar{w}, u_t) - (\bar{w}_x, (\bar{u})^2) - (\bar{w}, f)$ (33)
  The second derivatives vanish since we are considering linear FEM

3.4 Variational Germano Method (VGM)

The purpose of this chapter is to explain the methods to evaluate the c coefficients of the $M(\overline{w}, \bar{u}, c, f, h)$ term. If we knew the values of the c's components we would evaluate M and solve the coarse scale equation. To do that we will exploit the so called Germano Identity or more precisely his variational form: the Variational Germano Identity (VGI). It was first developed his filtered counterpart in [15] for the Navier Stokes equation; however in this thesis has been used the VGI.

As suggested in [4] first of all we require that the solution of the numerical method u^h is equal to $v^h = \mathbb{P}^h u$ which is the optimal representation of the solution u in V^h. We have to specify that

- $V^h \subset V$ is a finite dimensional subspace
- $\mathbb{P}^h: V \to V^h$ is an appropriate operator. We can define $\mathbb{P}^h$ as nodal interpolant of the exact solution on V^h or an L_2 or H^1 projector

For example if we are considering a H^1 projector we will use the H^1 semi-norm: $\mathbb{P}^h v$ is defined as the $\operatorname{argmin}_{v^h \in V^h} |v - v^h|_1^2$ where $|\cdot|_1$ is the H^1 semi norm. We can define a similar formulation for the L_2 or the H_0 (or others)projectors.

We have to remind that H^1 is the Sobolev space of functions that are square-integrable and whose derivatives are also square-integrable while L_2 is the space of scalar function which are only square-integrable.

So, since $v^h = u^h$, we can write

$$M(w^h, u^h, c, f, h) = -B(w^h, u^h) + (w^h, f) \forall\, w^h \in V^h \quad (34)$$

If N is the number of the coefficients of the vector c we will write $J \geq N$ more equations

$$M\left(w^{h_j}, u^{h_j}, c, f, h_j\right) = -B\left(w^{h_j}, u^{h_j}\right) + \left(w^{h_j}, f\right) \quad \forall\, w^{h_j} \in V^{h_j} \quad j = 1, \dots, J \quad (35)$$

Where

- $V^{h_j} \subset V^{h_{j-1}} \subset \dots \subset V^{h_2} \subset V^{h_1} \subset V$ are finite dimensional function subspaces
- $\mathbb{P}^{h_j}: V \to V^{h_j}$ is an appropriate map which define a the optimal representation of u in V^{h_j}

We will choose the weight function in $V^{h_j} \cap V^h = V^{h_j}$. So we can obtain the Variational Germano Identity subtracting the two previous equation:

$$M\left(w^{h_j}, u^{h_j}, c, f, h_j\right) - M\left(w^{h_j}, u^h, c, f, h\right) = -\left(B\left(w^{h_j}, u^{h_j}\right) - B\left(w^{h_j}, u^h\right)\right) \quad \forall\, w^{h_j} \in V^{h_j} \quad j = 1, \dots, J \quad (36)$$

The Variational Germano Method (VGM) allows us to find the coefficients of c without the solution of the initial PDE or any external input constant. In order to reduce the variables we will express u^{h_j} in terms of u^h. It is possible if $u^{h_j} = \mathbb{P}^{h_j} u^h$ so it is required that $\mathbb{P}^{h_j}\mathbb{P}^h = \mathbb{P}^{h_j}$. For example it is true if

- $\mathbb{P}^{h_j}$ and $\mathbb{P}^h$are interpolation operators
- $\mathbb{P}^{h_j}$ and $\mathbb{P}^h$ are L_2 projectors (like in this thesis)
- $V^h \subset H^m$ and $\mathbb{P}^{h_j}$ and $\mathbb{P}^h$ are H^n projectors with $n \leq m$

So the VGI will become

$$M\left(w^{h_j}, \mathbb{P}^{h_j} u^h, c, f, h_j\right) - M\left(w^{h_j}, u^h, c, f, h\right) = -\left(B\left(w^{h_j}, \mathbb{P}^{h_j} u^h\right) - B\left(w^{h_j}, u^h\right)\right) \quad \forall\, w^{h_j} \in V^{h_j} \quad (37)$$

$$j = 1, \ldots, J$$

So we have got J equations to evaluate the N coefficients of c. It is possible to write J equations to evaluate the coefficients of c only in the are considering J different refined meshes in order to compute the Germano Identity equation for each level. As a consequence we need J+1 meshes to solve the problem: 1 coarse mesh and J finer than the first one. This is the reason why the term h_j appear in the first term of the VGI.

First of all we must define the weight functions w. There are a lot of methods thanks to which we can do it and solve the problem :

- Dissipation method: if $N = J$ we choose $w^{h_j} = \mathbb{P}^{h_j} u^h$ so the system consist in J scalar equations

$$M\left(\mathbb{P}^{h_j} u^h, \mathbb{P}^{h_j} u^h, c, f, h_j\right) - M\left(\mathbb{P}^{h_j} u^h, u^h, c, f, h\right) = -\left(B\left(\mathbb{P}^{h_j} u^h, \mathbb{P}^{h_j} u^h\right) - B\left(\mathbb{P}^{h_j} u^h, u^h\right)\right) \quad (38)$$

$$\forall\, w^{h_j} \in V^{h_j} \qquad j = 1, \ldots, J$$

  We have only to compute the solution of the scalar algebraic system.

- Least- square method [16]: we choose $w^{h_j} = \phi_A^{h_j}(x)$ where $\phi_A^{h_j}(x)\ A = 1, \ldots, N^{h_j}$ are functions which span V^{h_j}.
  Than we define the residual

$$r_A^j = M\left(\phi_A^{h_j}, \mathbb{P}^{h_j} u^h, c, f, h_j\right) + B\left(\phi_A^{h_j}, \mathbb{P}^{h_j} u^h\right) - \left(\phi_A^{h_j}, f\right) \quad (39)$$

  And we will choose as the coefficients of c those which minimize the residual. We can exploit different algorithms to do that.

So, when c has been evaluated thank to VGM, the model term of the coarse scales equation $M(w^h, u^h, c, f, h)$ can be computed and we can get the coarse scale solution. We need to

define the structure of the model term in order to best represent the approximation of the Green operator.

However about VGM we have to remark that

- We can solve the Germano system because we know all the values of the formulation (obviously except from c)
- If c is not constant in the domain c has to be evaluate in each subdomain where it can be consider constant
- C depends on the definition of the optimal solution (and then of the choice of $\mathbb{P}^h$). In [7] is proved that the definition of the norm thank to which evaluating the best approximation influence the global solution. It is important to remark that the following coarse scale solution won't be optimally close to the exact value (for example obtained through DNS), so it is important the choice of $\mathbb{P}^h$. In this thesis we will consider a L_2 projector so we will minimize the L_2 norm
- In [4] has been proved that this procedure works better than the Smagorinsky model because it is too diffusive

In this work the least square method has been used.

The VGM is very useful because it can be used easily with unstructured grids. The only problem is defining the function w.Moreover it can be used to determine parameters for a generic numerical method.

On the contrary the filtered version of the Germano identity explained in [5] [14] presents a lot of difficulties to be used. First of all because it is necessary to assume that the filtering and the derivatives operation are commutative. Moreover we need a filtering operation $\mathbb{F}^H$ so that $\mathbb{F}^H = \mathbb{F}^H\mathbb{F}^h$ (where $\mathbb{F}^H$ $\mathbb{F}^h$are two special filter where $H > h$). For non homogenous flows, like the channel turbulent flow, it is impossible to aware this problems which will become the cause of important numerical errors.

If we avoid this difficulties, for example analyzing a homogenous isotropic turbulence problem, the VGM and its filtered counterpart presents other differences

- In the filtered case the model term is a tensor in the VGM a vector
- In the VGM the model term contains every term of the semi linear form while in the filtered only the non linear term contributes
- The constraint on the model term of VGM is weaker than the filtered method: the effect of the model on weighting functions outside of V^h is negligible.

Consequently the VGM results a more robust method so it is likely to be used even if both them present the big advantage of not needing external input to evaluate the model term coefficients.

3.5 Orthogonal Subscales (OSS) and Algebraic Subgrid Scales (ASGS)

In the previous chapter, one of the VMMs has been obtained and explained. However it is possible to define different VMMs depending on the subgrid scale approximation that is considered. In this paragraph the differences between the ASGS (Algebraic SubGrid Scales) model and the OSS (Orthogonal SubScales) model are briefly pointed out.

The ASGS model is the VMM already described. It defines

$$u' = \int_{\Omega} -\mathrm{g}'(L\bar{u} - f)\mathrm{d}\Omega = \tau R(\bar{u})$$

It is important to remember that $u \in V$, while $\bar{u} \in \bar{V}$ which is a finite subspace of V. As a consequence $u' \in V'$ and $V = \bar{V} \cup V'$. However since $u' = \tau R(\bar{u})$ and so u' is proportional to $R(\bar{u})$ they belong to the same space.

It is possible to define the selection of the space for the approximation of the subscales as determined by a projection $\mathcal{P}$ [17]. In this case obviously $\mathcal{P} = I$.

It is possible to obtain different VMM models changing the projection $\mathcal{P}$.

In fact, the OSS model, it's obtained following the projection definition

$$\mathcal{P} = \Pi_h^{\perp} = I - \Pi_h \qquad (40)$$

Where Π_h is the projection onto the FE space.

This projection consists in selecting the space of subscales orthogonal to the FE space. (so orthogonal to $\bar{V}$). As suggested in [17] the main motivation of the method is that a stability estimate for the projection onto the FE space of the convective term can already be obtained in the standard Galerkin method and so the only "missing" part is the orthogonal one.

In [18] it proved that OSS method introduces less dissipation in the coarse scale equation than ASGS, which permit to obtain good results since the approximate solution can capture the peaks very well.

3.6 Algorithm for the Solution of the Algebraic Systems

In this part we will briefly describe the algorithms used to solve the Variational Germano Method's system and the coarse scale equation one. It is use respectively the BFGS (Broyden-Fletcher-Goldfarb-Shanno algorithm) and the GMRES (Generalized Minimal RESidual method)

3.6.1 Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS)

The BFGS has the goal of finding the coefficients of the coarse scale equation closure model thanks to the Variational Germano Identity. In particular, as it has already been written before, it is necessary to minimize the residual R of the least-square formulation. Like it is suggested in [3] R is minimized when $\nabla R = 0$. BFGS is a very efficient scheme which can solve $\nabla R = 0$.

First of all we consider a first order Taylor expansion of ∇R

$$\nabla R(\vec{c}) = \nabla R(\vec{c}_0) + B(\vec{c}_0)(\vec{c} - \vec{c}_0) = 0 \qquad (41)$$

Where B is the Hessian of R with respect of $\vec{c}$. We set $\vec{c}_0 = \vec{c}_n$ and $\vec{c} = \vec{c}_{n+1}$ and we obtain

$$\vec{c}_{n+1} = \vec{c}_n - \alpha_n B_n^{-1} \nabla R(\vec{c}_n) \qquad (42)$$

Then the main parameters of the algorithm are presented

- α_n is a parameter that controls the step length. It has been evaluate as suggested in [19] thanks to the inexact line-search algorithm.
- B_n is an approximation to the Hessian at step n. We have to remark that $B_n = I$
- $B_{n+1} = B_n + \frac{y_n {y_n}^T}{{y_n}^T s_n} - \frac{B_n s_n s_n^T B_n}{s_n^T B_n s_n}$ (43)
- $s_n = \alpha_n(\vec{c}_{n+1} - \vec{c}_n)$ (44)
- $y_n = \nabla R(\vec{c}_{n+1}) - \nabla R(\vec{c}_n)$ (45)

The procedure will be stopped when the L_2 norm of ∇R is minor than a tolerance value or when the L_2 norm of the search direction $B_n^{-1} \nabla R(\vec{c}_n)$ is small enough [20].

3.6.2 Generalized Minimal RESidual Method (GMRES)

GMRES is a sort of generalization of the least square method which works very well solving not too big algebraic linear system. It is used to evaluate $\bar{u}(x)$ in the domain solving the coarse scale equation.

If the system is written in the form

$$Ax = b \qquad (46)$$

- A matrix of the system
- b vector known
- x unknown vector

It is possible to find the approximate solution $\tilde{x}$ using an iterative procedure so that the norm of the residual $\|A\tilde{x} - b\|_2$ is minimum. An iterative procedure is usually preferred in case of big or sparse matrix. GMRES is an orthogonal method because, starting with a trial solution x_0, it looks for $\tilde{x}$ in the space $x_0 + K$ so that $(A\tilde{x} - b) \perp L$

Where K is the space where we are looking for the solution and L is the space of the weighting function.

So if $r_0 = b - Ax_0$ we can write that the problem become find $\tilde{x}$ so that

$$\tilde{x} = x_0 + \delta \quad \delta \in K \qquad (47)$$

$$(r_0 - A\delta, w) \quad w \in L \qquad (48)$$

We then define $V = [v_1, \dots, v_m]$ and $W = [w_1, \dots, w_m]$ matrix whose columns are the basis of K and L. So we can write

$$\tilde{x} = x_0 + Vy \qquad (49)$$

$$(W^T AV)y = W^T r_0 \qquad (50)$$

And so

$$\tilde{x} = x_0 + V(W^T AV)^{-1} W^T r_0 \qquad (51)$$

In [21] was explained that GMRES is an algorithm which follow the above procedure where K belongs to the Krylov subspace. The Krylov subspace K_m is made by orthonormal vectors whose form is $p(A)r_0$ where p is a polynomic of A. If $x \in K_m$ $x = \{r_0, Ar_0, A^2 r_0, \dots, A^m r_0\}$.

However to avoid linear dependent vectors it is commonly used the Arnoldi procedure [22]. Resuming the GMRES step are

- First, thanks to Arnoldi procedure, it creates an orthonormal base V_m
- Then, since $\tilde{x} = x_0 + V_m y_m$, GMRES try to find y_m so that the norm residual of $b - A\tilde{x}$ is minimized

CHAPTER 4

4. CODE STRUCTURE

4.1 Object Oriented Programming (OOP)

In this paragraph it is briefly described the OOP, which is the procedure used to write the code use for the thesis.

The Object Oriented Programming (OOP) was created by Nygaard in 1969 in order to create a software which deal with abstract objects characterized by their own function instead of traditional programming method.

An object is something typified by the data and the operation which can operate with it. It is possible to work with an object having no idea of its structure thanks to the operation it allows to do [23].

A class is a set of the objects that has the same structure and functions. More precisely, the classes are Abstract Data Type, and they state what kind of data they can represent. Thank to that feature it is possible to use a class (or better it is possible to use the interface functions that characterize the class) without knowing the details of its implementation. The collocation of data and functions in a single entity, the class, is the central idea of the OOP.

To define a class are necessary 2 operations

- Declaration: it states the data and the member function (called methods) of interface
- Definition of the methods: it states the implementation of the methods. To do that, it is important to define the type of the output of the function, its name, the list of the any input and the code of the function itself.

All the data and the methods of a class are private, except for the so defined public data. Instead, the protected data are those which a derivative class can deal with. This distinction of the data contained in a class is called information hiding and it helps to prevent sharing data from a class to another.

As a consequence, an object is an instance of a class. An object belongs to a class like a variable to his type. To access to the public data of an object or to use a function of its class it is necessary to use the point (·) so the structure will be objectname.Namefunction(input) (or namedata).

Two of the main methods of a class are the constructor and the destructor. If they are not implemented C++ will create a default version of them. The constructor consists in a class function which can initialize an object when it is created without using any other member

function. A constructor can receive some data or function as input to define its internal feature. Instead, a destructor is a function which delete the object to save memory and it can receive value as input

A derivative class is a subclass of a pre-existence class. It inherits data and methods because an object of the derivative class belongs to pre-existence one. From a practical point of view, this fact allows to transmit a set of common characteristics from a base class to a derivative without this leading to a duplication of the code, while offering the opportunity to adapt or extend the behavior to cases of specific use. A subclass can be private protected or public. If it is:

- Public: it can access to public or protected elements of the original class (the public data will remain public and the protected one will remain protected)
- Protected: it can access to the public and protected elements of the original class but they become protected
- Private: it can't access to any element of the original class

It is important to remark that a class can inherit methods and data from different classes: it is the so called "multiple inheritance".

Another important feature of OOP is the polymorphism, i.e. the property thanks to that different objects of the same pre-existent class (but belonging to different subclasses) can be characterized by functions with the same name but different implementation. So we can use the same interface dealing with different class objects .In order to use the polymorphism we have to define a "virtual" function in the original class and then in each subclass implement it in different ways. It is important to remark that we need a pointer to use the "virtual" function (so instead of · it is required ->).

The OPP is very useful because, using a class, the programmer doesn't need to define every time all the properties of an object. In particular, in order to work with fluid dynamic equations, it is very useful define very general classes related to the elements of the equations that you always deal with. For example, it is very useful define a class related to a general numerical model , or a class which contains all the information to define a basis function or one for the value source term (the force term of a differential equation) or for a mesh. Then it is possible to define several subclasses whose code implement specific feature. If we are considering the "model" class, one of its subclass can be a Finite Element model.

As a consequence it is possible to build a lot of classes a priori, without needing information of the specific problem you are going to solve. There are several useful libraries like MEX and MFEM which store a lot of pre-implemented class and subclass that are very useful because they already contains the biggest part of the functions and the data that are requested for a

general (but not only general) analysis. But they are useful also because, if you have solved a specific problem, you can add to the library the classes and the functions you have eventually written, in order to help the next user to deal with the same problem, or use the same method to another one. However it is very important, before adding new class to the library, checking that there are not a pre-existent class to perform that task.

So thanks to these libraries you have to write only the parts of the code that are specific for your application and that can help you to focus on the main characteristic of your analysis. Moreover, having classes which deal with specific part of the problem you can perform a lot of operations internal to the object and share only the data that are required by other object or functions.

In this work the above mentioned libraries MEX and MFEM have been used to create the code. In the following lines we will briefly describe them and the main classes that has been exploited.

4.2 MFEM

MFEM is the so called Finite Element Discretization Library. It contains all the classes that you need in order to perform a finite element analysis. The class are divided into 4 big groups

- Main mesh classes: this group contains 4 high level classes that are
  - *Mesh*: a class of abstract meshes
  - *NCMesh*: A class for non-conforming AMR on higher-order hexahedral, quadrilateral or triangular meshes. It is a Mesh's subclass
  - *Element*: class of abstract element. Some of its subclasses are for example triangle, exagon and so on
  - *Element transformation*
- Main finite element classes: this group contains several high level classes that are
  - *FiniteElement*: which is an abstract class for Finite Elements. Its subclasses are scalar finite element and vector finite element whose subclasses are respectively
    a) nodal finite element, positive finite element, NURBS finite element
    b) quadrilateral finite element, segment finite element, hexaedron finite element
  - *FiniteElementCollection* which is the collection of finite elements from the same family in multiple dimensions. This class is used to match the degrees of freedom of a *FiniteElementSpace* between elements, and to provide the finite element restriction from an element to its boundary.
  - *FiniteElementSpace*: responsible for providing FEM view of the mesh, mainly managing the set of degrees of freedom.
  - *GridFunction*: class for grid function or vector with associated FE space
  - *BilinearFormIntegrator* and *LinearFormIntegrator*: abstract base classes for linear and bilinear integrator
  - *LinearForm*, *BilinearForm* and *MixedBilinearForm*: classes for linear form or vector with associated finite element space and linear form integrators. In the bilinear case it deals with Matrices and not vectors.

- Main linear algebra classes and sources: this group contains several high level classes that are
  - *Operator*: a class of abstract operators
  - *Vector*: a class of abstract vectors
  - *DenseMatrix* and *SparseMatrix* : a class of data type dense and sparse matrix.
  - parse smoothers and linear solvers
- Main parallel classes: this group contains several high level classes that are
  - *ParMesh*: class for parallel meshes
  - *ParNCMesh* A parallel extension of the NCMesh class. The basic idea is that all processors share the coarsest layer ("root elements"). This has the advantage that refinements can easily be exchanged between processors when rebalancing since individual elements can be uniquely identified by the index of the root element and a path in the refinement tree.
  - *ParFiniteElementSpace*: a class of abstract parallel finite element space.
  - *ParGridFunction*: class for parallel grid function.
  - *ParBilinearForm* and *ParLinearForm*: class for parallel bilinear and form.
  - *HypreParMatrix* and *HypreParVector*: wrapper for hypre's ParCSR matrix class. Hypre stand for high performance preconditioner.
  - *HypreSolver* and other hypre classes: abstract class for hypre's solvers and preconditioners.

  As already said, each high level classes is connected to a lot of different subclasses; moreover they can inherit functions and data by more than one high level class. So MFEM contains a lot of different classes which can used to perform very different tasks. However it can happen that the user has to create a new subclass for a specific problem; in that case it will be helped by higher level classes in the code of which all the basic feature needed are implemented.

4.3 Mex

The MEX Library is a group of classes to tailor the MFEM class . It consists on an easy to use build and test system, and several functions which are commonly used in computational fluid dynamics applications. MEX differs from MFEM also because it contains a lot of very useful classes such more than one writer class, a database class, a class for storing integration points data and so on. We will briefly describe the main MEX classes that have been used for this work. The description of the classes has been taken by the MEX wiki.

- *Constrainer*: When solving a problem, one normally constructs a system of ordinary or algebraic equations. Constrainers are objects that are added to such systems to fix a certain set of degrees of freedom. In finite-element methods, this is done by removing the row corresponding to a non-zero test of the degree of freedom (or called dof) and replacing it with an equation like $dof = value$ (a Dirichlet constraint).On the other hand, the *ConCnstr* (continuity constrainer) can

be used to apply periodic boundary conditions to be constraining one degree of freedom to be equal to another $dof_1 - dof_2 = 0$.At the moment the constrainers available in MEX are limited to Dirichlet and continuity constrainers.

- *FEModel*: Often we form models using the Galerkin method, which is based on the variational form of the (system of) PDE to be solved. In MEX, such a system is assembled and solved by a finite-element model, or *FEModel*. A *FEModel* of course contains the resulting system of equations, but these are constructed using some other important objects. The most basic of these, is the finite-element discretization or FED. The FED defines the final unknowns of the system, based on a MFEM finite-element space (*FESpace*) and information concerning periodic boundary conditions. A FED also handles global interpolation in a parallel environment. The actual assembly of the system of equations is performed using *MexIntegrator* objects, which evaluate the contribution of each domain or boundary element to the integrals appearing in the weak form. Finally *FEMode*l will also likely contain a number of *Constrainer* objects which replace certain equations in the system with explicit relations, typically to define known values of the solution on domain boundaries.
- *MexIntegrator*: *MexIntegrators* are used to evaluate contributions to integrals related to assembling the system, or to post processing. They are derived from MFEM integrators, but contain additional data and functions to facilitate the writing of efficient but high-level code. *MexIntegrators* are broadly divided in two types. *DomainIntegrators*, which evaluate contributions to integrals defined in the interior volume of the domain, and *MexBdrFaceIntegrators*, which evaluate the contributions of boundary surfaces.

  Assembling systems: In their standard mode, *MexIntegrators* are passed to a System, which then performs a loop over elements to determine the contribution of all elements to a global weak form. In this mode, a *MexIntegrator* will return a single element matrix or vector each time it is called

  Evaluating domain or boundary quantities: *MexIntegrators* can also be used to compute integral quantities based on solution vectors. In this case a loop over elements is implemented within the integrator itself. In the current version of Mex, there are two versions of the *MexIntegrator* (the second called *MexIntegrator2*). Generally *MexIntegrator2* should be used, as it has several advanced capabilities, particularly with respect to integration with arbitrary functions, and the saving of integration point data. The Original *MexIntegrator* is retained for backwards compatibility, but will eventually be dropped.
- *MexObject*: Many objects in Mex are derived from the *MexObject* base class. From this they inherit standard logging methods (controlled by the *logLevel*, *screenLog* and *fileLog* commands) and procedures for measuring memory use and CPU time. They also can be asked for their name and the number of the process they reside on. Most *MexObjects* have constructors which contain the

arguments: *(ParamDB &prm_, string name_="").* "*prm_*" is the parameter database used to initialise the object. A pointer to this database with the name *"prm"* will be saved for use within the object. The name is an optional argument. If left blank, the object name will be its derived type.

- *Model*: In many applications, one wishes to use an existing discretization at a higher level to perform a task. Examples include the training of a neural net using a fluid discretization, or the combination of a fluid and solid discretization to perform FSI. For these types of problems there exists a standard "model" class that allows one to compute processes described by either a system of either linear of nonlinear equations generically. A model contains a link to either a system of (linear or nonlinear) algebraic equations or a system of (linear or nonlinear) ordinary differential equations. For the former, it also contains a "*solve*" function and the latter an "*advance*" function.

  *Model*s may also contain other models with their own *solve* or *advance* functions. For example, a fluid model may contain another set of equations that are solved in a segregated way to define coefficients for a turbulence model. Then the solve function for this second model would be contained within the fluid model's advance call.
- *ParamDB*: Some of the objects used in Mex have a large number of configuration parameters. Furthermore, this number tends not to be constant as new features are frequently added. Rather than maintaining constantly-changing interfaces, one can use a single parameter interface to pass large amounts of configuration data to an object. The normal approach to this is to read a single database at the start of a run (based on a command-line argument) and then pass the database around so that each object can extract configuration data from it. When multiple instances of the same object are used, one can pass a "*name*" to the database to differentiate the parameters for one instance from those of another.
- *Solver*: *Solver*s are used to find the solution vector of Algebraic systems. If the system is linear, a member of the Mex "*LinearSolver*" class is used. If it is non-linear, a member of the "*NonLinearSolver*" is used. NonLinear solvers, however, often require a "*LinearSolver*" as an input object. The choice of linear solver depends on the problem considered. Direct linear solvers are robust, but scales are not so efficient (typically with N^3, where N is the number of unknowns). Iterative linear solvers have much better scaling, but their convergence is sensitive to the conditioning of the system matrix. When condition numbers are high, iterative solvers usually require a preconditioner (an object which creates an approximate inverse, allowing the definition of well-conditioned problem). For small linear systems (N<100), one can use the Mex "*SerialDirect*" solver. Since it is serial, it solves the same problem on each processor, without communication. As it is insensitive to conditioning, however, the *SerialDirect* solver is often useful for prototyping implementations, when only a single

processor is required. For large systems to be solved in parallel, we typically use *Hypre* . At the moment, the only available non linear solver is the *NewtonNLS*. It implements a simple Newton method and is quite effective for most problems. It has several configuration parameters. An important one as far as speed is concerned is the matrix update interval. Often matrix and preconditioner updates are expensive, while their the effect of updating the Jacobian on convergence can be small (particularly if a good initial guess of the solution is available, as occurs when marching in space-time). In such cases increasing the matrix update interval can dramatically reduce cost.

- *SubModel*: A *subModel* is an independent object which is used to provide information used in the construction of a model's algebraic system, yet can be controlled or configured by other objects or software packages. It has a very generic interface which allows it to be used for many purposes, but in a typical FEM application, it is used to evaluate a double or vector at an integration point. An example would be the evaluation of a subgrid scale model quantity that has been calibrated externally by a Germano procedure or Neural Network. The functions in the *SubModel* can also be linked to functions evaluated by other software.
- *System*: For most discretization techniques, the aim is to replace the continuous problem by one which requires the solution of a system of equations with a finite number of unknowns. For steady problems, discretization then leads to a large algebraic system which may be linear or nonlinear. This is also true for unsteady problems when space-time discretizations are used. When treating unsteady problems, however, it is more common to use a semi-discrete approach, in which initially only the spatial terms of the equations are discretized. This leads to a system of linear or nonlinear ordinary differential equations. Time marching methods are then used to complete the method. The semi-discrete approach is often preferred because it makes it easy to switch between well-known time integration methods, including to families of explicit methods which allow for low-memory decoupled solution procedures. Systems of equations are represented in Mex by "*System*" classes. These can be broadly divided in two types, *AlgSys*: Algbraic systems (which arise from complete discretization), or *ODESys*: Systems of ordinary differential equations (which arise when one dimension e.g. time is not yet discretized). Both may be either linear or nonlinear. Small Systems can be loaded manually. For finite-element problems, we normally use finite-element systems (*FEAlgSys* or *FEODESys*), which obtain equations by integrating over the elements in a domain.
- *TMarch*: Time marches are used to advance systems of ordinary differential equations in time. These may be either explicit (the solution update does not depend on the next time level) or implicit. Implicit time marches typically require a *LinearSolver* as input. Time marches may also be linear or non-linear. Time

marches for non-linear systems of ODEs typically employ multiple stages (e.g. predictor-corrector, Runge-Kutta) or corrector passes (e.g. generalised alpha). As for algebraic systems, avoiding frequent matrix/preconditioner updates can significantly reduce the cost of time marching non-linear problems. It is often possible to use the same matrix/preconditioner combination for several time steps.

- *ValSrc*: Value sources are generic interfaces to objects which can provide data associated with a set of coordinates. They have many uses, including supplying the value of source terms or solutions from other fields at integration points, obtaining statistics, or plotting solutions. You can access value sources in two ways, one where the element containing the requested coordinate is known, and one where it is unknown. For the latter a search is involved, so it is best to pass all of the required coordinates to the *ValSrc* object at the start of the run, and then just ask for the values without giving coordinates. Then the value source will use the previous coordinates so that the corresponding elements are only found once.

4.4 Application Structure

In this chapter the code structure and the process of the numerical simulation have been explained in order to understand the succession of operations that are required to exploit the variational numerical methods, explained in the previous chapters, to evaluate the solution of the Burger equation.

1. The input of the main program are defined. They are
    - The name of the input file, connected to the specific version of the problem to solve. In this way to change some of the feature of the problem(for example boundary condition, number of elements, tau-model and so on) without changing the code and compiling it every times.
    - The space refine number *nspace* which represents how much the mesh will be refined. If the mesh is too much coarse and the simulation doesn't converge, we can refine it, so that the number of elements will be 2^{nspace} times the previous ones.
    - The time refine number *ntime* which represent how much the difference in time Δtbetween a step and the next one in the time integration will be refined. If the Δt is too much big and the simulation doesn't converge, we can refine it so that the Δt will be 2^{ntime} times smaller than the previous ones.
2. The code get the parametric file and it store all them in a *Paramdb* object in order to easily access to them.

3. A lot of parameters, useful for the next step, are get from the database (and the general inputs) and set.
4. Setting of the effective integration time step and the writing solution time step. We have to remark that only if the 2 time step are the same the application will output all the time steps solution.
5. A Burger model object has been associated to each mesh level as suggested by VGM. As explained previously, it let us to write a Germano Identity equation for each level and so to evaluate the coefficient of the closure sub model. A tau sub model has been added to each of them.
6. The solution vector is defined.
7. A solution vector and a finite element space for post processing on the finest mesh are defined
8. Definition of the problem. In this phase the force term and the boundary condition are imposed to the problem. Moreover it is possible to add the exact solution of a problem in order to evaluate the differences between the exact and the approximate solution. The exact solution can be added analytically or as a result of a DNS. It is necessary to note that in this work
    - the boundary condition are set to 0 (at each level obviously).
    - The exact solution of the analyzed cases have been evaluated by a DNS.
    - The definition of the force term is explained in the next chapter.
9. Preparation of the solution output files and definition of the writer objects
10. Preparation of the statistical integrator
11. Set up of the Germano assembler and solver objects. The Germano assembler is first initialized then added to each level Burger's model. In this way the Germano procedure has been computationally implemented in the application. Moreover the BFSG algorithm has been set to be used to solve the Germano linear system.
12. Preparation of the output files for the model coefficients and the projection error.
13. Definition of the projection model for the exact solution object.
14. Preparation of the time march
15. Time march. It is the most important phase. It can be splitted in several parts which are repeated for each time step.
    - Writing of the closure model coefficients of the previous time step. They will be used for evaluating the current step solution. The first coefficients are set by the user in the input file.
    - Evaluation of the solution. This step is divided into
        - Evaluation and assembling of the stiffness matrix and the right hand side vector as required by Galerkin discretization of the Burger equation. We have to remark that in this discretization the closure model has been taken into account. It is important to remark that in this phase the OSS case differ from the ASGS case.

In fact, the different formulation of u' requires a projection of the residual before multiplying it for τ.

- Evaluation and assembling of the mass matrix. It can be done only considering $(w, \bar{u}_t)$ or considering also the terms $(w_x, \bar{u}_t u')$ and $(w_x, u'u')$.
- Advance in time evaluation of the current time step solution in the finest level.

- Germano procedure. This step is divided into
  - Setting of t and Δt
  - Interpolation of the solution from the finest to the coarse scale
  - Assembling of the Germano Identity system
  - Solving of the Germano Identity System thanks to the BFSG algorithm. In this step the new closure model coefficients have been evaluated. They will be used in the next time step.
- Evaluation of the L_2 errors. First it has been computed between the solution and the exact solution, then between the solution and the projection of the exact solution on the coarse mesh

Then the time march is ended.

16. L_2 error evaluation. It is defined as $\|u_{exact} - u^h\|_2$ so the L_2 norm of the difference of the exact and the numerical solution
17. Writing of the results. It consists in the screen output of the computational time and the L_2 global error and the writing of the Gnuplot output files.
18. The end. Stop of the application and memory delectation

CHAPTER 5

5. ANALYSIS CASES

5.1 Forcing Term

The forcing term of a differential equation is the right hand side term which makes the same equation (with the same boundary condition) having very different solutions. In this work two different forcing terms are considered. The so called Gab force term and a stochastic force term.

5.1.1 Gabriel Force Term

This formulation of the force term takes its name from Gabriel Maher who used it in [3]. It is defined in order to vary both on time and in space:

$$f = 10\sin(t)\sin(2\pi x) + 11 \qquad (52)$$

This formulation ensures a sharp layer near the right boundary of the domain.

5.1.2 Stochastic Force Term

The main task of LES is to numerically evaluate the velocity and the pressure field of a turbulent flow. A turbulent flow is characterized by structures of very different scales. This is due to the inertial cascade which transfers energy from the biggest, whose size is connected to the Reynolds number, to the smallest, whose size is proportional to the Kolmogorov scale. This is the reason why the velocity field change every time so that the turbulence can be considered a random phenomenon which can be studied statistically.

The use of the stochastic force term is an attempt to simulate a turbulent flow in order to evaluate if the application can predict the flow . In fact, this force term is also called Burgerlance in analogy of a turbulent flow in a 1D domain where the Burger equation stands for Navier Stokes ones and the force term can be thought as a randomly varying pressure gradient. The first study of a stochastic force term applied to the Burger equation was described in [24]; however the model of the force term that has been used in this thesis was the Chamber one [25].

The random force term f varies in space and time so that its average in time is constant and the solution approaches a statistically asymptotically steady state .It is important remarking that choosing a high value of the constant can cause the leak of stability of the solution method.

Considering the adimensional form of the Burger equation

$$u_t + uu_x - \frac{1}{Re}u_{xx} = \psi_0\psi(x,t)\psi(x,t) \qquad (53)$$

Where

- Re is the Reynolds number $Re = \frac{L\breve{u}}{\nu}$
- L is the length of the domain (In this work $L = 1$)
- $\breve{u}$ is the dimensional and u the adimensional speed
- ψ_0 is the average constant value
- $\psi(x,t)$ is the random component of the force term

As suggested in [25] we define

$$\psi(x,t) = \Re\left(\sum_{n=N_1}^{N_2} s_n a_n(t) e^{i2\pi nx}\right) \qquad (54)$$

Where

- The formulation is a complex Fourier series
- $\Re$ is real part operator
- s_n are real coefficients
- n is the wave number
- N_1 and N_2 are the lowest and the highest wave number forced
- a_n are the random complex coefficients

The random coefficients are chosen so that

$$a_n(t_{k+1}) = e^{-\Delta t/T_n}\left[a_n(t_k) + c_{n(k)}\right] \qquad (55)$$

Where

- $c_{n(k)}$ is a complex number obtained from two consecutive numbers in a random sequence thanks to random number generator
- $T_n = \frac{1}{16\nu\pi^2}$ is the time constant which depends on ν. It was chosen to match the viscous timescales of each mode of the solution in order to maximize the cross correlation between u and ψ
- Δt is the time step of the numerical simulation

In this way, they are uniformly distributed between -1 and 1 and they were mutually uncorrelated and stationary over long time.

The minimum number of forced modes tested was a single mode which is proportional to the GAB force term, while the maximum were 40 modes (from $\mathrm{n} = 1$ to $\mathrm{n} = 40$). If it is decided to force all the 40 modes available, 80 random numbers are required from the

generator for each time step. The resultant forcing term corresponded to a simple band limited white noise-process with 40 modes.

The spectrum of the forcing term is determined by the real coefficient s_n. In order to obtain the average of ψequal to 1and the desired spectrum $s_n = \left(\frac{3}{40T_n}\right)^{1/2}$.

Moreover they satisfy the condition

$$\frac{1}{3}\sum_{n=N_1}^{N_2} {s_n}^2 T_n = 1 \qquad (56)$$

Where each individual term of the sum is one of the value of the special spectrum of ψ

5.2 Subgrid Models

We are trying to solve the problem

Find $\bar{u} \in \bar{V}$ so that

- $B(\bar{w}, \bar{u}) + M(\bar{w}, \bar{u}, c, f, h) = (\bar{w}, f)$
- $\bar{u}(0,t) = \bar{u}(L,t) = 0$ (L is the end of the domain t is time)
- $\bar{u}(x,0) = 0$
- $B(\bar{w}, \bar{u}) = (\bar{w}, u_t) - (\bar{w}_x, (\bar{u})^2) + (\bar{w}_x, \nu \bar{u}_x)$
- $M(\bar{w}, \bar{u}, c, f, h) = -2(\bar{w}_x, \tau R\bar{u})$

All the procedures that occur to solve the problem have been already described, except from the definition of τ as a function of the vector of coefficients c. it will be done in this paragraph.

Since we have modeled

$$u' = \tau R(\bar{u})$$

If the residual is small enough it is that

$$u' = \int_{\Omega} -\mathrm{g}'(L\bar{u} - f)\mathrm{d}\Omega = \mathrm{M}'(L\bar{u} - f) = G\, R(\bar{u})$$

So our model would be exact if we choose τ such that it is equal to the Green operator

$$\tau = G \qquad (57)$$

However G has not been evaluated except from the advection-diffusion equation so we need to approximate it. In [26] Shakib describes a general procedure for τ for the weighted residual formulation of the Navier Stokes equation where τ is a symmetric semidefinite

positive matrix obtained solving several eigenvalue problems. Then he compared with the definition of τ presented by Huges and Mallet in [27] where they evaluated τ for a pure advection problem and then adjusted the model for the presence of diffusion. Shakib continued his work analyzing a 1D steady advection diffusion problem. In this case, like in this thesis, τ is a scalar because only one equation is analyzed.

He wrote that if piecewise linear finite elements are used, τ can be evaluated such that the H^1 seminorm of the solution error $u - u^h$ is minimized. This condition is equivalent to having a nodal exact solution. So he defined τ so that

$$\tau = \frac{h}{2|\alpha|}\xi(\alpha) \qquad (58)$$

Where

- $\alpha = \frac{h|a|}{2v}$ the Peclet number
- a is the advective speed
- v is the viscosity
- h is the element mesh size
- $\xi(\alpha)$is the so called diffusion corrector factor which is exact if $\xi(\alpha) = \cot(\alpha) - \frac{1}{\alpha}$

Since we are looking for a numerically formulation of τ, $\xi(\alpha)$ formulation has to be modified. Shakib found a fourth order accuracy formulation, thanks to a truncation error analysis so that

$$\xi(\alpha) = \sqrt{\frac{\alpha^2}{9 + \alpha^2}} \qquad (59)$$

Which is equivalent to

$$\tau = \left(\left(\frac{2a}{h}\right)^2 + 9\left(\frac{4v}{h^2}\right)^2\right)^{-1/2} \qquad (60)$$

His analogue unsteady counterpart will be

$$\tau = \left(\left(\frac{2}{\Delta t}\right)^2 + \left(\frac{2a}{h}\right)^2 + 9\left(\frac{4v}{h^2}\right)^2\right)^{-1/2} \qquad (61)$$

So this one can be considered as the best τ approximation, if we don't need a fifth order accuracy formulation.

In this thesis the structure of the Shakib formulation of τ is used; it has been modified only to adapt it to the Germano procedure and to the BFSG algorithm. The τ_{shakib} formulation will be

$$\tau_{shakib} = \left(\left(\frac{2}{\Delta t} \right)^2 + c_0^2 \left(\frac{u}{h} \right)^2 + 100 c_1^2 \left(\frac{v}{h^2} \right)^2 \right)^{-1/2} \qquad (62)$$

In fact

- c_0 and c_1 are the unknown coefficients that are calculated thanks to the Germano Method. We have to remark that the previous formulation is the best one for the advection diffusion equation. So the evaluation of the coefficients, that won't be the same, will improve the solution [3].
- The factor 100 is added in order to keep the BFSG algorithm stable

In this work the Shakib formulation will be compared with a linear definition of c_0 and τ

$$\tau_{lin} = c_0 h \qquad (63)$$

The comparison will be done for the ASGS and the OSS case.

Moreover the target of this thesis is to find a formulation of τ which will improve the Shakib one. So we have defined a space variant formulation which can find the best coefficients for every element of the domain. So τ won't be constant but it will be function of x.

The first formulations that will be tested

$$\tau = h \left| c_0 + c_1 \cos\frac{\pi x}{L} + c_2 \sin\frac{\pi x}{L} \right| \qquad (64)$$

called Space Variant Tau –SVT

The second one can be seen as the consecutive of the CSVT (in a Fourier sense).

$$\tau = h \left| c_0 + c_1 \cos\frac{\pi x}{L} + c_2 \sin\frac{\pi x}{L} + c_3 \cos\frac{2\pi x}{L} + c_4 \sin\frac{2\pi x}{L} \right| \qquad (65)$$

Space Variant 2 Tau SVT2

These formulation has been chosen so that both the space dependent functions are linear independent and it is possible to obtain a linear combination whose result is a global constant

We have checked that some of these formulation will be better than the linear one; then we tried to verify if, defining a Shakib space variant formulation, it can be better than the original Shakib one.

The Shakib space variant will be

$$\tau_{shakibsvt} = \left(\left(\frac{2}{\Delta t}\right)^2 + \left(h \left| c_1 \cos\frac{\pi x}{L} + c_2 \sin\frac{\pi x}{L} \right|\right)^2 \left(\frac{u}{h}\right)^2 + 100\left(h \left| c_3 \cos\frac{\pi x}{L} + c_4 \sin\frac{\pi x}{L} \right|\right)^2 \left(\frac{\upsilon}{h^2}\right)^2 \right)^{-1/2} \quad (66)$$

$$\tau_{shakibsvt2} = \left(\left(\frac{2}{\Delta t}\right)^2 + \left(h \left| c_0 + c_1 \cos\frac{\pi x}{L} + c_2 \sin\frac{\pi x}{L} + c_3 \cos\frac{2\pi x}{L} + c_4 \sin\frac{2\pi x}{L} \right|\right)^2 \left(\frac{u}{h}\right)^2 + 100\left(h \left| c_0 + c_1 \cos\frac{\pi x}{L} + c_2 \sin\frac{\pi x}{L} + c_3 \cos\frac{2\pi x}{L} + c_4 \sin\frac{2\pi x}{L} \right|\right)^2 \left(\frac{\upsilon}{h^2}\right)^2 \right)^{-1/2} \quad (67)$$

CHAPTER 6

6. ANALYSIS

6.1 Introduction

In this chapter 2 different numerical cases are studied. The task is to solve numerically the unsteady Burger equation varying the Variational Multiscale Methods and the definitions of τ. The complete problem is to find $u \in V$ so that

$$u_t + uu_x - \nu u_{xx} = f \text{ on } \Omega$$

$$u = 0 \text{ on } \delta\Omega$$

$$u(x,0) = 0$$

Where

- $\Omega = [0,1]$ is the spatial domain
- $T = [0{,}5]$ is the temporary domain
- $\nu = 0{,}001953$ is the viscosity so the Reynolds number is 512.
- f is the force term. In the first paragraph the Gabriel forcing term has been used, in the second the Random formulation of Chamber.

However, the problem solved in this thesis is the discrete formulation of the previous one which consist in finding $u^h \in V^h$ so that

- $B(w^h, u^h) + M(w^h, u^h, c, f, h) = (w^h, f)$
- $u^h(0,t) = u^h(1,t) = 0$
- $u^h(x,0) = 0$
- $B(w^h, u^h) = (w^h, {u^h}_t) - (w^h{}_x, (u^h)^2) + (w^h{}_x, \nu u^h{}_x)$
- $M(w^h, u^h, c, f, h) = -2(w^h{}_x, u' u^h) = -2(w^h{}_x, \tau R u^h)$
- $R = (w^h, {u^h}_t) - (w^h{}_x, (u^h)^2) - (w^h, f)$

In the discrete case, the domain is splitted into 64 equal elements and the time step is $\Delta t = 0{,}05$. Linear finite elements are used. The results obtained by LES are compared with those computed by DNS. In the DNS a 1024 element mesh. In fact if we evaluate the dimension of the Kolmogorov scale it is $9 \cdot 10^{-3}$. The DNS element dimension is $1 \cdot 10^{-3}$ and then it is acceptable. $\Delta t = 0{,}005$ is the DNS time step in order to resolve even the smallest structures. The Germano initial coefficients are all equal to 2,0.

The DNS was computed using a code which solve the burger equation using a finite element discretization without splitting u into the coarse scales and the fine scales.

In the following chapters, first the differences between a solution and another one are underlined, then all of the solutions are compared in order to find the best one. For every case, two pictures are presented: a first one related to the global solution and the second which is zoomed in the part of the plot where is bigger the difference between the solutions. Plots are taken at the last time step.

6.2 Gabriel Force Term

6.2.1 OSS compared with ASGS

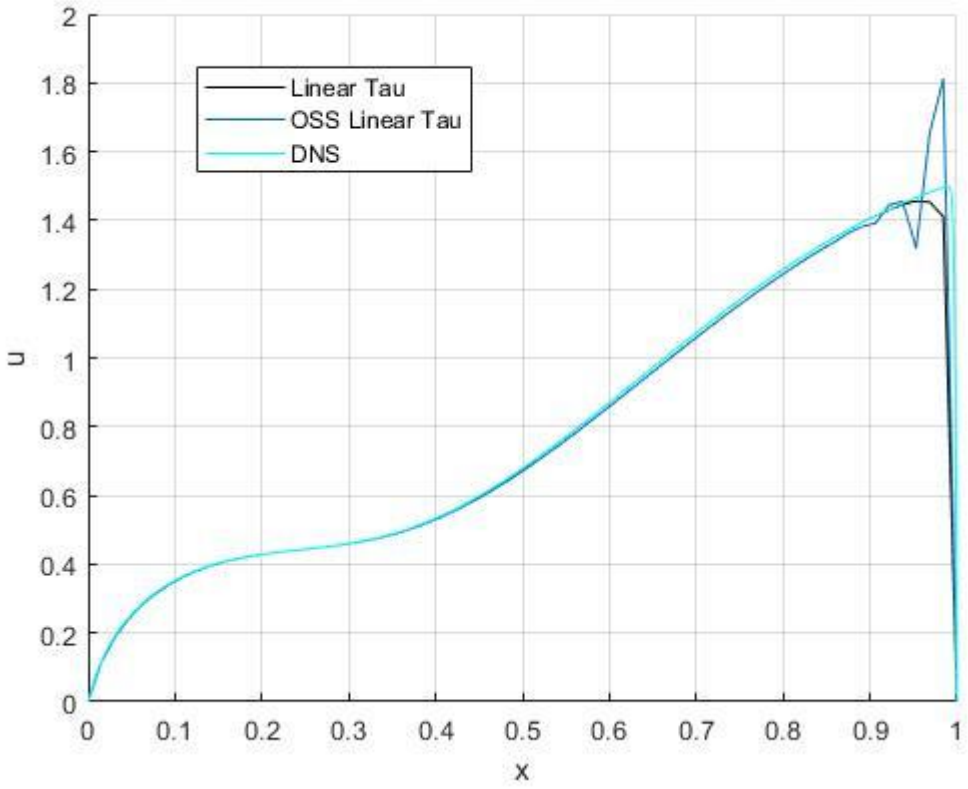


Figure 3 Comparison between OSS and ASGS solution

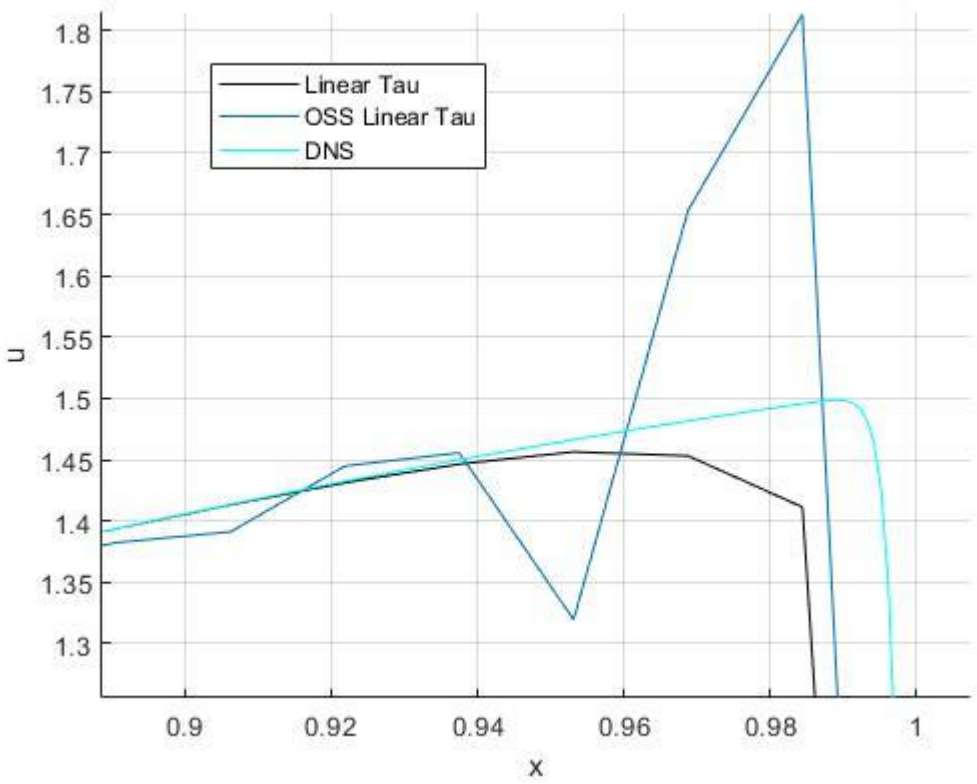


Figure 4 Comparison between OSS and ASGS solution detail

The first comparison is between the ASGS and the OSS solutions. Both cases have been obtained using the linear formulation of τ.

It is possible to observe that

- The OSS solution results more fluctuating than the ASGS one. Fluctuations can be observed not only close to the step but they propagates in the previous 5 integration points. This fact is due to the property of OSS (described in[26]) of introducing less dissipation. It can be useful because the OSS method can help to follow better the peaks however this can lead, like in this case, to oscillatory behaviors.
- The OSS is locally closer to the exact one, except from the area before the step; for example it is possible to notice that $\left(\frac{\partial u}{\partial x}\right)_{x=1}$ of the OSS case is closer to the DNS then the ASGS one
- The OSS results much more precise if the exact solution is smooth. . In fact, as the following picture can prove, for every time step the linear case L_2 error is bigger than the OSS one.

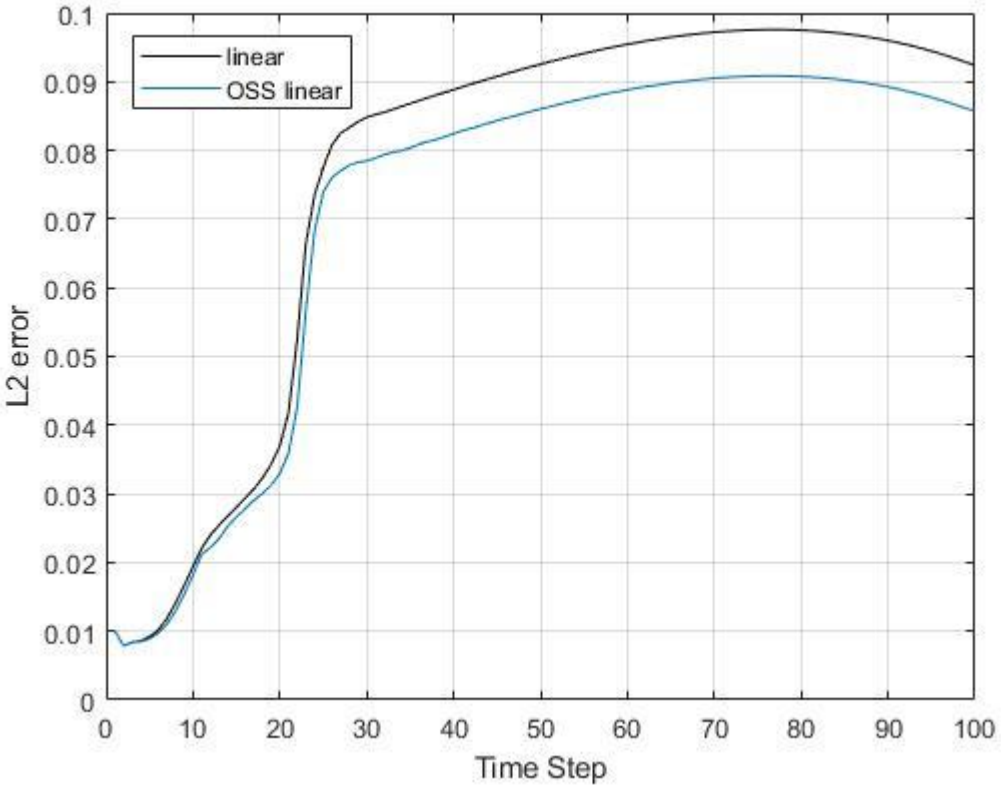


Figure 5 L2 error comparison between OSS and ASGS solution

We can conclude that in this case the ASGS model results better, even if the OSS is more precise because it doesn't present oscillation that are not present in the exact solution.

6.2.2 Shakib compared with Linear tau

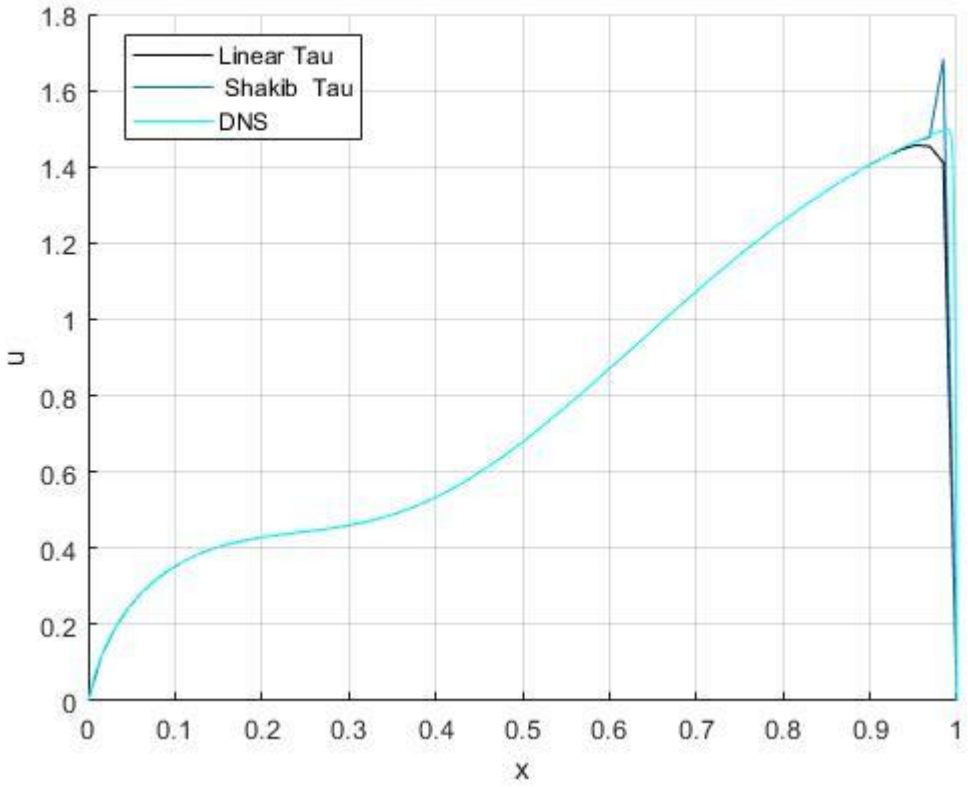


Figure 6 Comparison between Shakib and Linear Tau solution

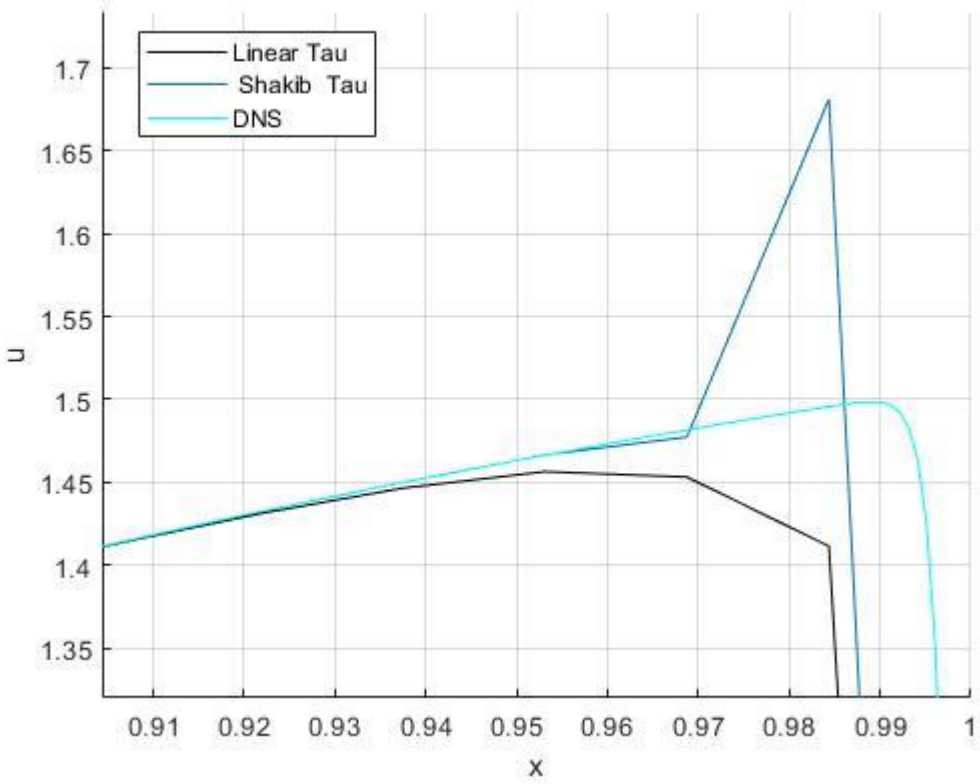


Figure 7 Comparison between Shakib and Linear Tau solution detail

The second comparison is made using the plot of the linear τ solution and the Shakib one. This comparison has already been described in [1]. In any case we can underline that

- The Shakib solution over dimension the step presented by the DNS solution.
- The linear solution is more numerically dumped because it introduces more dissipation. In fact, the contribute of u' of the linear case is generally bigger than the Shakib one. It can be seen in the following plot:

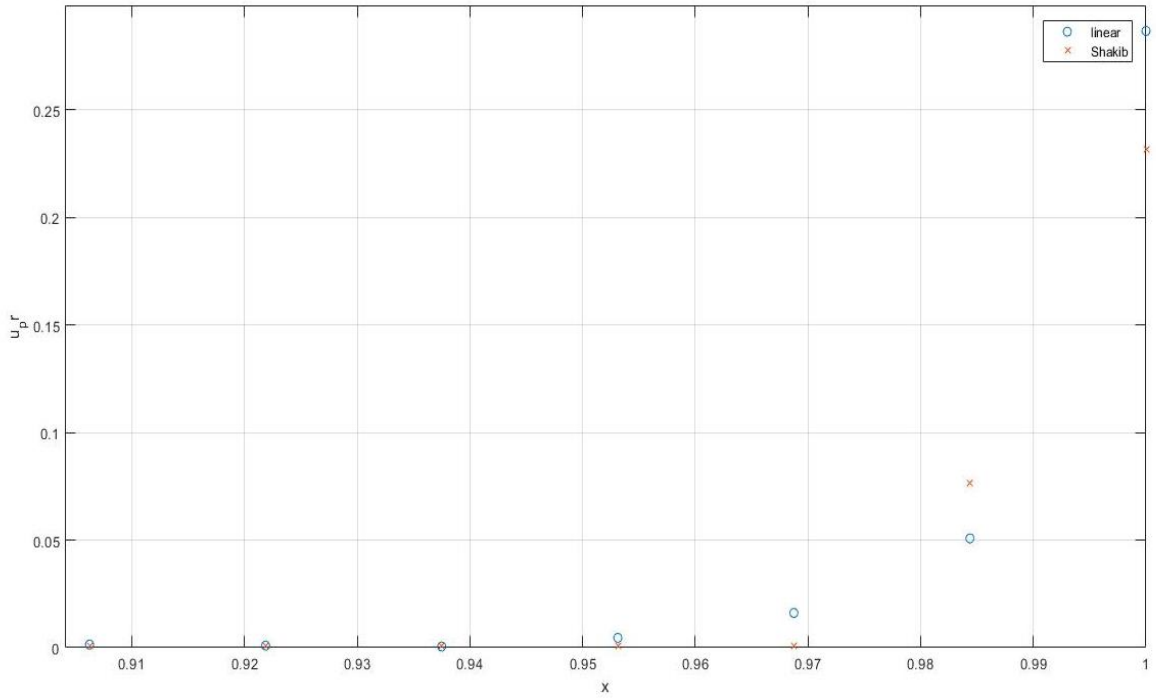


Figure 8 u' comparison between Shakib and Linear Tau solution

- The Shakib solution is both globally and locally more accurate. To prove it, it can be useful the plot of the L_2 error as function of time. The Shakib curve is always lower than the linear one

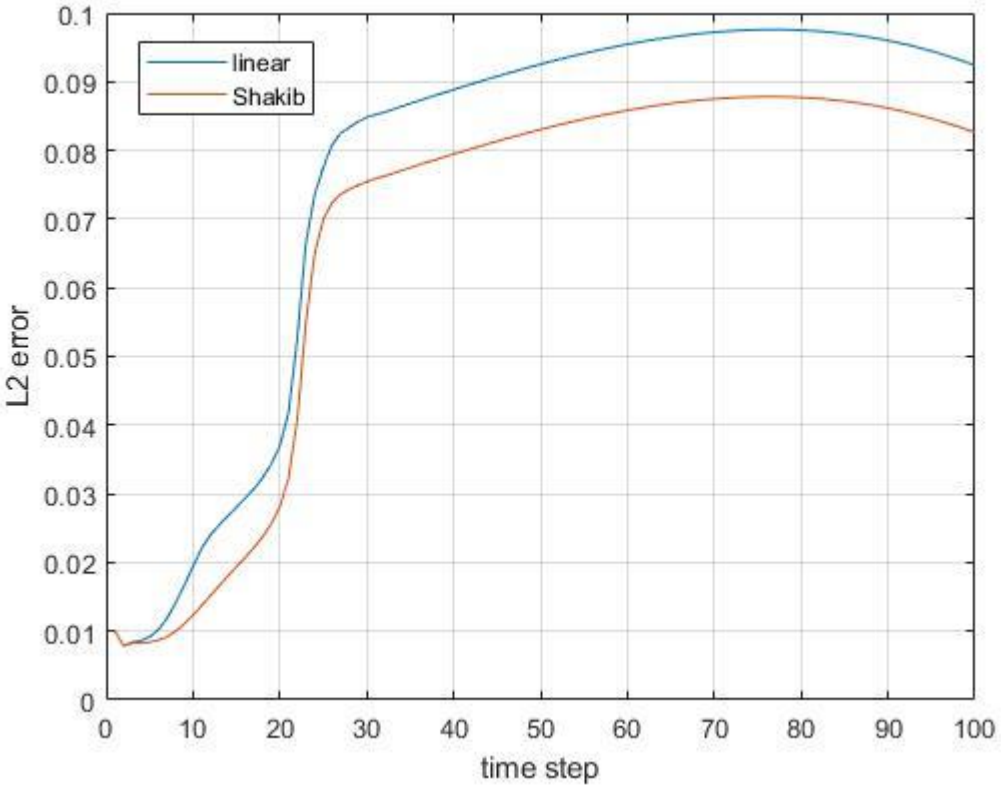


Figure 9 L2 error comparison between Shakib and Linear Tau solution

The Shakib solution is better than the linear one; the only defect is that it over dimension the value of u near to the step.

6.2.3 Shakib compared with OSS Shakib

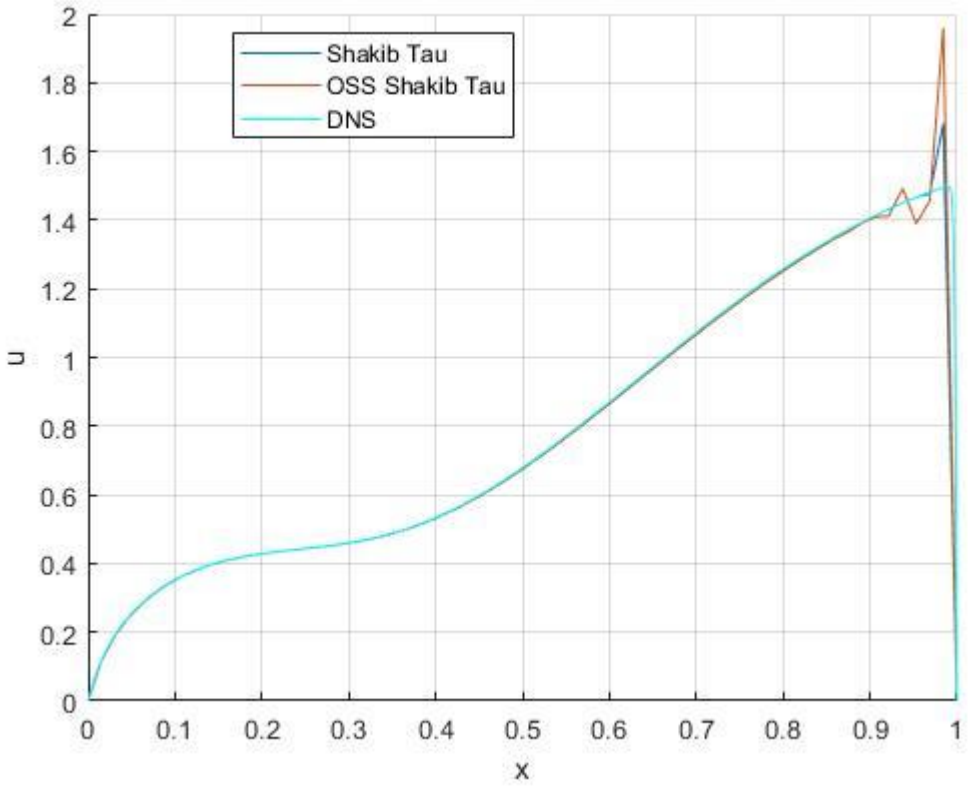


Figure 10 Comparison between Shakib and OSS Shakib solution

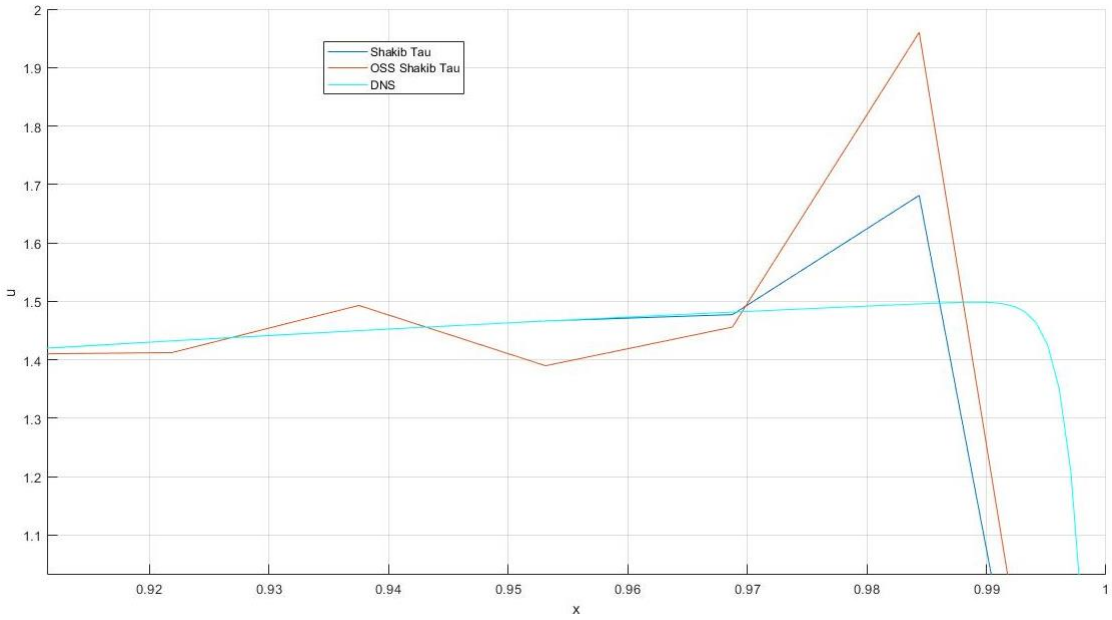


Figure 11 Comparison between Shakib and OSS Shakib solution deatil

In this case we are evaluating the Shakib solution and the Shakib solution obtained using the OSS model. Main differences between the solutions are that

- The OSS model induces oscillations to the Shakib solution as it happened for the linear case. Moreover it also over dimensions the value peak of Shakib solution near to step of the DNS one. So mixing the Shakib formulation of τ and the OSS model the defects of both them are emphasized.
- The Shakib OSS solution is only closer to the DNS near to the boundaries. Far from them, the Shakib solution is more precise.
- The ASGS Shakib solution is generally lightly more accurate as it can be seen in t the L_2 error plot

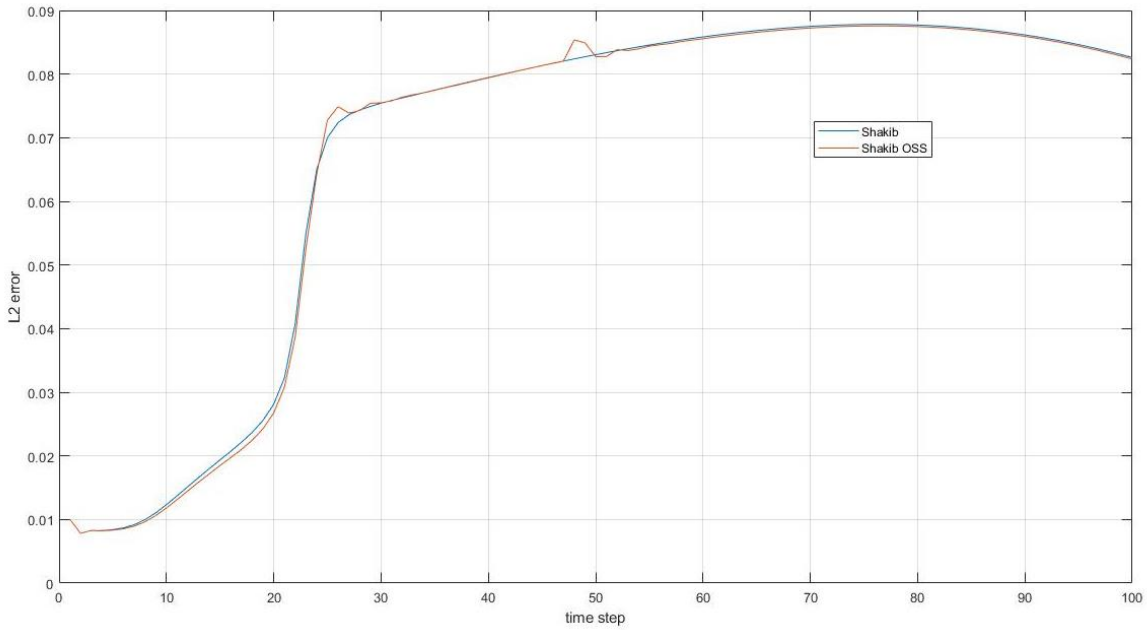


Figure 12 L2 error comparison between Shakib and OSS Shakib solution

The OSS model formulation creates too many oscillation, solving this problem, because it is less dissipative then the ASGS one. This fact was proved in the literature by Codina [26] and is showed by both the linear and the Shakib case. As a consequence, in the space variant analysis we will focus on the ASGS analysis since the OSS solutions can't be the best one.

6.2.4 OSS compared with OSS Shakib

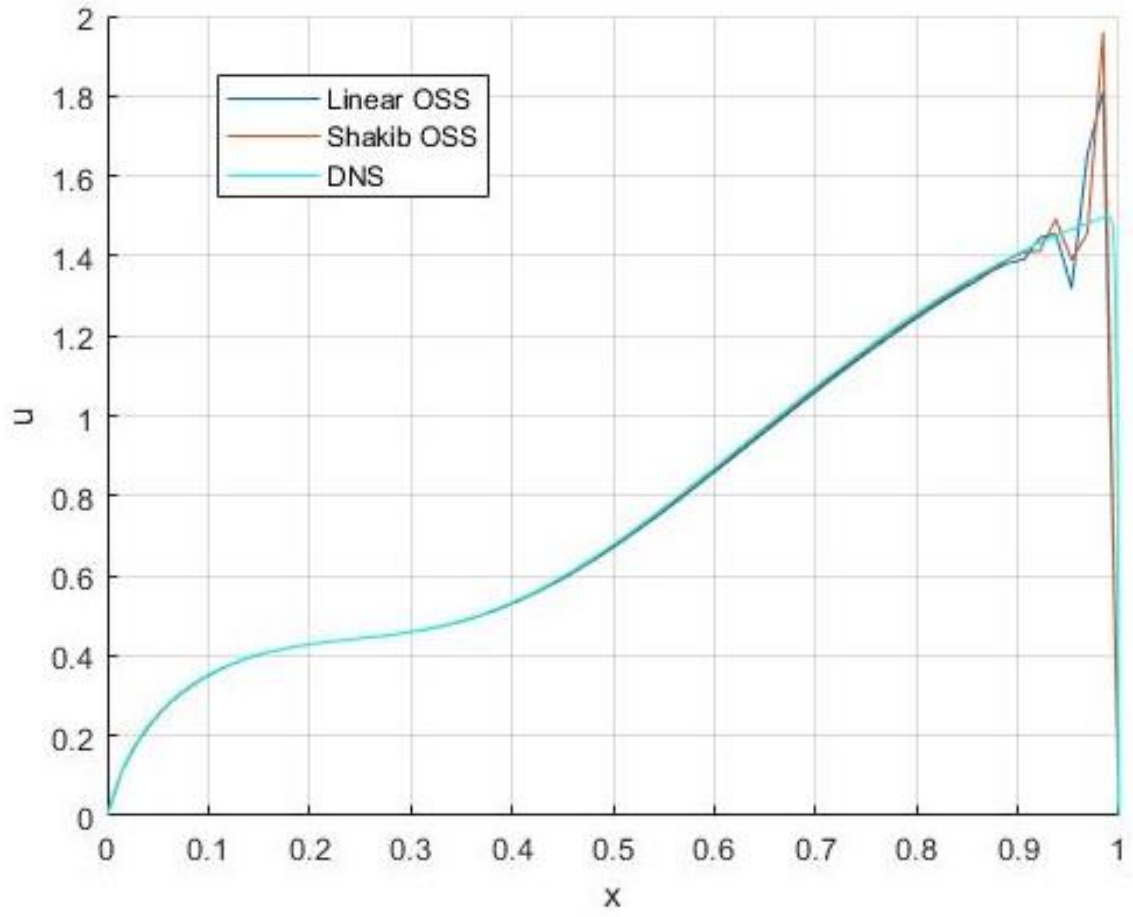


Figure 13 Comparison between OSS and OSS Shakib solution

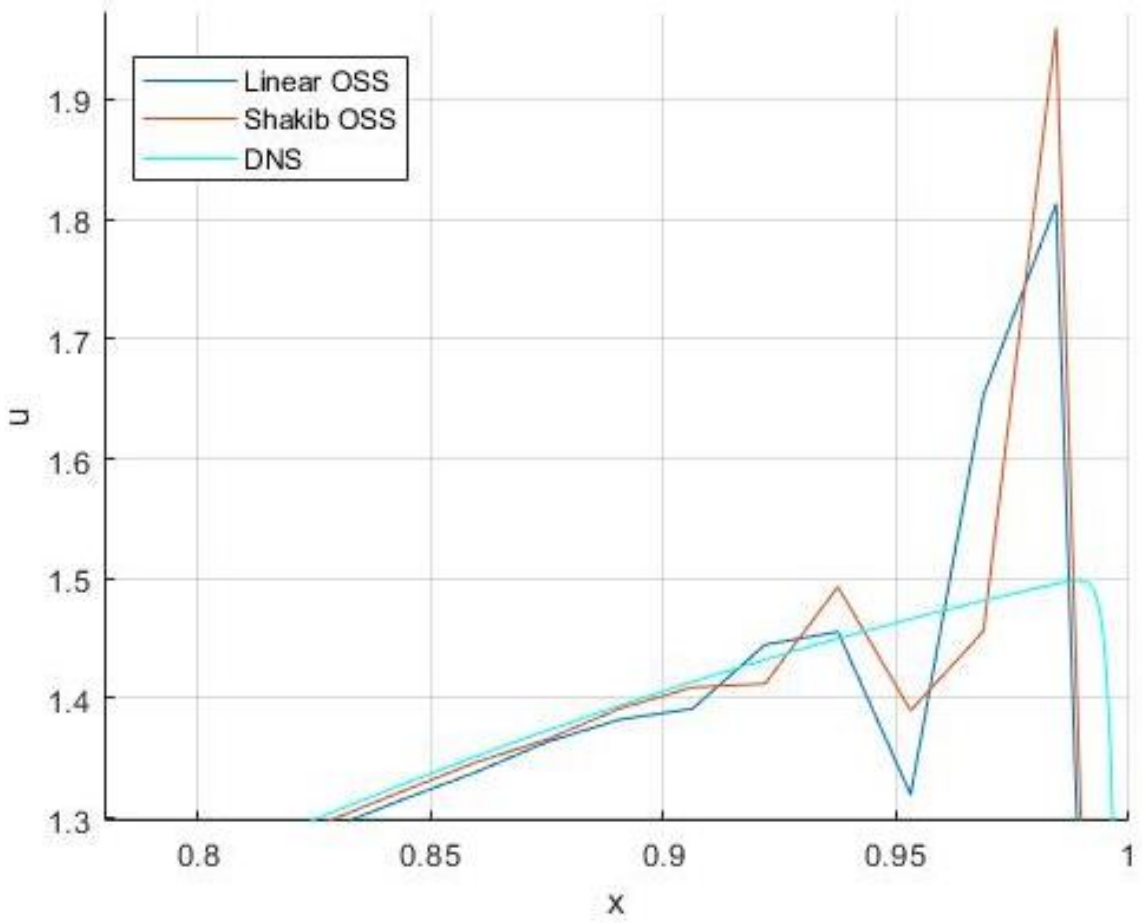


Figure 14 Comparison between OSS and OSS Shakib solution detail

In this paragraph it is demonstrated that the Shakib formulation "behaves" better than the linear one using the OSS model too. In fact it is possible to observe that

- Except for the value of u immediately before the step of the DNS, the Shakib case presents smaller oscillations
- The L_2 error of the Shakib OSS solution is lower than the linear OSS case for every time step

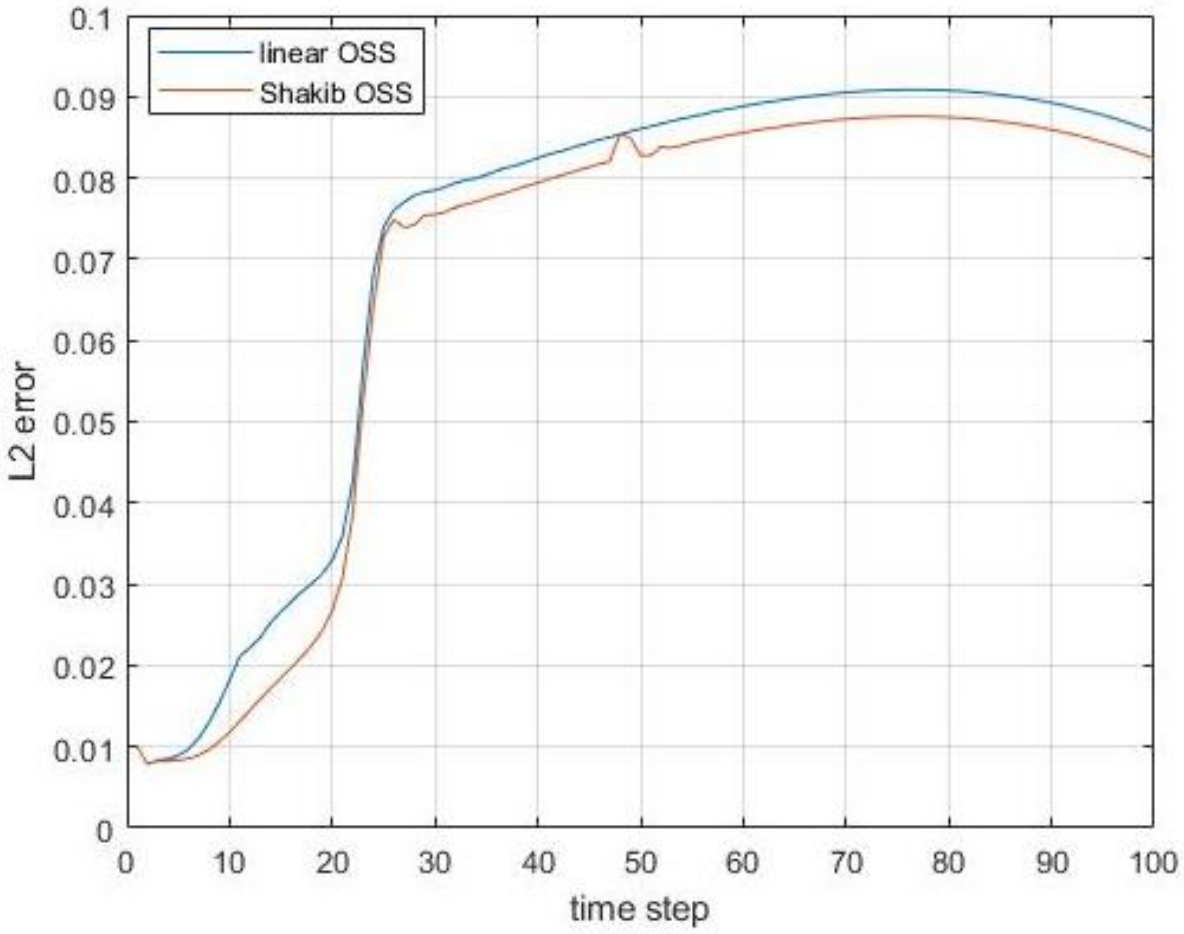


Figure 15 L2 error comparison between OSS and OSS Shakib solution

6.2.5 Linear compared with Space Variant and with Space Variant 2

In this paragraph the linear and the space variant formulation solutions are compared. The big difference between them is that, for the space variant ones, the code evaluate τ for every integration point, solving the Germano system for all of them. This can improve the resolution of the fine scales since in this way it is possible to introduce dissipation where it is needed. In fact with a higher order Fourier τ is possible to approximate better big gradients. We will check it in the following lines.

It is important to remark that we have not considered the OSS case because of the undesired oscillations and because both the OSS space variant solutions need 256 integration point to converge. This is the reason why, even if they may be more precise, they need a computational effort that is at least 4 time bigger then the ASGS ones.

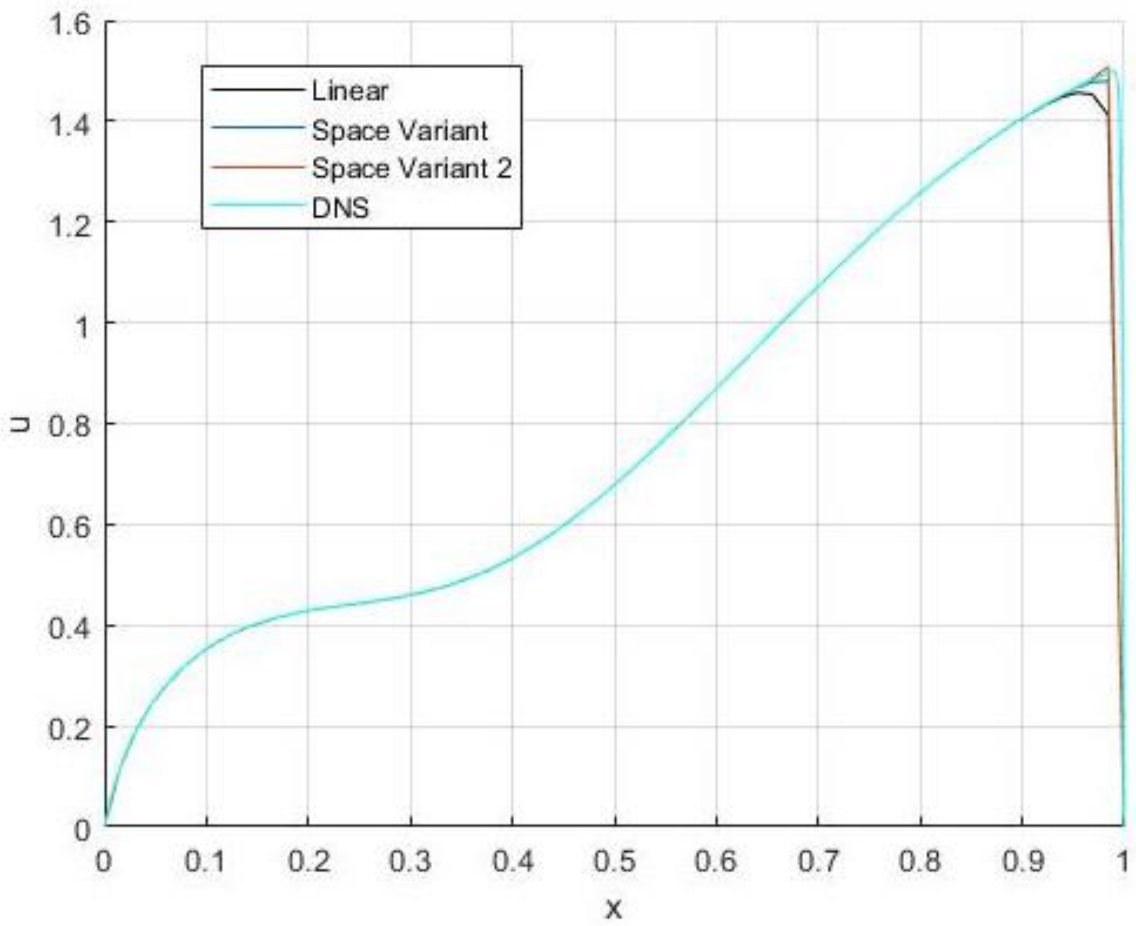


Figure 16 Comparison between Linear, Space Variant and Space Variant 2 solution

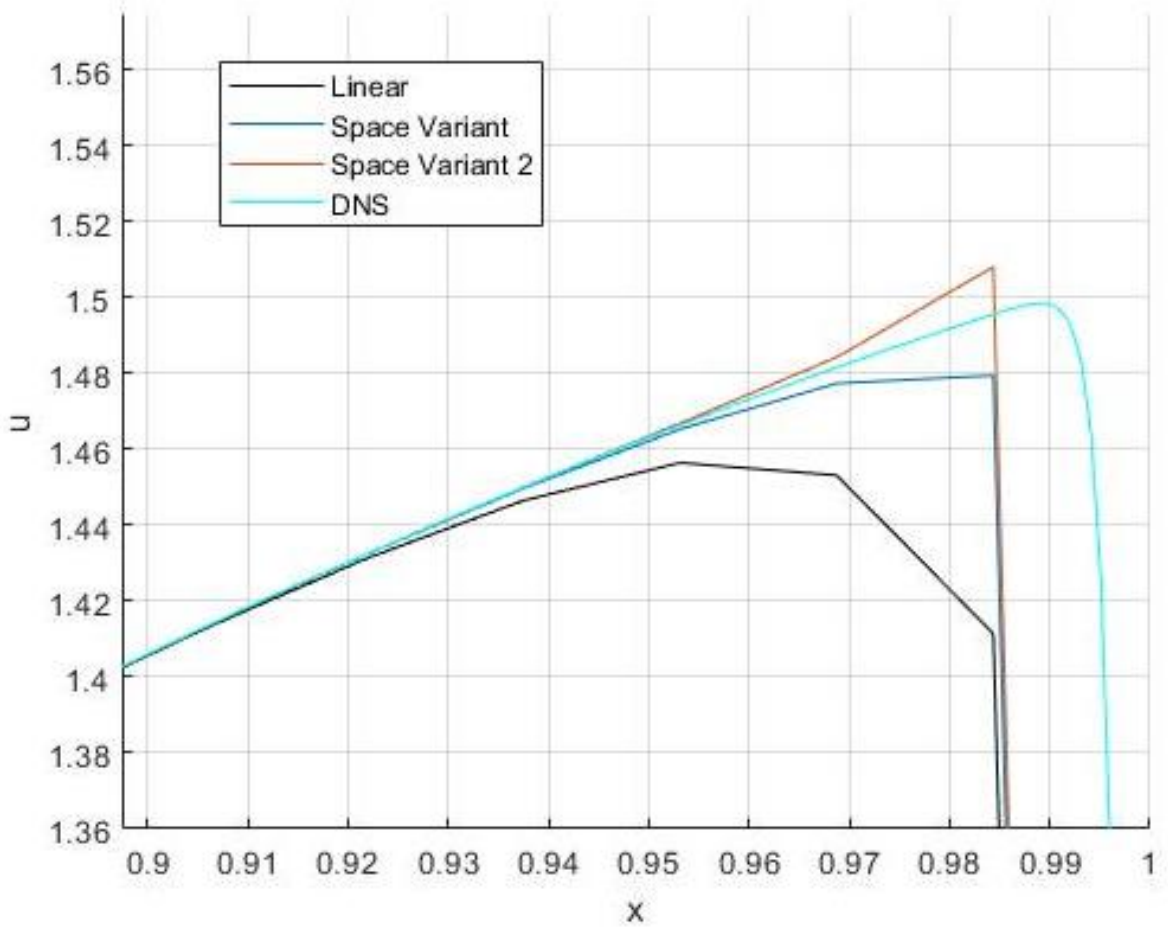


Figure 17 Comparison between Linear, Space Variant and Space Variant 2 solution detail

We can make a comparison between the 2 space variant solutions, the linear and the DNS. So

- The space variant solution is less dissipative than the linear one and it approximates better the DNS.
- The space variant 2 improves the space variant solution because it can follow the step of the DNS better as supposed by the theory.
- The space variant and the space variant 2 are locally very similar. The value of $\left(\frac{\partial u}{\partial x}\right)_{x=1}$ of both the solutions are really almost the same.
- Increasing the terms of the Fourier expansion of the expression of τ, the solution becomes more precise: in the following plot, the Space Variant 2 solution characterized by the lowest L_2 error for all the time step of the simulation.

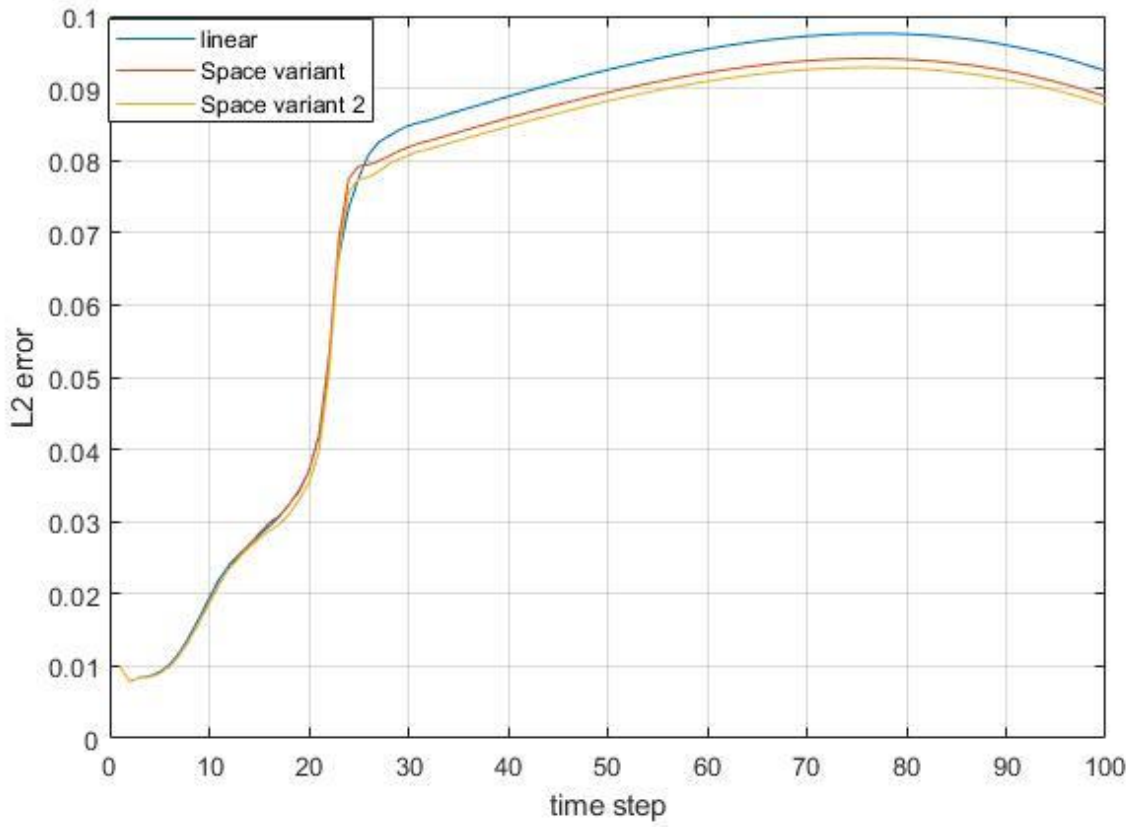


Figure 18 L2 error comparison between Linear, Space Variant and Space Variant 2 solution

- It is interesting to plot the trend of u' as function of x. We can notice that it is almost 0 for every point of the domain because the residual of the coarse scales is close to 0. But it differs from that value near to the step

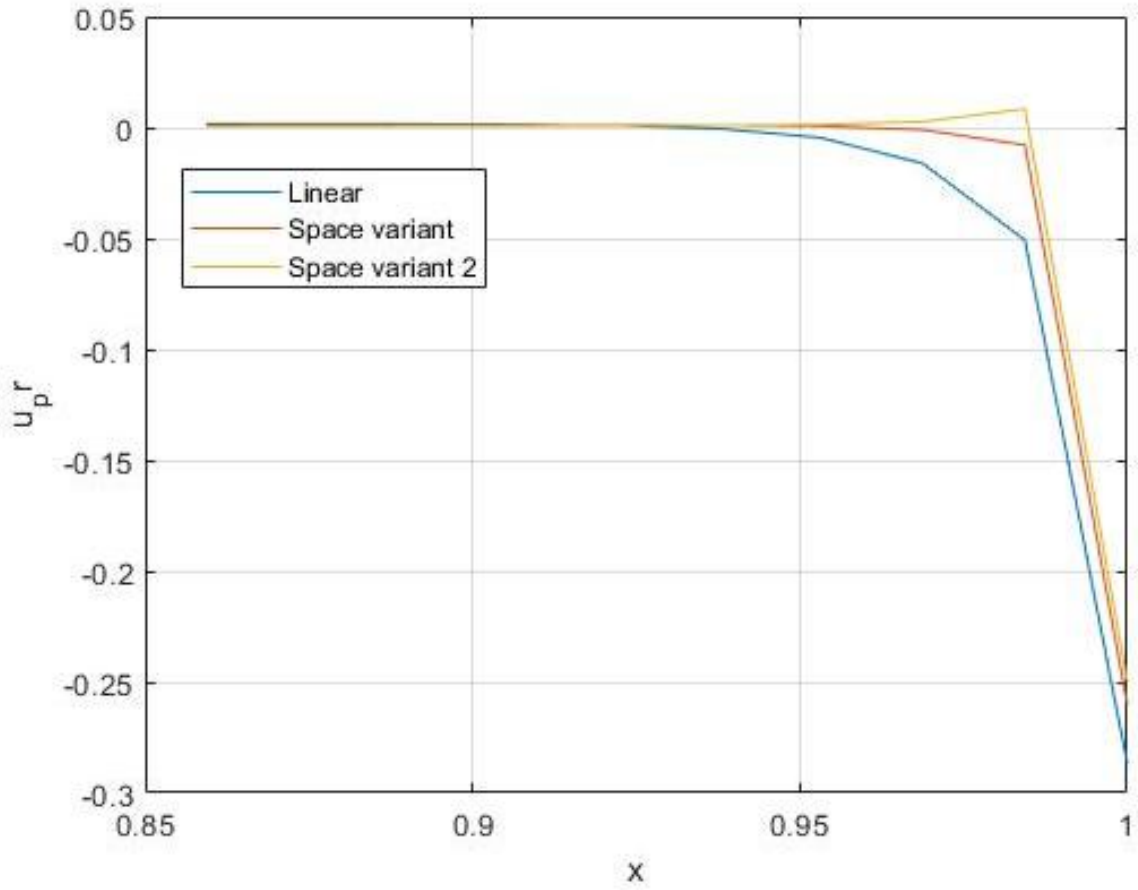


Figure 19 u' comparison between Linear, Space Variant and Space Variant 2 solution

- Finally it is plotted the tau as function of the integration point. We can recognize the constant, the first and the second levels of the Fourier series.

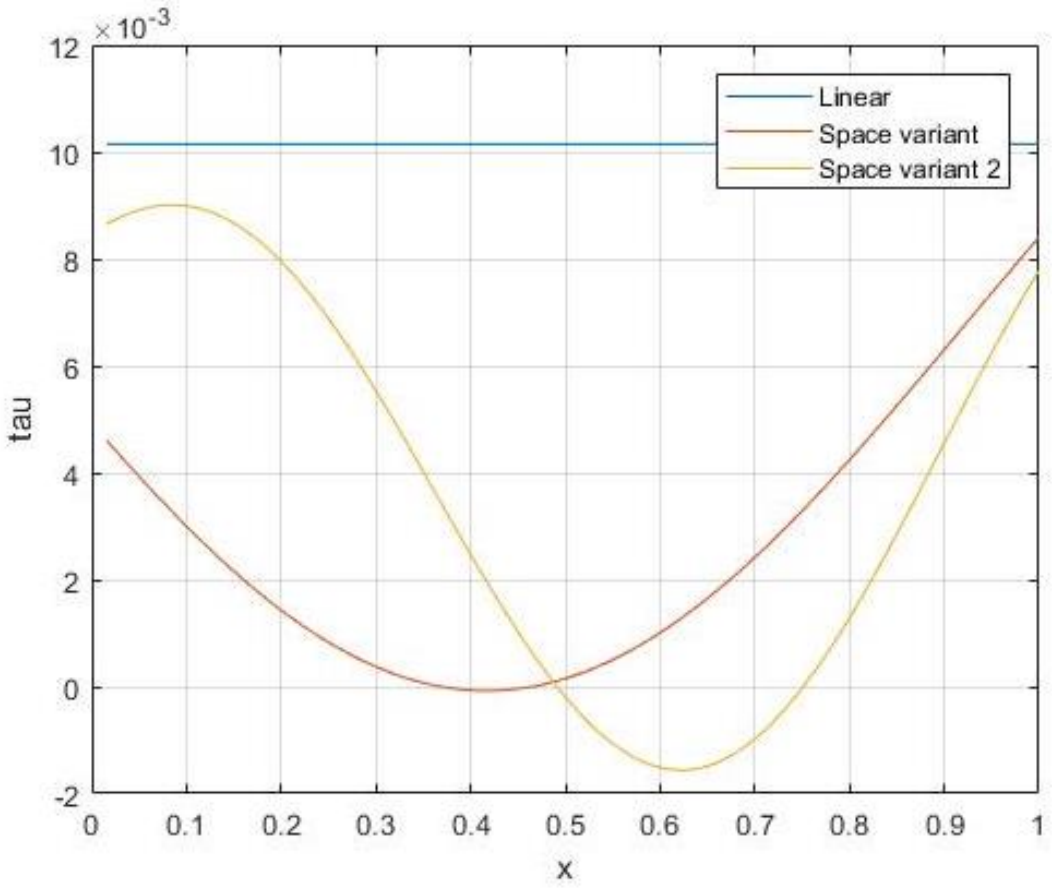
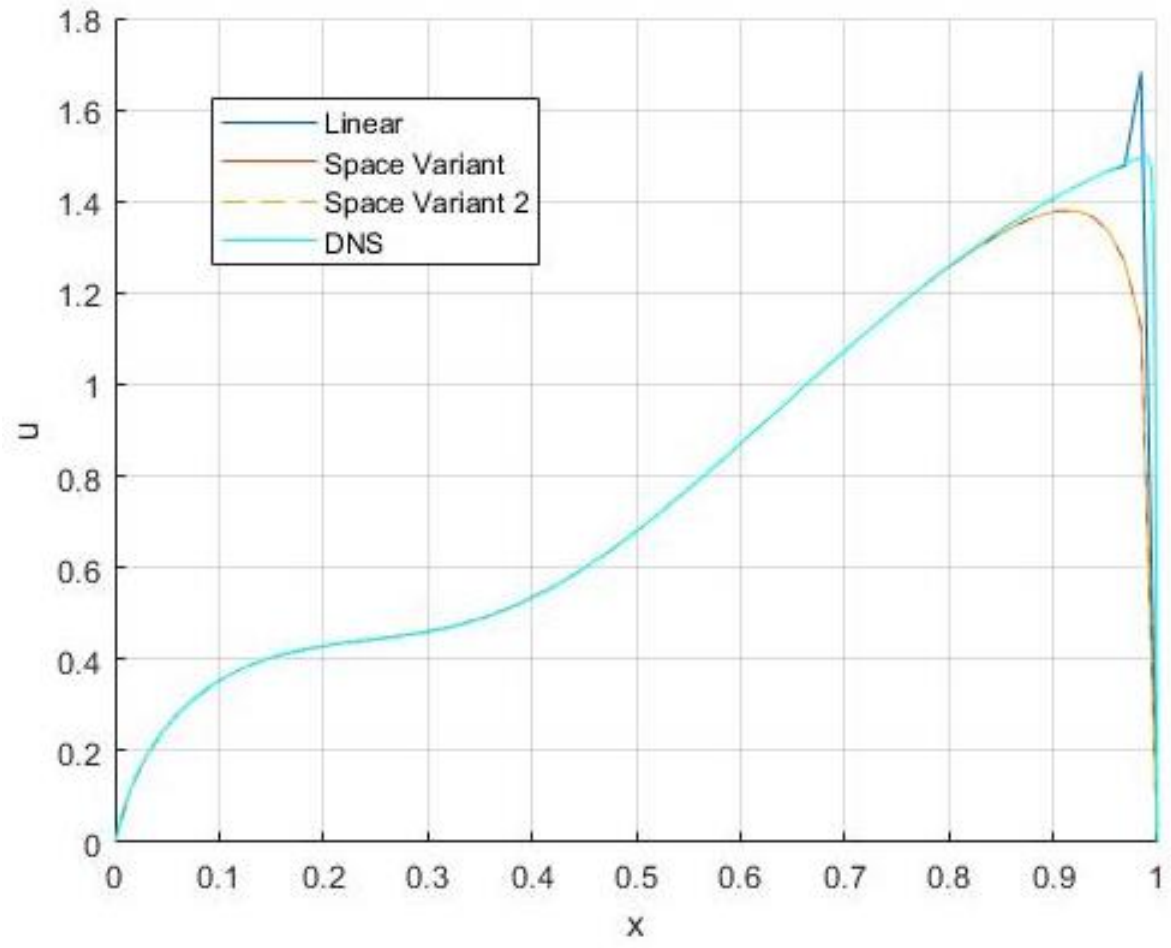


Figure 20 Tau comparison between Linear, Space Variant and Space Variant 2 solution

Thanks to the space formulation of τ we can improve the VGM solution of the burger equation. However, adding new terms, seems it does not improve so much the solution as the first space variant do compared by the linear one. Further research may prove it.

6.2.6 Shakib compared with Shakib Space Variant and with Shakib Space Variant 2

In this chapter we try to improve the Shakib formulation using a space variant τ. As it is proved previously, it is possible to improve a constant formulation using a space variant one; so, we try to do the same using a formulation with the same structure of the Shakib one. To do that, we will substitute the coefficients c_0 and c_1 with the spatial variant formulations of τ. Respectively the Shakib space variant and the Shakib space variant 2 formulations are the Shakib extension of the space variant and the space variant 2



.

Figure 21 Comparison between Shakib, Shakib Space Variant and Shakib Space Variant 2 solution

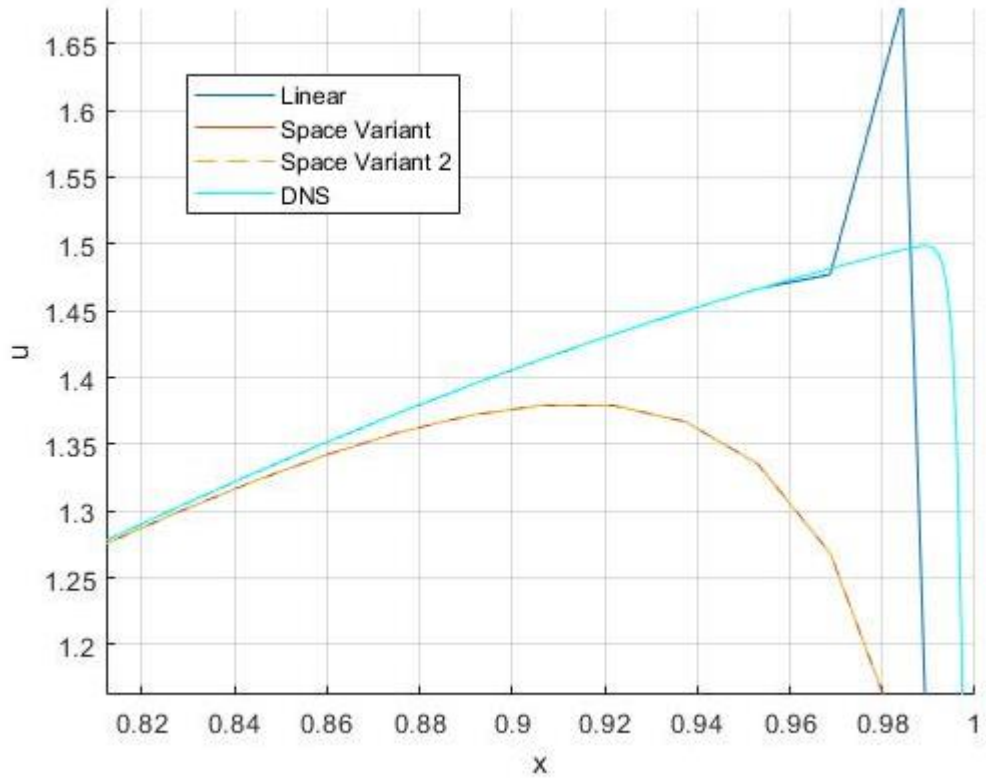


Figure 22 Comparison between Shakib, Shakib Space Variant and Shakib Space Variant 2 solution detail

It is clear that the trend of the Space variant Shakib solutions is completely unexpected. Let's point out main observations that are possible to make looking at these plots

- Both the space variant solutions are so much diffusive that they look very inaccurate. As a prove of that we can show the L_2 error trend as function of time. For every t the error of the traditional Shakib solution is smaller. The Shakib space variant formulation, for the biggest part of the time steps, presents an error which is 50% bigger than the constant is space Shakib formulation.

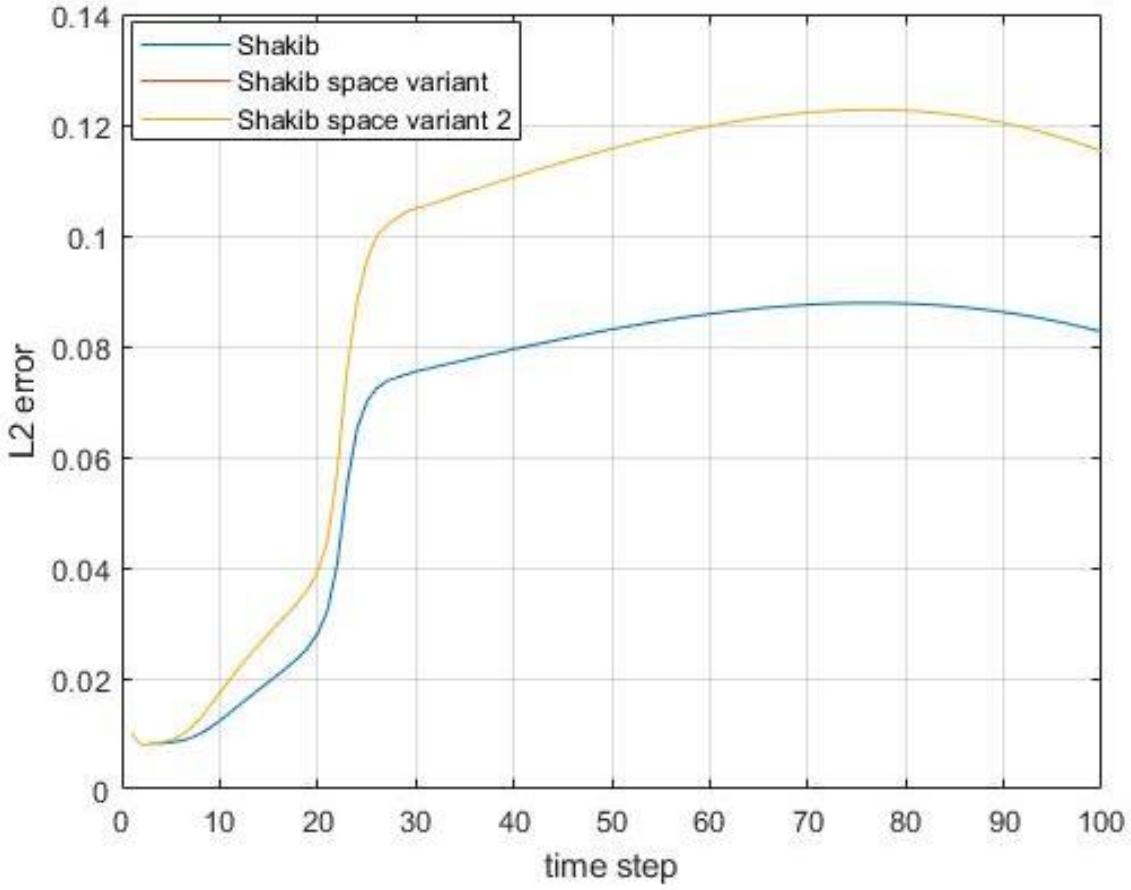


Figure 23 L2 error comparison between Shakib, Shakib Space Variant and Shakib Space Variant 2 solution

- The second unexpected result is that the Shakib space variant and the Shakib space variant solution look the same solution. In fact they present a very similar τ: the difference is about 10^{-4}. It is also difficult to recognize the $\sin(x)$ and the $\sin(2x)$ form of the τ because the plot is very stretched.

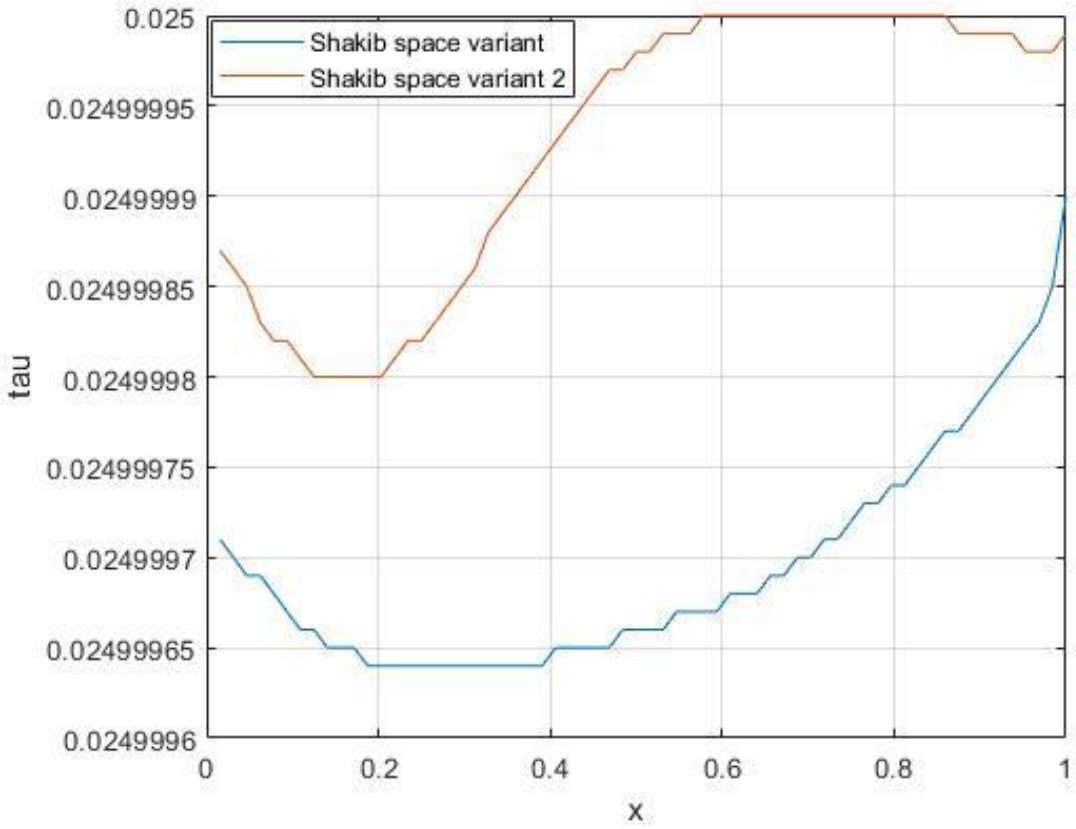


Figure 24 Tau comparison between Shakib, Shakib Space Variant and Shakib Space Variant 2 solution

- The high numerical dissipation introduced by these space variant formulation is proved by the high mean value of τ which is very much bigger than linear of the previous chapter.

We can conclude that the space variant Shakib formulations are not good to achieve the best general solution of the problem. Moreover, the space variant 2 increases a lot the computational effort obtaining a solution which is almost the same of the formulation with 4 less coefficients. It may be possible to improve these results changing the space variant formulation in order to adapt it better to the problem (and maybe in order to reduce the number of the coefficients). A possible explanation of this results can be the fact that, since the biggest part of the coefficients remains for all the time step close to the initial value of 2, the BFSG model doesn't find a big improvement of the solution changing them. So, they introduce almost the same dissipation for all the time steps.

6.2.7 General Comparison

Finally we want to choose the best solution among the ones that we have analyzed. It would neither present undesired fluctuations nor be too much dissipative.

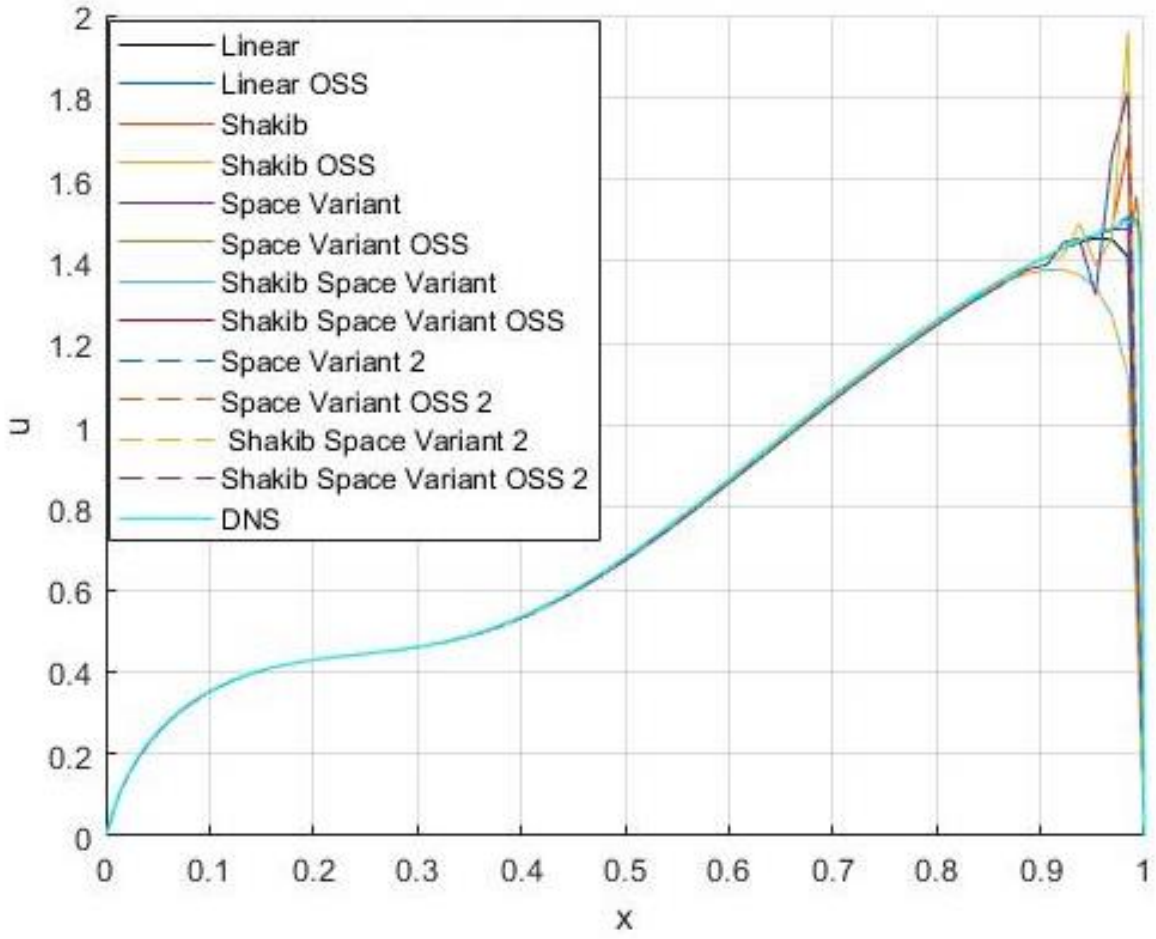


Figure 25 General comparison between all the formulations

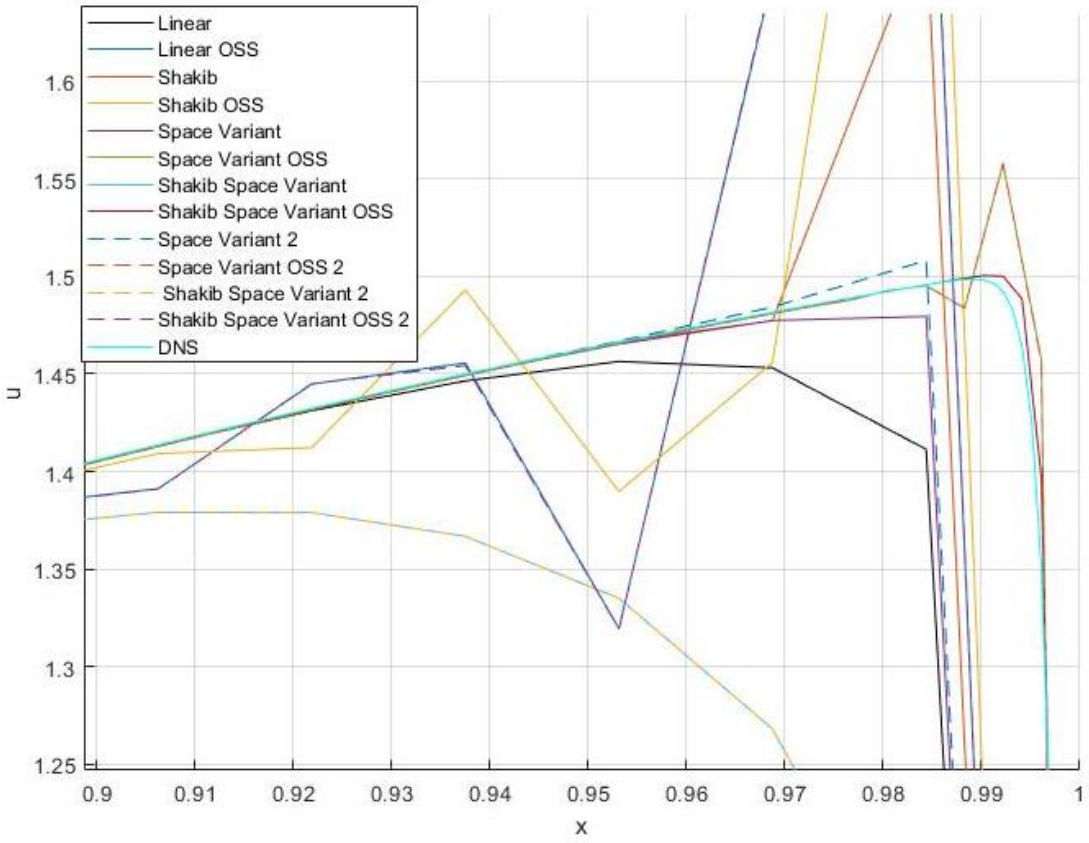


Figure 26 General comparison between all the formulations detail

We can notice that

- The best solution overall is the Shakib Space variant OSS solution which is the most precise both locally and generally. The problem is that it is not comparable with the other solutions because it has been obtained with a 256 elements mesh and so it requires more computational effort. It is also important to underline that it differs from the other OSS solutions because it doesn't present oscillations
- There are 2 more couple of solution which presents almost the same plot.The Space variant OSS is very similar to the Space variant OSS 2 and the Shakib space variant 2 is very similar to the linear OSS.
- The best solution (considering only the solutions obtained using a mesh of 64 elements) is the space variant. It represents an optimal compromise between high accuracy and computational effort.
- Using the most complicate models, is not a garancy of obtaining the best results. A reason to explain this fact can be that the more operations are done by the calculator the more numerical error propagates.
- Even if the space variant Shakib formulations are not so good, we have improved the traditional Shakib solution using the Space variant 2 (as we can see in the following plot) that is the main task of this thesis.

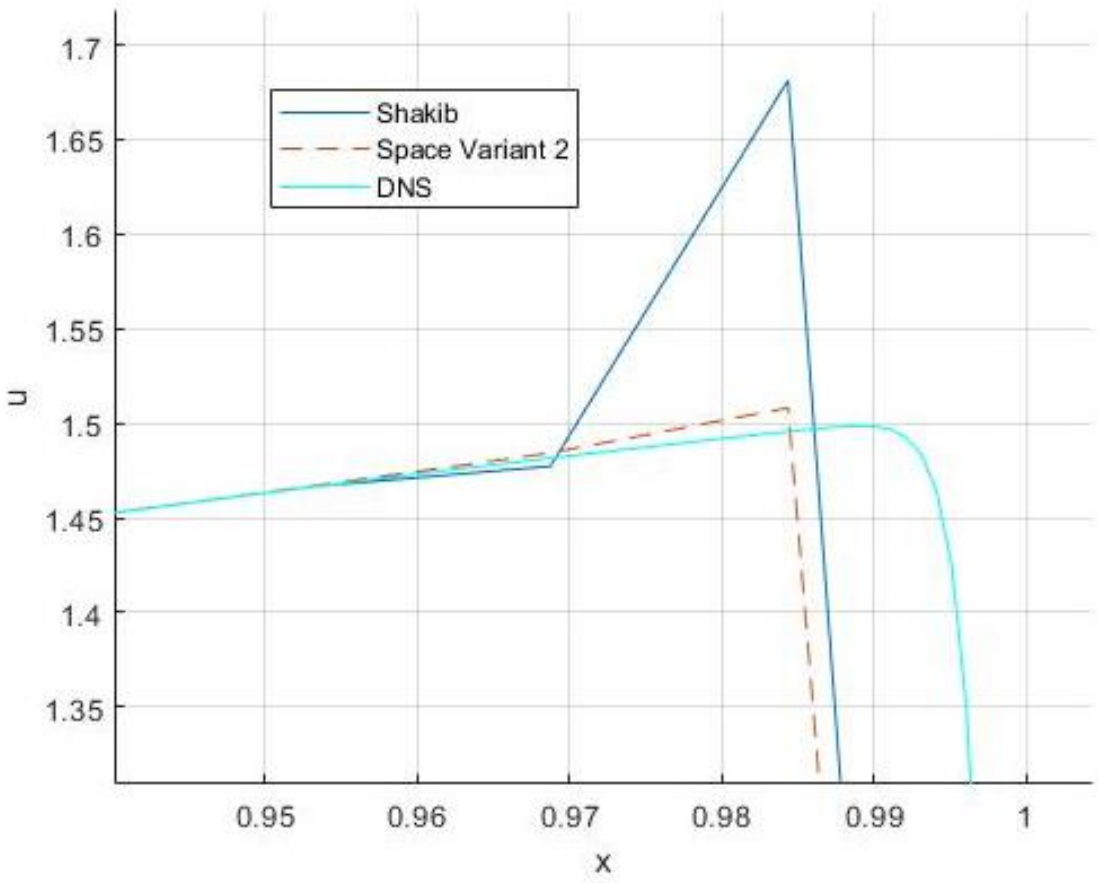


Figure 27 Comparison between Shakib and Space Variant 2 solution detail

6.3 Stochastic force term

The study of the Chamber random force term had been developed using the same formulations and procedure used in the study of the Gabriel force term. It is chosen a number of 5 modes of the random force term. However the number of elements of the mesh have been changed in order to make the solution converge and consequently the number of the time step have been increased.

So it has been necessary to use a 256 element mesh and a $\Delta t = 3{,}3 \cdot 10^{-3}$. However, using so fine mesh, the element length is smaller than the Kolmogorov scale (the dimension of the elements is $4 \cdot 10^{-3}$) so the resolution can be considered undue to a LES. Another difference is the simulation time which has been set to 1s.

6.3.1 OSS compared with ASGS

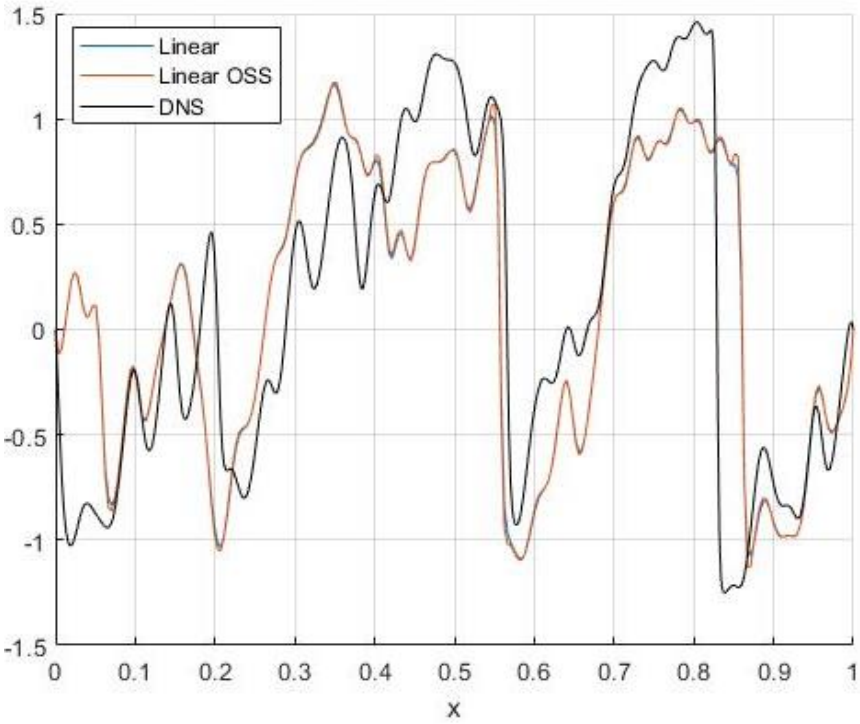


Figure 28 Comparison between ASGS and OSS solution

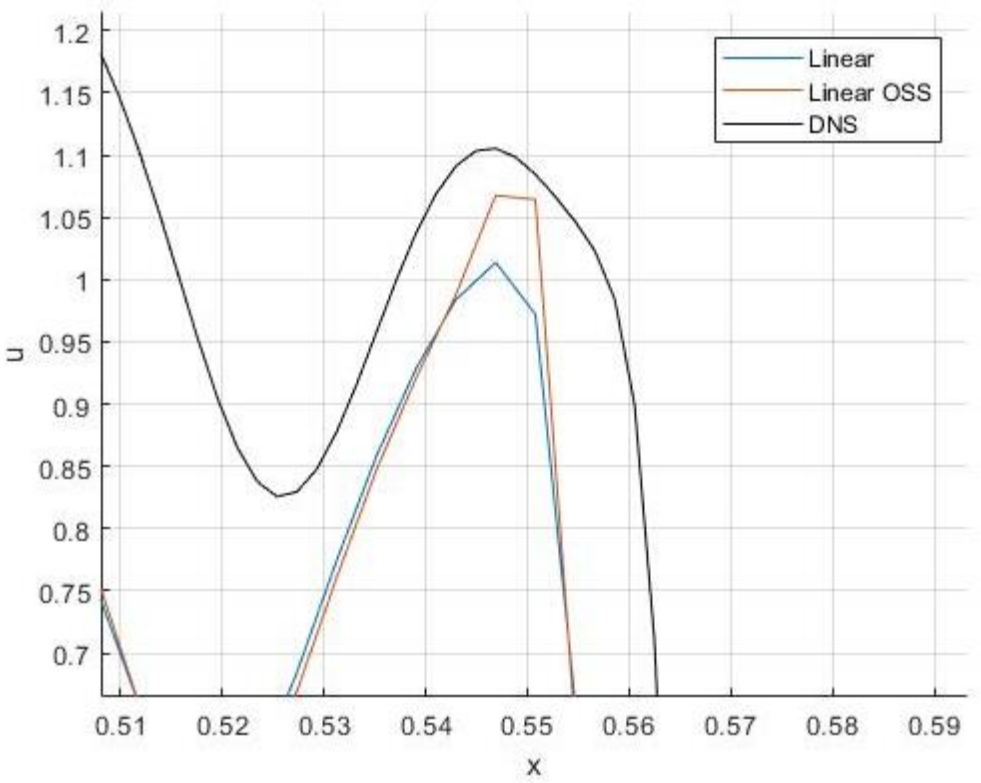


Figure 29 Comparison between ASGS and OSS solutions detail

The first comparison is between the linear ASGS and the OSS solutions. It is possible to observe that

- As we have seen in the previous analysis the OSS is less dumped so it can follow better the fluctuations of the solution.
- The approximate solution are slower than the DNS to react to big gradients imposed by the stochastic force term. Moreover they can't represent all the modes because the smallest are of the same dimension of the elements of the mesh. However both the ASGS and the OSS solution has the same trend

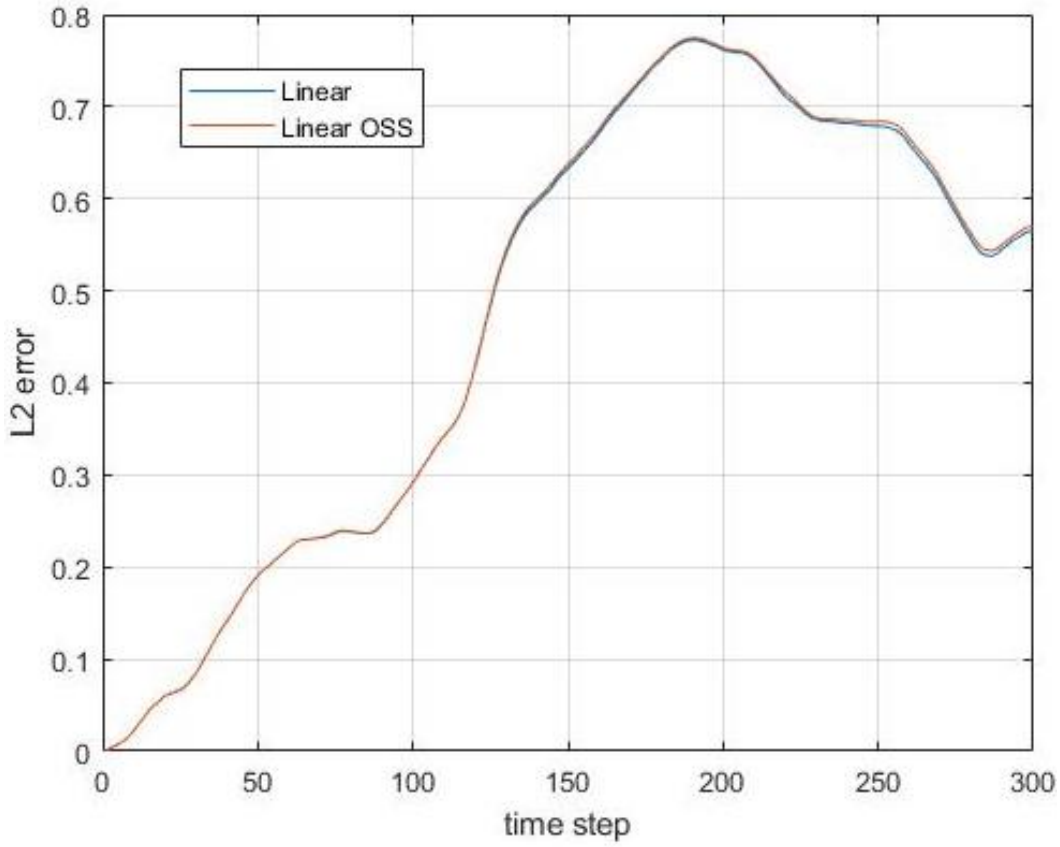


Figure 30 L2 error comparison between ASGS and OSS solution

- The L_2 error of the two solution is very similar even if the OSS is more precise. This is caused by the fact that it introduces less dissipation in the coarse scale equation. It can be seen also in the following picture where u' trend of the 2 solutions is plotted

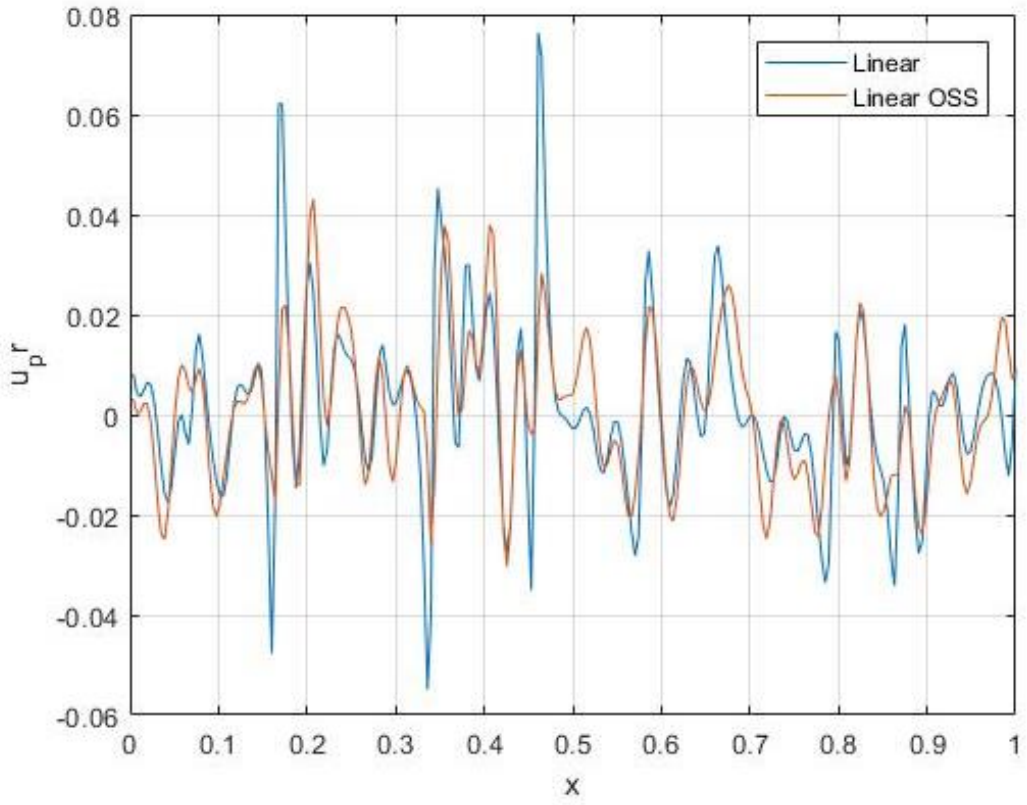


Figure 31 u' comparison between ASGS and OSS solution

- Finally it is presented an example of the plot of the coefficient c_0 as function of time to show the random behavior of the problem analyzed.

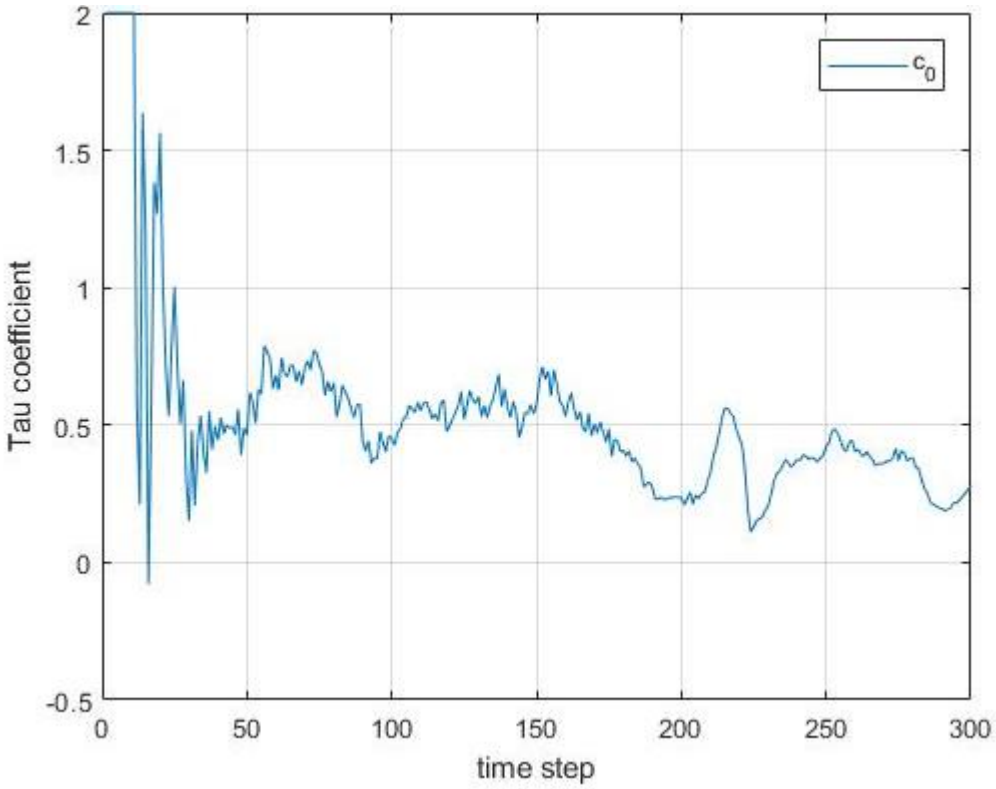


Figure 32 tau coefficient trend of linear OSS solution

6.3.2 Shakib compared with Linear tau

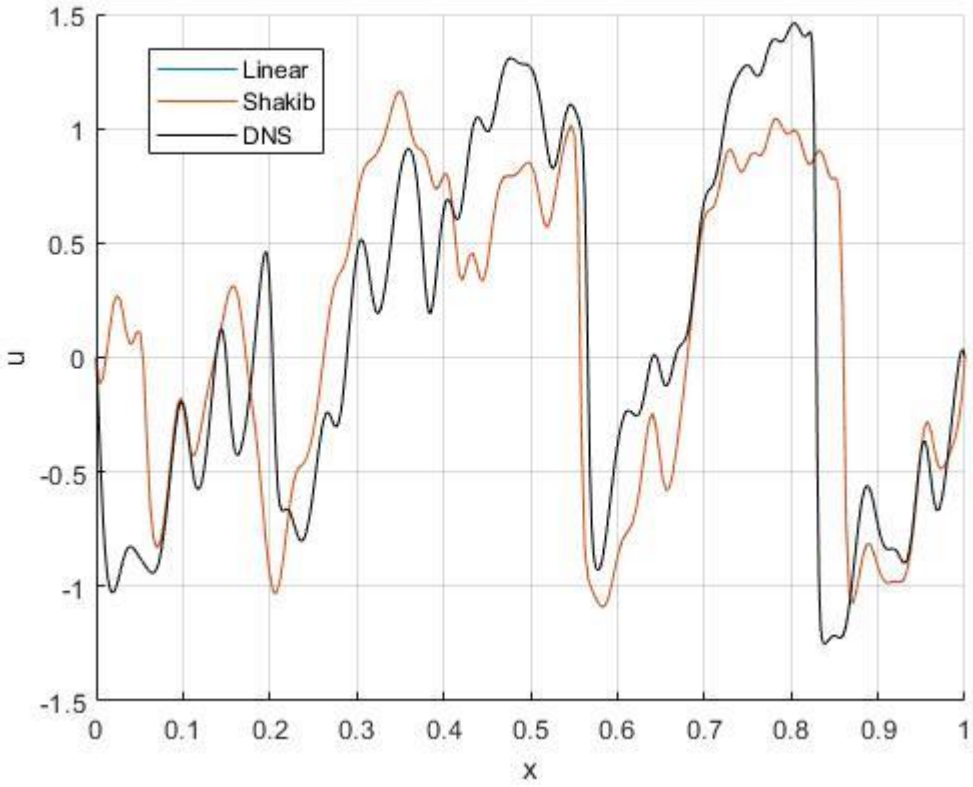


Figure 33 Comparison between Shakib and linear solution

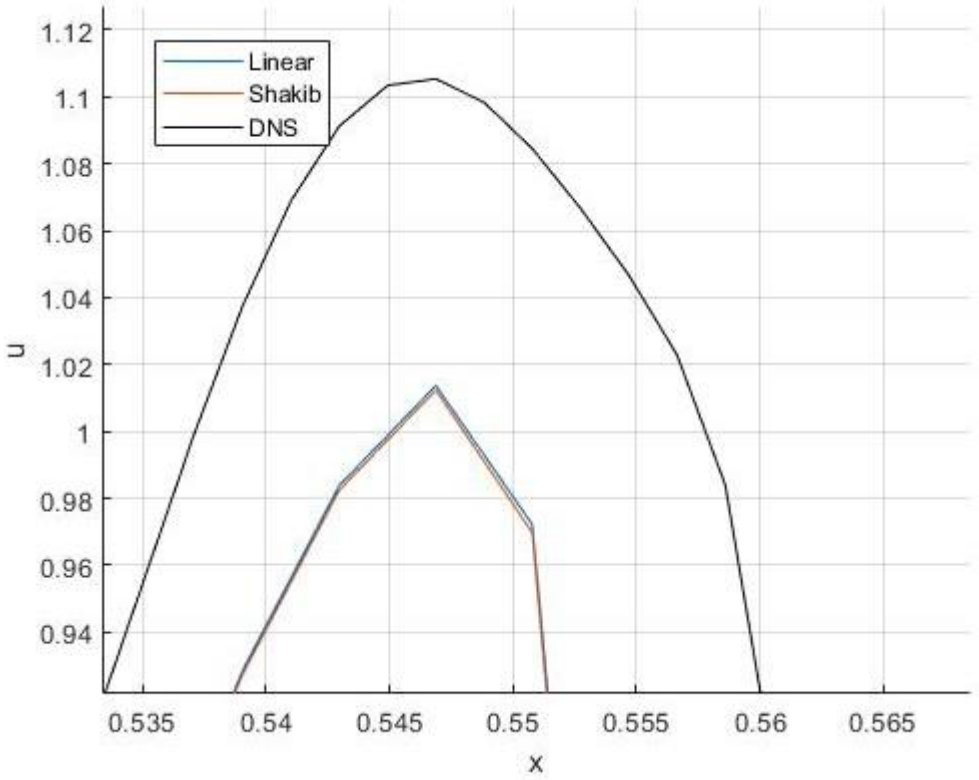


Figure 34 Comparison between Shakib and linear solution detail

The second comparison is made using the plot of the linear τ solution and the Shakib one. It is important to underline that

- The Shakib and the linear solutions are very similar each other in fact it is difficult to distinguish them also in the zoomed image. However the values of u' and τ are very different each other as it can be seen in the following pictures. The cause is that the influence of the sub model on the coarse scale equation is very small compared with the force term

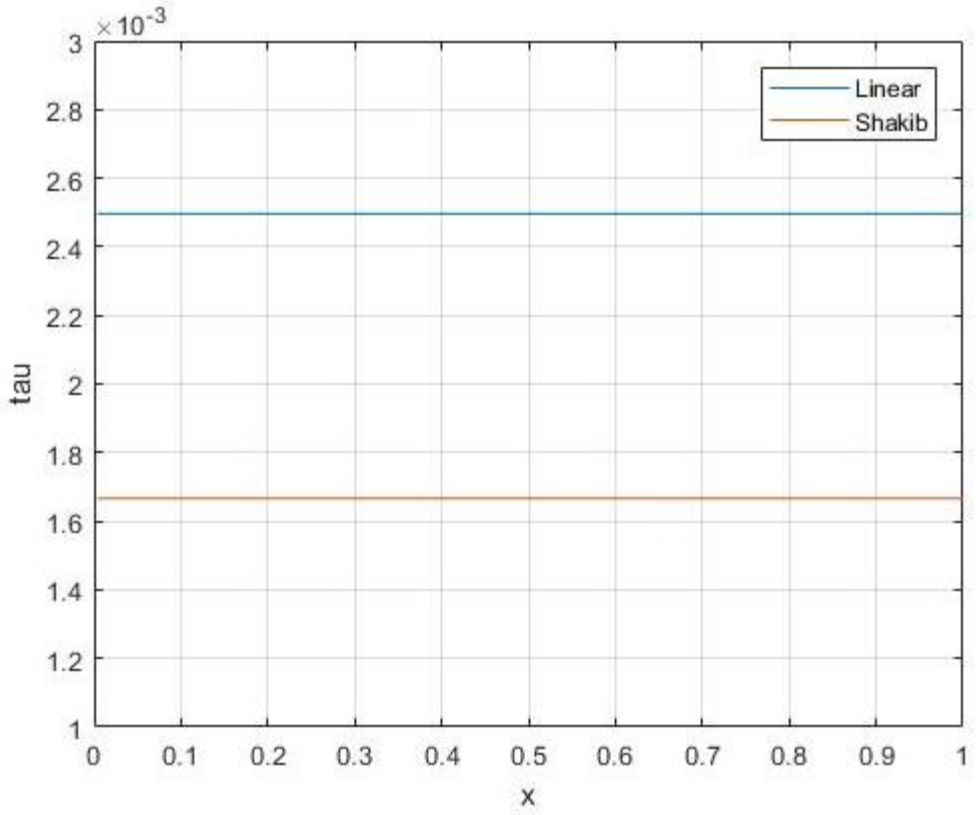


Figure 35 tau comparison between Shakib and linear solutions

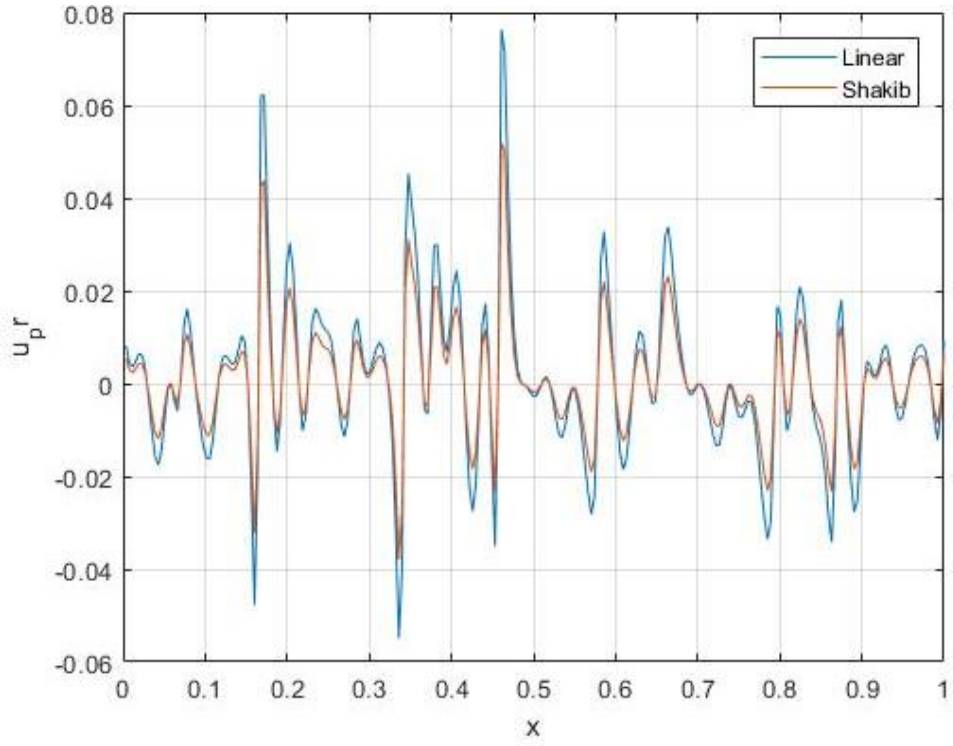


Figure 36 u' comparison between Shakib and linear solution

- A undesired phenomenon take place in the Shakib solution: the coefficients don't present a random trend as seen in the linear cases. (see the following picture). They tend to remain constant in time. It happens also for the Shakib space variant formulations: in these cases the coefficients remain equal to the initial values. As a consequence, since the results obtained don't represent an adequate solution of the problem, they have been omitted.

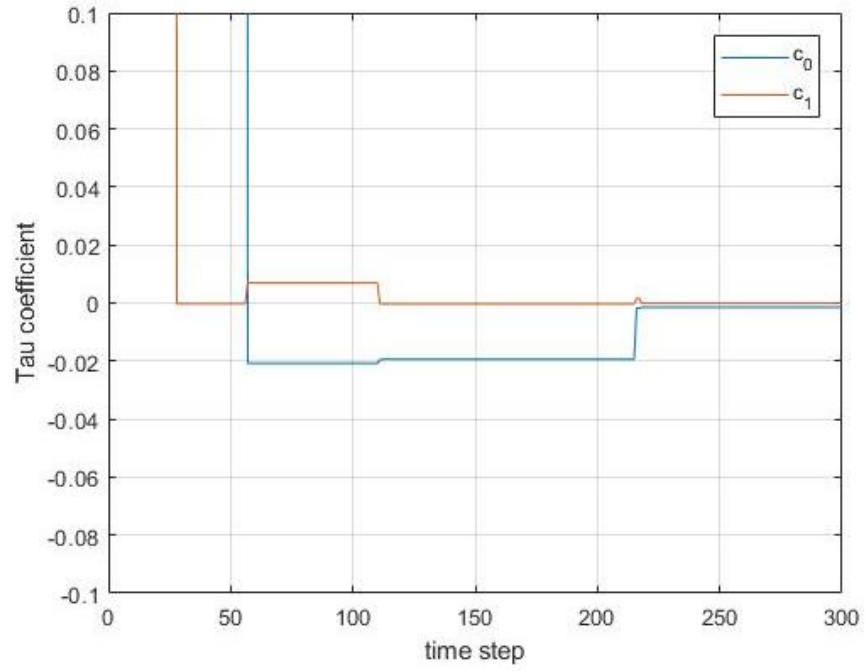


Figure 37 tau coefficients trend of Shakib solution

6.3.3Linear compared with SVT and SVT2

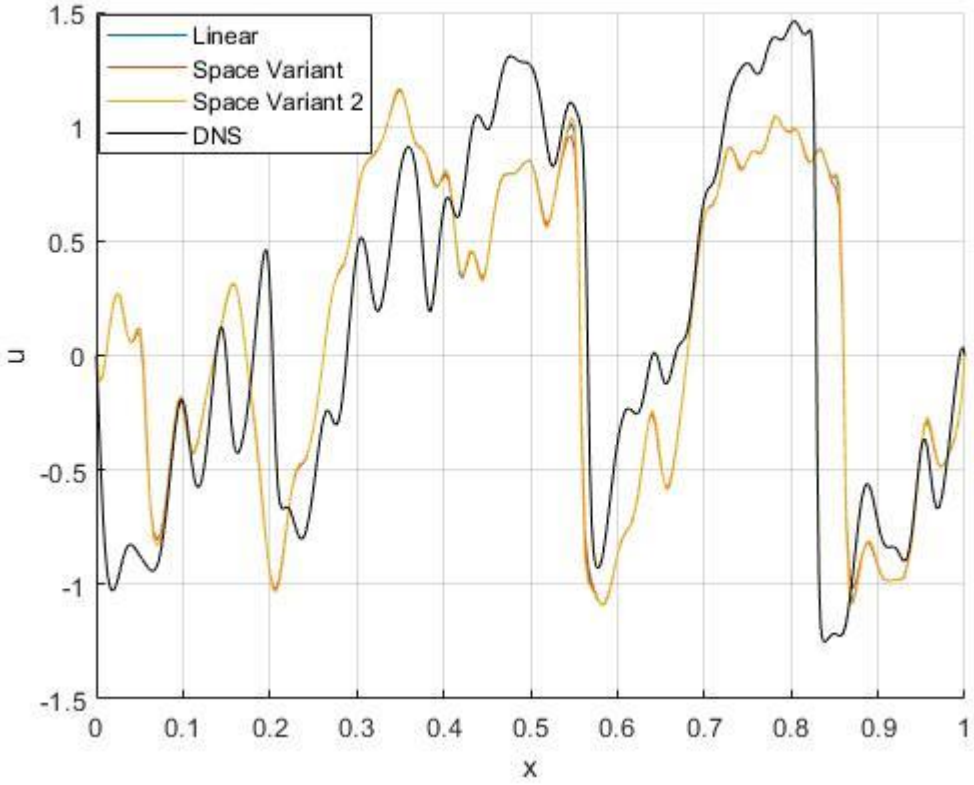


Figure 38 Comparison between linear, space variant and space variant2 solutions

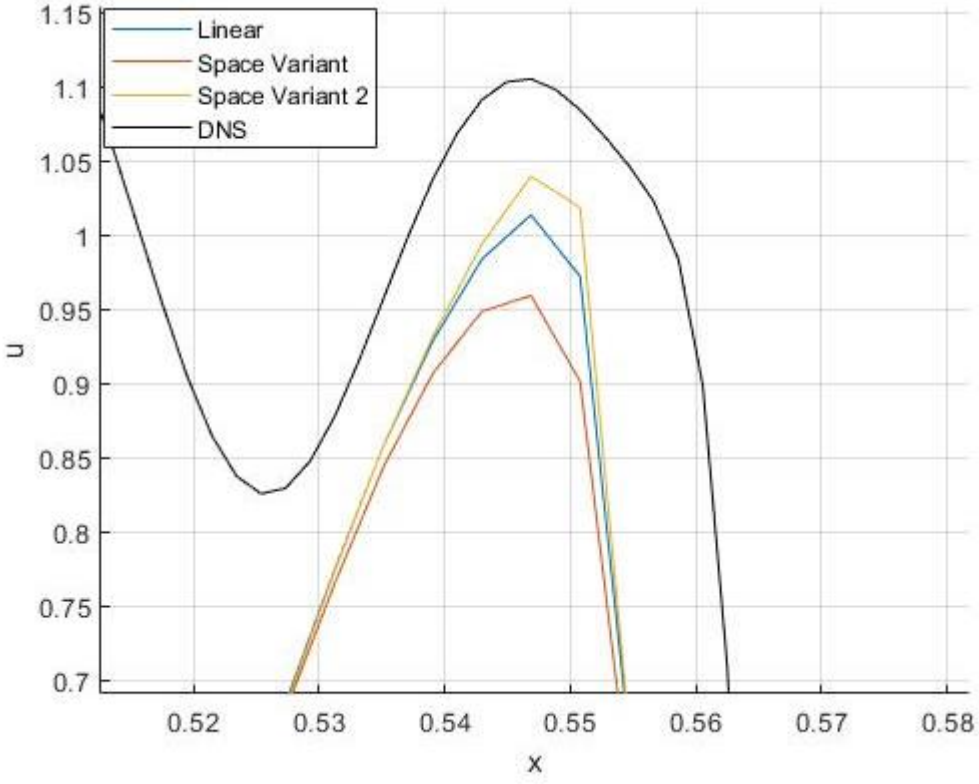


Figure 39Comparison between linear, space variant and space variant2 solutions detail

In this section the differences between the linear and the space variant solutions are pointed out.

- First of all we have to remark that the differences are very small; they can be observed only where the approximate solution present peaks. In all other points the solution are very close each other.
- The space variant 2 solution can best follow the oscillations of the DNS even if, as already written, the smallest mode can't be represent by the LES solutions. On the other hand the space variant one seems to be less precise then the linear.
- As we can see in the following picture the space variant 2 is the formulation which introduce less dissipation in the coarse scale solution. This is the reason why its solution is less dumped

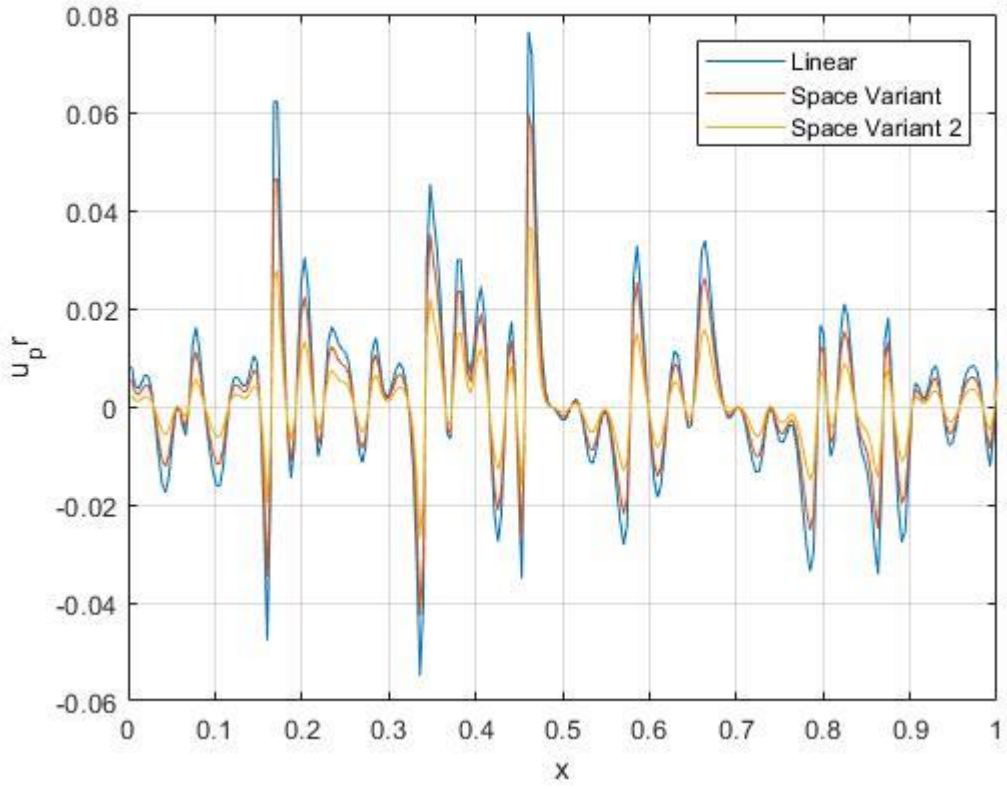


Figure 40 u' comparison between linear, space variant and space variant2 solutions

- In fact the resulting τ of the space variant 2 is the smallest

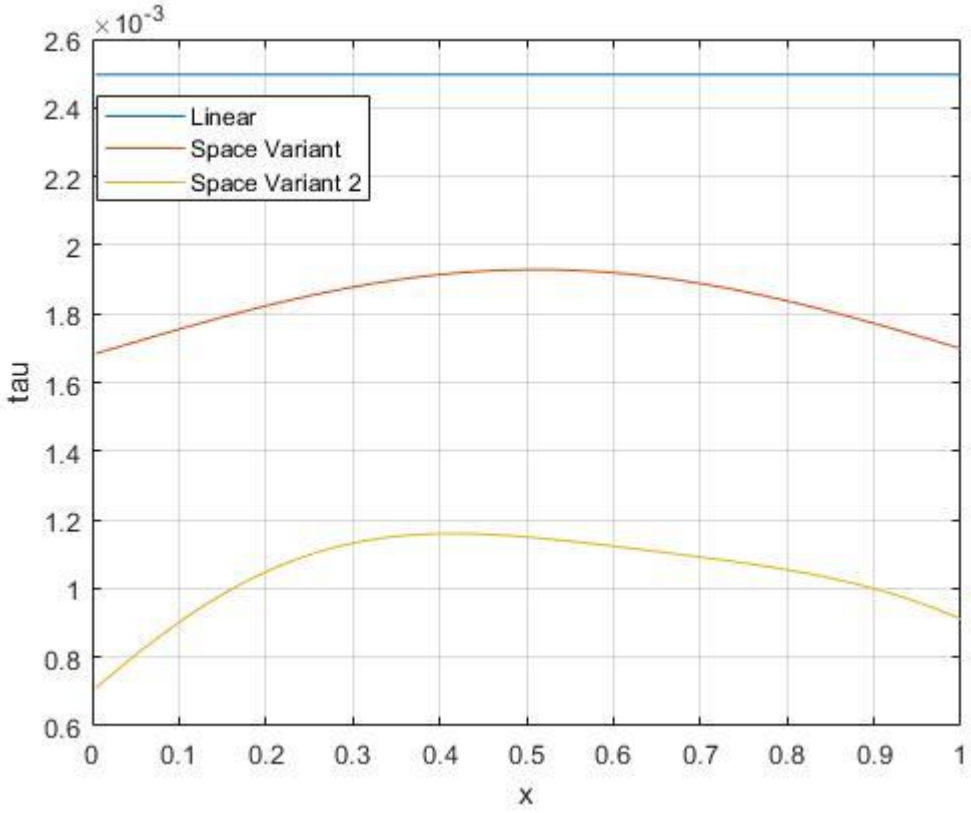


Figure 41 tau comparison between linear, space variant and space variant2 solutions

6.3.4 General comparison

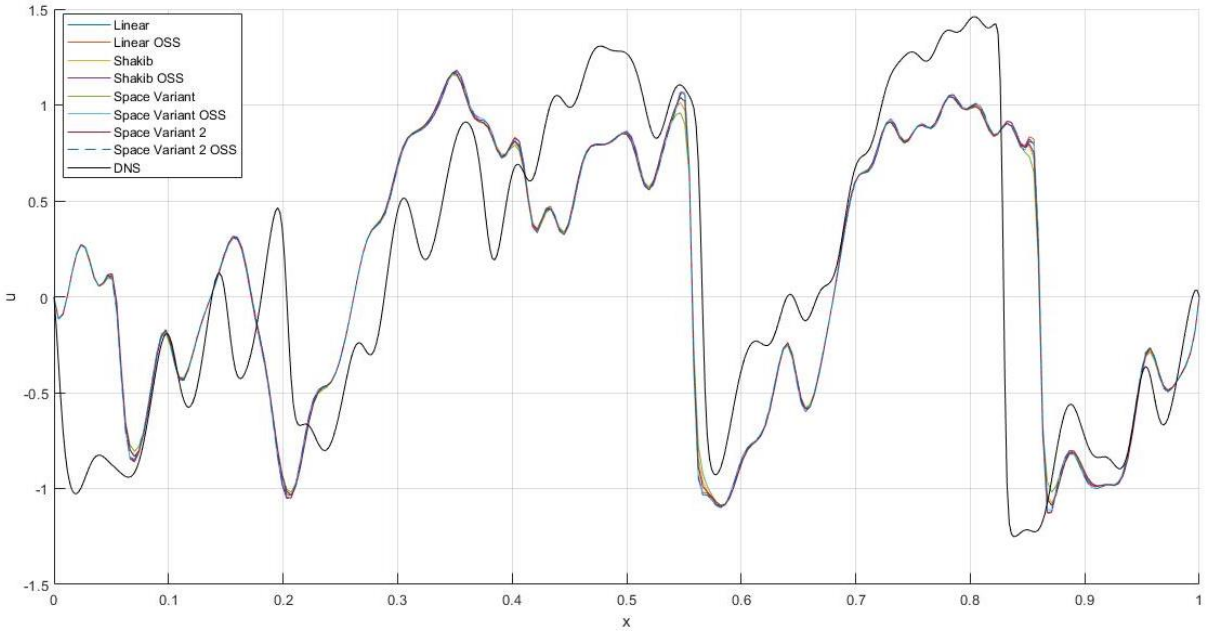


Figure 42 General comparison between all the solutions

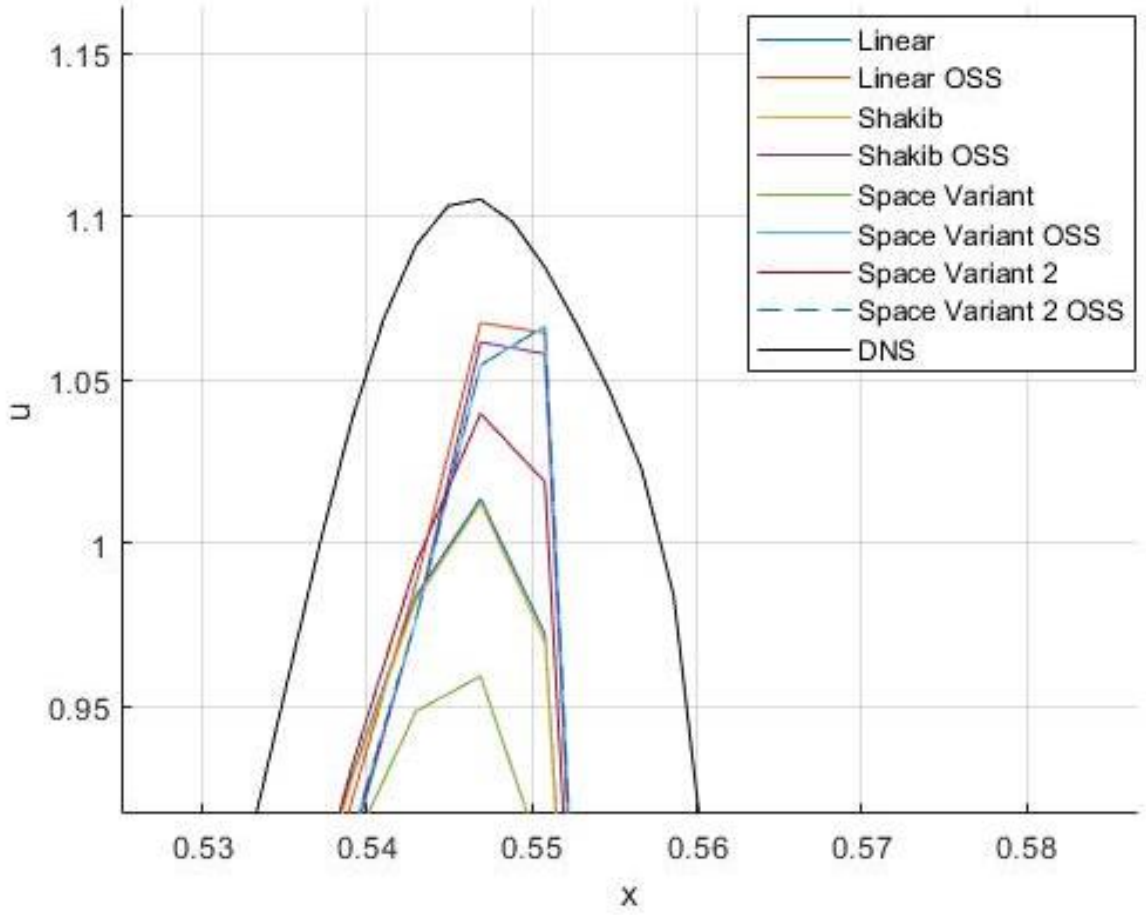


Figure 43 General comparison between all the solutions detail

We can conclude the analysis of the random force term remarking that

- The best solution is the Linear OSS one because it can best follow the fluctuations of the DNS; generally the OSS solutions look better than the ASGS ones
- The effects of the not resolved modes and the delay of the approximate solution are relevant. In fact it is bigger the difference between the DNS and all the approximate solutions than the one between the approximated solutions themselves.
- The Shakib Space variant solution have been excluded by the analysis because of their unrealistic trend

CHAPTER 7

7. CONCLUSIONS

In this thesis the Variational Germano Method has been used to solve numerically the 1D Burger equation, which has the same structure of the Navier Stokes equation. Two force term have been used to test different Subgrid Scale models in order to find the best one. The best solution is that is more similar to the DNS so it doesn't present undesired oscillations, is characterized by the minimum L_2 error and it doesn't request a big computational effort.

Analyzing the Gabriel force term we have managed to improve the Shakib formulation of τ, which is considered the best approximation of the Green operator, using a space variant one (SVT2).However we haven't been able to define a good space variant formulation with the same structure of the Shakib one. Finally we have verified that the OSS method introduces too little numerical diffusion and so in this case it doesn't result very useful.

Then we have tried to test the same formulations with Chamber random force term which can better represent a turbulent flow. We have obtained very bad results because the approximate solutions can't follow all the modes of the force term; the difference between each solution is less significant compared by the deviation from the DNS . Moreover the Shakib space variant formulations can't obtain numerical reasonable solutions. However the main feature of the subgrid scale models observed in the first analysis are confirmed.

Further improvements of this work can be a new Shakib space variant formulation which can allow to be better than SVT2, and the use of a different solver to avoid numerical difficulties encountered testing the Chamber force term. Another improvement can be to obtain a relationship between the Reynolds number the formulations and the integration points they need to converge. Moreover this method can be used in the future to solve the Navier Stokes multi-dimensional equations.

BIBLIOGRAPHY

[1] T.J.R. Huges ,G. Scovazzi,L.P. Franca “Multiscale and Stabilized Methods” Encyclopedia of Computational Mechanics vol 3 cap.2,2004

[2] Y. Bazilevs, V.M. Calo,J.A. Cottrel,T.J.R. Huges, A. Reali, G.Scovazzi “Variational Multiscale residual based turbulence modeling for large eddy simulation of incompressible flow” Computer methods in applied mechanics and engineering vol 197 pp. 173-201,2007

[3] G.Maher “Variational Germano Optimization of Arbitrary Unresolved Scale Models” Master of Science Thesis ,TU Delft,2014

[4]A.A. Oberai, J Wanderer “A dynamic approach for evaluating parameters in a numerical method” International journal for numerical methods in engineering vol 62 pp. 50-71,2004

[5] A.A. Oberai, J. Wanderer “Variational formulation of the Germano identity for the Navier-Stokes equation” Journal of Turbolence , 6,N7,2011

[6] A.A. Oberai, D. Sondak “Application of the variational Germano identity to the variational multiscale formulation” International journal for numerical methods in biomedical engineering vol 27 pp. 335-344,2009

[7] A.A. Oberai, Z. Wang “ Optimal numerical solution of PDEs using the variational Germano identity” Computer methods in applied mechanics and engineering vol 197 pp. 2948-2962,2008

[8] Z. Wang, A.A. Oberai “A mixed large eddy simulation model based on the residual based variational multiscale formulation” Physics of Fluids vol 22, 2010

[9] I. Akkerman, K.G. van der Zee, S.J.Hulshoff “A variational Germano approach for stabilized finite element methods” Computer methods in applied mechanics and engineering vol 199 pp. 502-513,2010

[10] M. Landajuela BCAM Internship 2011

[11] S.J. Hulshoff “CFD 3: Large eddy simulation documents for the course” 2017 version

[12] S.J. Hulshoff “Computational modelling documents for the course” 2017 version

[13] J. Smagorinsky “General circulation experiments with primitive equations. I. The basic experiment.” Monthly Weather Rev. 91:99-164,1963

[14] T.J.R. Huges, G. Sangalli “Variational multiscale analysis: the fine scale Green’s function, projection, optimization, localization and stabilized methods” SIAM Journal of numerical analysis, vol 45pp. 539-557,2007

[15] M. Germano , U:Piomelli, P. Moin,WH. Cabot “A dynamic sugrid-scale eddy viscosity model” Physics of fluids 3(7): 1760-1765, 1991

[16] Lilly DK. A proposed modification of the Germano subgrid-scale closure method. Physics of Fluids A; 4(3):633–635, 1992

[17] O. Colomes, S. Badia, R. Codina, J. Principe “Assessment of variational multiscale models for the large eddy simulation of turbolent incompressible flows” Comput. Methods Appl. Mech.Eng, 285 pp 32-63,2015

[18] R. Codina “Stabilized finite element approximation of transient incompressible flows using orthogonal subscales” Comput. Methods Appl. Mech.Eng, 191 pp 4295-4231,2002

[19] Y. Yuan, “ A modified BFGS algorithm for unconstrained optimization," IMA journal of Numerical Analysis, vol. 11, pp. 325-332, 1991.

[20] P. Wolfe “Convergence conditions for ascent methods," SIAM review, vol. 11, pp. 226-235, 1969

[21] Y. Saad, M. Schultz “GMRES a generalize minimal residual algorithm for solving non symmetric linear systems” SIAM Sci. Stat.Comput. vol 7 pp 856-869,1986

[22] W. E. Arnoldi.”The principle of minimized iteration in the solution of the matrix eigenvalue problem” Quart. Appl. Math., 9 , pp. 17-29,1951

[23] L.J. Aguilar “Fondamenti di programmazione in c++. Algoritmi, strutture dati ed oggetti.” McGraw-Hill

[24] D.T. Jeng “Forced Model Equation for Turbulence.” Physics of Fluids, 12 pp2010-2006 1969

[25] David H. Chambers “ Statistical Representations of Cohrernt Structures in Turbulent Flow Fields.” Phd Thesis submitted to Univeristy of Illinois at Urbana Champaign, 1987

[26] F. Shakib “Finite element analysis of the compressible Euler and Navier-Stokes equations.” PhD thesis, Stanford University, 1988

[27] T.J.R. Huges, M.Mallet “A new finite element formulation for computational fluid dynamics: III. The generalized streamline operator for multidimensional advective diffusive systems”,Comput. Methods Appl. Mech.Eng, 58 pp 305-328,1986