Politecnico di Torino

von KARMAN INSTITUTE FOR FLUID DYNAMICS

Politecnico di Torino

Dipartimento di Ingegneria Meccanica e Aerospaziale

in collaboration with

Von Karman Institute

Department of Environmental and applied fluid dynamics

Master of Science in Aerospace engineering

Thesis in:

Development of a spectral method code for simulating liquid films instabilities in hot dip galvanisation

Supervisor:

Sandra Pieraccini

Miguel Alfonso Mendez

Candidate:

Aleksandr Ferrigno

s288790

academic year **2022 / 2023**

Acknowledgements

This work is the conclusion and the culmination of my studies which were characterised by ups and downs during which I was encouraged by all those people that always believed in me therefore I sincerely thank them. A special thank goes to my family and my girlfriend who always supported me with everything needed to produce this master thesis that I hope will be useful to everybody that will follow my steps. Last but not least I would love to thank my supervisor Miguel Alfonso Mendez who led me extremely well during my journey making sure that the work, before being done, was correctly understood.

Abstract

Hot dip galvanisation is an industrial process of coating iron and steel in zinc. The process can be optimised with the usage of gas actuators which can reduce the thickness, therefore the zinc used is less and the cost is reduced, meanwhile improving the homogeneity of the final coating. This cost-effective technique is known as jet wiping and it is widely used in industrial processes. However the interaction between the gas accelerated by the jet and the liquid film produces a wavy final coating films. The aim of the research team of the Von Karman Institute, in collaboration with ArcelorMittal, is the development of the BLEW (Boundary LayEr Wiping) software to predict the liquid films dynamic. The aim of this work is to keep improving the BLEW solver introducing a spectral method needed to reduce the computational time and numerical diffusion. This thesis presents the spectral method, introduces different examples used to understand the ins and outs of the scheme and finally shows the application to the real case study.

Contents

CHAPTER

1 Introduction

The scope of this master thesis is to present the work done in improving the solver of the BLEW environment by implementing a spectral scheme instead of the currently used finite volume scheme which is treated extensively in [2].

This work is the first step taken in the application of spectral methods on the BLEW environment and has the goal of creating a solid foundation and giving an understandable code of the first implementation of the scheme.

Therefore, firstly, basic theory of spectral techniques is presented, followed by some examples; secondly, the BLEW environment is introduced; thirdly, the spectral scheme applied to simplified BLEW equations is discussed; lastly, conclusions and future development are projected.

1.1 Hot-Dip Galvanisation

Coating techniques are industrial processes based on the deposition of liquid film on solid surface. The Hot-Dip Galvanisation is one of them, in fact moving steel strips are coated in a thin layer of fluid zinc which once cooled down is capable of resisting oxidation. However, at common strip speed, the liquid layer drag is far to thick and uneven which results in high cost and later structural problems.

To avoid these problems and optimise the process the jet wiping technique, known also as air-knife coating technique, is used. It utilises high speed gas jet impinging on the liquid layer which form a run back flow down to the bath reducing the thickness and improving the homogeneity of the layer itself. Moreover thanks to the development of technologies and the advancement of computer science it is possible to implement Machine learning algorithms to control the velocity output of the jet to further improve the final result. The schematic of the process is shown in figure 1.1.

The moving metal strip passes in the liquid bath in order to be coated in zinc and then it is pulled upward where thanks to the gravity a part of the coating is removed. However its thickness is still considerable, therefore the jet-wiping technique is applied which further decreases the final thickness of the coat. The output of the jet is determined by the controller which follows policies generated by the Machine Learning algorithm. The part related to the Machine Learning is presented in [5].

The coating layer, during the process, is divided in two regions: the final coat of thickness h_f and the coat characterised by run back flow of thickness h_R caused partially by gravity. Those two parameters depend on: the velocity U of the metal strip, nozzle gauge pressure ΔP_n, distance Z between the nozzle and the strip, the nozzle slot width d and the nozzle tilt angle α as well as liquid properties such as: density ρ, dynamic viscosity μ and surface tension σ. These are clearly shown in figure 1.2.

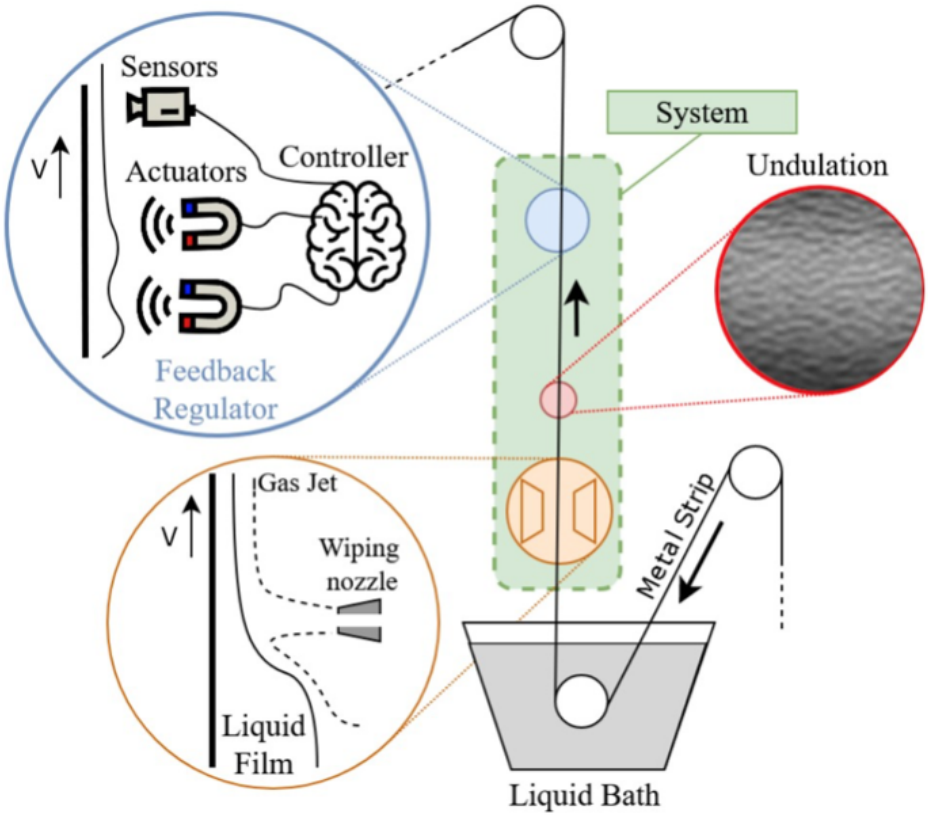


Figure 1.1: 2D-Schematic of the Jet Wiping Process applied to hot-dip galvanisation

Although the jet wiping technique is a very efficient and profitable coating process, it is not free from problems. Mainly two: undulation and splashing. The first problem arises in certain operating conditions where the jet starts to oscillate introducing a wavy pattern in the final coating, while the second one occurs due to high velocity of the metal strip causing separation of the liquid film from the substrate in the run back flow. While undulation are instabilities that compromise just the homogeneity of the final coat the splashing are more dire film instabilities because they degrade significantly the wiping conditions (pressure gradient and shear stress at jet impingement), and leads to unstable conditions and to the formation of zinc droplets which leads to poor wiping efficiency and potentially to the obstruction of the nozzle due to solidified zinc droplets. Moreover this is concerning for the safety of the worker

responsible of the elimination of the layer of oxidised zinc. In addition high production rates are to be maintained, which is not yet totally accomplished, in order for the process to be industrially practical. These instabilities are shown in figure 1.2.

Therefore, a predictive model of the instabilities of liquid film is needed in the industrial lines. While more information on splashing can be found in [3], the undulation problem is still topic of research.

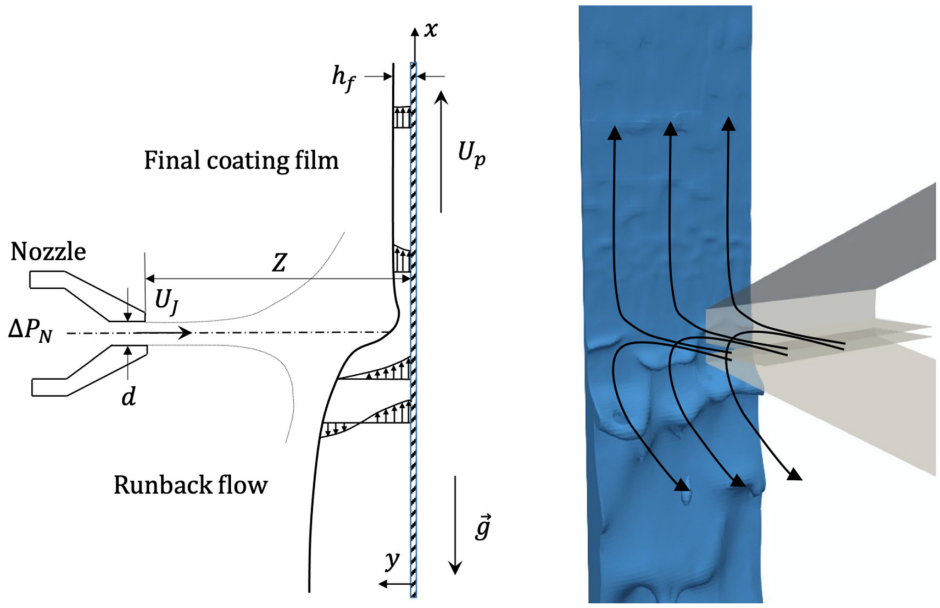


Figure 1.2: 3D-Schematic of the Jet Wiping Process applied to hot-dip galvanisation

CHAPTER

2 Spectral Methods

Spectral methods, together with finite difference methods and finite element methods, are numerical schemes used in solving Partial Differential Equations, known also as PDEs, and they are the ”most recently”, 1970s, discovered out of the three schemes.

If one wants to solve Ordinary Differential Equation, for brevity ODEs, or PDEs with high accuracy on a relatively simple domain and if the data defining the problem are smooth, then spectral methods are usually the best choice because they can achieve higher accuracy with respect to other methods.
The last characteristic but not less important is the capacity of these methods to simplify greatly the application of implicit schemes to complex equations which means less work for the mathematicians, less code complexity and therefore reduced computational costs. This aspect will be shown in the practical examples.

This chapter presents the main idea behind the spectral methods and for a deeper understanding the reader is referred to [4] and [6]. Afterwards, some examples will be presented.

2.1 Theory of Spectral Methods

Spectral Techniques are based on the idea of writing the solution of differential equations as a sum of basis function where each term is multiplied for certain coefficients. The choice of these leads to an appropriate representation of the differential equations. For instance the *Fourier transform*, FT, of a function $u(x)$, $x \in \mathbb{R}$, is the function $\hat{u}(k)$ defined as

$$\hat{u}(k) = \int_{-\infty}^{+\infty} e^{-ikx} u(x) dx \tag{2.1}$$

Where $\hat{u}(k)$ can be interpreted as the amplitude density of u at the wave number k, therefore, on one hand, the FT is a transform that converts a function into a form that describes the frequencies present in the original function. The output of the transform is a complex-valued function of frequency. On the other hand to reconstruct u, function expressed in the physical domain, from $\hat{u}$, expressed in the frequency domain, the *inverse Fourier Transform* is needed to be performed. Therefore

$$u(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{ikx} \hat{u}(k) dk \tag{2.2}$$

Since there are equations that can not be solved analytically, for instance the Navier-Stokes equations, then they are solved by approximations. These are calculated in a discrete space therefore the wave-number k will no longer range over $\mathbb{R}$, instead the wave-number domain is a bounded interval of length $\frac{2\pi}{h}$ where h is the length of the interval discretization therefore it should be remembered that k is bounded because x is discrete. Intervals of length different from 2π are easily handled by a scale factor.

Something that should be paid attention to is the *aliasing* phenomenon responsible for the indistinguishability of the signals sampled. This phenomenon is caused by too low sam-

ple rate for sampling a particular signal or too high frequencies present in the signal for a particular sample rate. It is easily understandable from the figure 2.1 below. A way to mitigate this problem is by applying the The Nyquist-Shannon theorem also known as the sampling theorem. It states that a periodic signal must be sampled at twice the maximum frequency characterising the data inputted. Otherwise it is possible to filter the higher frequencies.

Another requirement of the spectral methods is the periodicity of the boundary conditions which at a first glance may suggest that this methods have limited relevance for the practical problems, however periodic grids are surprisingly useful in practice. In these methods the periodicity is not explicitly stated, as happens in the other two numerical technique where boundary conditions are defined, but it is implicit, which means that the function that represents the initial condition should be periodic in the physical domain. If this condition is not respected then huge oscillations in the solution are introduced at the boundary of the domain.

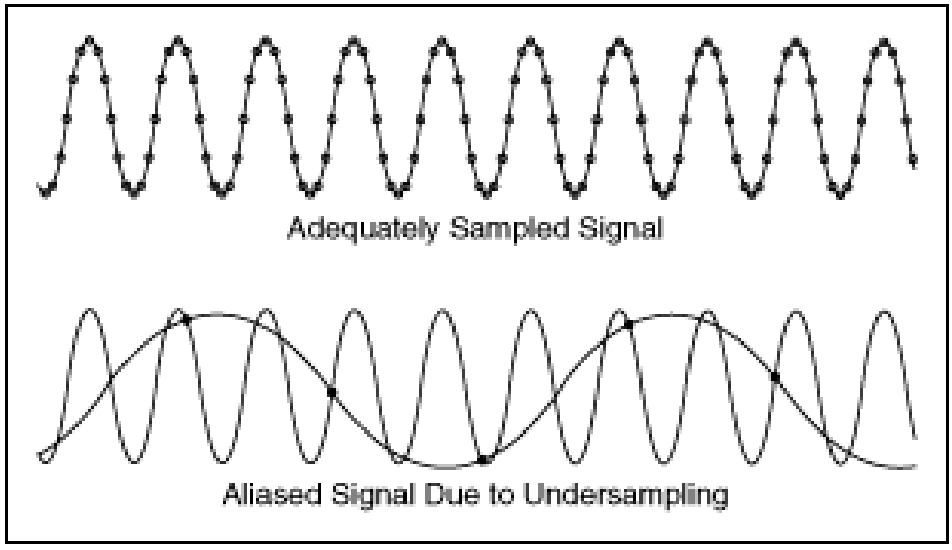


Figure 2.1: aliasing phenomenon

2.1.1 Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is a multiplication operation by which a matrix of coefficients is multiplied by the sampled data vector. The DFT is therefore the representation of the frequencies characterising the data vector sampled. The Fourier analysis is widely used in practical applications and it is performed by applying the DFT which, as stated before, is the representation of the frequencies characterising the sampled vector. However it is not an efficient operation since it computational cost is equal to $O(n^2)$. The DFT and iDFT are respectively defined as

$$\hat{u}_k = \sum_{j=1}^{N} u_j e^{-ikx_j}, \quad k = -\frac{N}{2}+1, ..., \frac{N}{2} \tag{2.3}$$

$$u_j = \frac{1}{2\pi} \sum_{k=-N/2+1}^{N/2} \hat{u}_k e^{ikx_j}, \quad j = 1, ..., N \tag{2.4}$$

For a more strict demonstration the reader is redirected to [4] and [6].

In addition to the increase in the accuracy and the reduction of the computational cost, as reported in the practical examples shown below, spectral methods simplify the computation of derivatives based on the following algorithm:

- Given a function $u(x)$, compute $\hat{u}(k)$
- Define $\partial^\nu \hat{u} = (ik)^\nu \hat{u}$
- Compute $\partial^\nu u(x)$ from $\partial^\nu \hat{u}$

2.1.2 Fast Fourier Transform

The Fast Fourier Transform (FFT) is an algorithm that optimise the Discrete Fourier Transform operation, or its inverse, iDFT. Fourier analysis converts a signal from its original domain to a representation in the frequency domain and viceversa. If the DFT and its inverse are computed following their definition reported respectively in (2.3) and (2.4) then the process is too slow to be of practical use. However the FFT is extremely ingenious because it allows the reduction of the computational cost of the DFT from a quadratic order, $O(N^2)$, to an almost linear one, $O(NlogN)$, speeding up the process. This is done by factorising the DFT matrix into a product of sparse, mostly zero, factors.

The spectral methods relaying on FFT can be summed up as follows:

- Given an initial condition which is a periodic function $u(x)$ expressed on a equally spaced grid of spacing h
- Determine the wave-numbers which are function of the number of grid points and the spacing of the grid itself
- Determine $\hat{u}(k)$ using the FFT, function in the frequency domain
- Compute the necessary derivatives while being in the frequency domain as : $\partial_v \hat{u} = (ik)^v \hat{u}$
- Switch back to the physical domain utilising the iFFT
- Compute the Right Hand Side of the equation studied
- Integrates in time what was obtained

Not only these schemes reduces the computational cost but they also simplify greatly the mathematical calculations and therefore the code implementation. This is accomplished thanks to the shift between domains which permit to execute operations between terms that otherwise could not happen; for instance the product between a variable x and the derivative of a variable y can be implemented as a product of the variable y times the derivative of the variable x. This flexibility facilitate the following operations reducing the time spent and the equation's complexity, which lead to a mathematically more beautiful relation.
The example from before is explicated below:

$$x\partial_x y \xrightarrow{\text{FFT}} \hat{x} i k_x \hat{y} = \hat{y} i k_x \hat{x} \xrightarrow{\text{iFFT}} y\partial_x x \tag{2.5}$$

2.2 Practical examples

This section has the scope to present the application of the spectral methods to a set of different equations, in particular, they will be presented based on their complexity. These case studies are : 1-D advection equation, 1-D Burgers equations, 2-D advection equation and finally 2-D Burgers equation. These examples are realised in python.

2.2.1 1D Advection Equation

$$\partial_t u + a\partial_x u = 0 \tag{2.6}$$

This equation is a PDE which describes the transportation of substance and its properties by bulk motion of fluid using some scalar field, $u(x,t)$ unknown, carried along a direction x by a flow of constant speed a. It requires an initial condition

$$u(x,0) = u_0 \tag{2.7}$$

where u_0 is given. Moreover if the transport occurs in a finite domain then boundary conditions are required. If the wave is moving from left to right in a domain $x \in [x_L; x_R]$ then the velocity is defined as $a > 0$ thus the following boundary condition must be given:

$$u(x_L, 0) = u_L(t) \tag{2.8}$$

Its solution has the form

$$u(x,t) = u_0(x - at) \tag{2.9}$$

Therefore the initial profile is simply "swept along" with velocity a. To solve it numerically two methods are used : Upwind Scheme and Spectral Scheme.

Upwind Scheme

This scheme is used to solve ODEs and PDEs using the upstream variables to compute the derivatives which are estimated using a set of data points biased to be more "upwind" of the query point, with respect to the direction of the flow. This scheme is not the focus of the thesis work therefore it will not be treated extensively; it is only introduced for comparisons.

The scheme used for the comparison is the simplest and it is first order, moreover, if the velocity is considered to be positive then the scheme is defined as

$$\frac{u_i^{n+1} - u_i^N}{\Delta t} + a\frac{u_i^n - u_{i-1}^n}{\Delta x} = 0 \tag{2.10}$$

This formulation can be written in its matrix form as

$$\bar{u}^{n+1} = -\frac{a}{\Delta x}(FD \cdot \bar{u}^n) \tag{2.11}$$

where FD is a matrix which stands for First Derivative. Its effect, when multiplied by $\bar{u}$, is to compute the approximation of the derivative upon division by Δx.

The upwind scheme is stable if the discretization parameters satisfy the Courant–Friedrichs–Lewy stability condition (also known as CFL condition). The CFL condition states that Δt and Δx should satisfy (eq. 2.12) and the quantity $C = a\Delta t/\Delta x$ is called Courant number.

$$C = a\frac{\Delta t}{\Delta x} < 1 \tag{2.12}$$

The Scheme is presented in the appendix while the numerical solution of the one dimensional advection equation is shown in figure 2.2.

Spectral Scheme

To solve numerically this equation utilising spectral methods it is needed firstly to define the solution as follows

$$u(x,t) = \sum_{n=1}^{N} \hat{u}_n(t) e^{ik_n x} \tag{2.13}$$

Then the following algorithm is used to compute the solution:

1. A discrete domain is defined in space and time
2. Initial condition is defined
3. Wave-numbers are calculated
4. FFT of the solution is computed
5. Space derivatives are calculated as : $\partial_x u = aik\hat{u}$
6. The previous derivatives are transformed back into space domain using the iFFT and all the terms at the Right Hand Side are computed

7. Finally the solution is integrated in time with the Runge-Kutta method used by *"solve ivp"* which is a built in function in python. Then a gif is built to show visually the results.

The code is presented in the appendix. The results obtained with the algorithm, on the considered test case, are shown in figure 2.3.

The domain length is 3 with $x \in [0;3]$, the number of grid points is equal to 100 and therefore the grid spacing is given by L/nx, equal to 0.03. The velocity considered is 1 and the C is equal to 0.8 therefore the time stepping is calculated from its inverse. The duration of the simulation is 4 second. The initial condition is a Gaussian function with mean value $\mu = 0.6$ and standard deviation equal to $\sigma = 0.05$

Comparison

Figure 2.2 and 2.3 are snapshots of the solution obtained with the two methods. The red line represents the analytical solution while the blue one is the computed one.

The discretisation characterising the upwind method, shown in figure 2.2, satisfies the CFL condition but its grid mesh is coarse therefore the numerical dispersion, which is always present at different magnitudes based on the quality of the numerical scheme, is enormous. This phenomenon can be seen from the comparison between the analytical solution and the numerical one. To solve this problem the spatial mesh should be finer.

The solution obtained with spectral scheme, shown in figure 2.3, does not display numerical dispersion due to its incredible accuracy for smooth functions as reported in [6]. The method should be way faster than the upwind one but it is not appreciated in this case since the quantity

of the data used is not high enough; in fact its true prowess is shown when it works with billions of data.

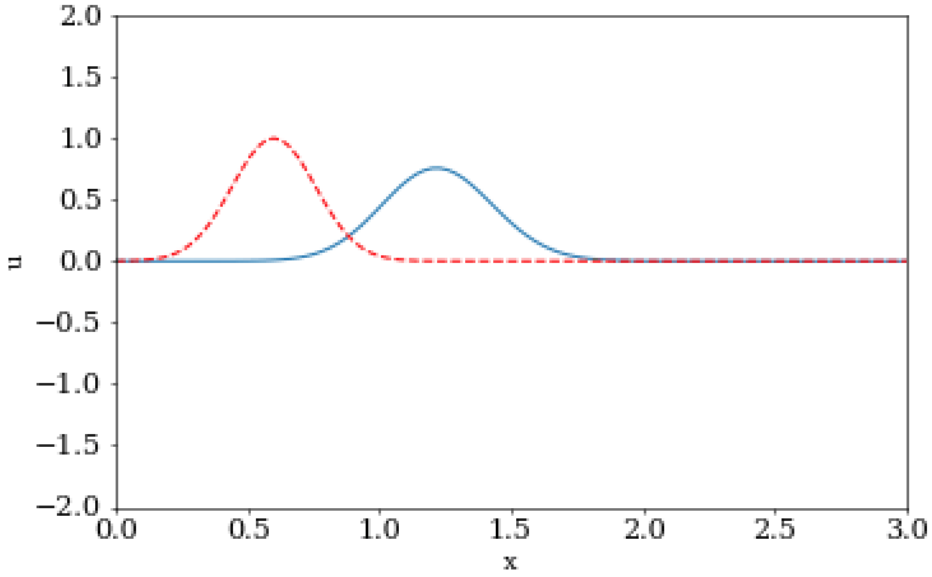


Figure 2.2: Upwind scheme applied to the 1D advection equation. The broken red line represent the initial condition while the continuous blue one represents the solution at different time steps.

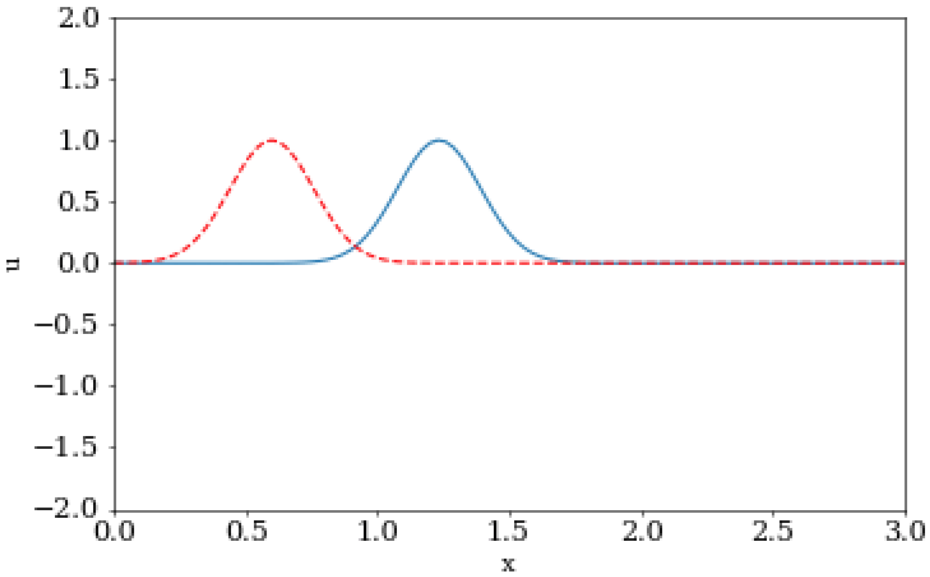


Figure 2.3: Spectral scheme applied to the 1D advection equation. The broken red line represents the initial condition while the continuous blue one represents the solution at different time steps

2.2.2 1D Burgers Equation

$$\partial_t u + u(t,x)\partial_x u = \nu \partial_{xx} u \tag{2.14}$$

This equation is a non linear PDE which combines, on the left, the non linear advective term and, on the right, the diffusive term. This equation has a certain degree of similarity to the the Navier-Stokes equations. It is a one dimensional equation describing the motion of a fluid characterised by non linear advective and diffusive terms with the absence of the driving force given by the pressure gradient. If the kinetic viscosity tends to zero then the equation becomes the inviscid Burgers' equation which can develop shock waves. As in the previous case an initial condition and a boundary layer condition are set. It should be remembered that spectral scheme does not require boundary conditions because they are considered implicitly in the definition of the initial condition which must be periodic.

In this case the domain length is 3, the number of grid points is 128 therefore the grid spacing is 0.023; the number of time steps is 256 for a duration of 4 seconds therefore the time stepping is 0.016. The initial condition considered is the same used in the previous case. The physical viscosity considered is $\nu = 0.1$.

Upwind Scheme

As in the previous case the upwind scheme for this equation is :

$$\frac{u_i^{n+1} - u_i^N}{\Delta t} + u_i^n \frac{u_i^n - u_{i-1}^n}{\Delta x} = \nu \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \tag{2.15}$$

and it can be written in its matrix form as

$$\bar{u}^{n+1} = -\bar{u}^n (FD \cdot \bar{u}^n) + \nu (SD \cdot \bar{u}^n) \tag{2.16}$$

(FD) stands again for First Derivative while (SD) stands for Second Derivative and they both are matrices representing the multiplying coefficients each term approximating the respective derivative. The first one is a bi-diagonal matrix characterised by zeros everywhere except the main and the immediate lower diagonal while the second one is three-diagonal one characterised by zeros everywhere except the main and the immediate upper and lower diagonals. Then the coefficients multiplying the boundary conditions are added in their respective position. Thus the FD matrix has 1 on the main diagonal and -1 on its immediate lower one, while SD matrix has -2 on its main diagonal and 1 on its immediate upper and lower co-diagonal. The code is shown in the appendix while the results of the application of the numerical method is displayed in figure 2.4.

Spectral Scheme

As in the previous case the following solutions is considered:

$$u(t,x) = \sum_{n=1}^{N} \hat{u}_n(t) e^{ik_n x} \tag{2.17}$$

As stated in the algorithm, after performing a FFT the derivatives are calculated in the frequency domain as follows:

$$\partial_x \hat{u} = ik_x \hat{u} \tag{2.18}$$

$$\partial_{xx} \hat{u} = (ik_x)^2 \hat{u} \tag{2.19}$$

After performing the iFFT of the equations above the right hand side is computed as

$$\partial_t u = -u\partial_x u + \nu \partial_{xx} u \tag{2.20}$$

Now the integration in time is performed by the *"solve ivp"* command which uses the default Runge-Kutta method. The visual result is shown in the figure 2.5.

Comparison

The solution of the 1D Burgers equation solved with the upwind scheme is diffused way quicker with respect to the spectral one because the upwind method is characterised by the physical, $\nu = 0.1$, and numerical diffusion while the spectral scheme is affected by only the physical one. In order to see the magnitude of the numerical dispersion affecting the upwind scheme the physical diffusion is considered null while the 1D Burgers equation is solved by the upwind scheme while it maintain its value, $\nu = 0.1$ when the equation is solved by the spectral scheme.

In figures 2.4 and 2.5 the initial condition, represented by the broken red line, and a snapshot of the numerical solution, represented by the continuous blue one, after $t = 1[s]$ are

shown.

Figure 2.4 shows the behaviour of inviscid Burgers equation which is able, as stated before, to represent a shock wave. The equation should display the formation of this discontinuity while moving in the right direction. However the numerical diffusion is considerably high, therefore while the discontinuity travels it keeps being diffused, decreasing therefore the accuracy of the numerical solution over time.

Figure 2.5 displays the proper behaviour of the equation 2.14 which accounts for a 0.1 physical diffusion. The scheme is extremely stable and present a high accuracy over time.

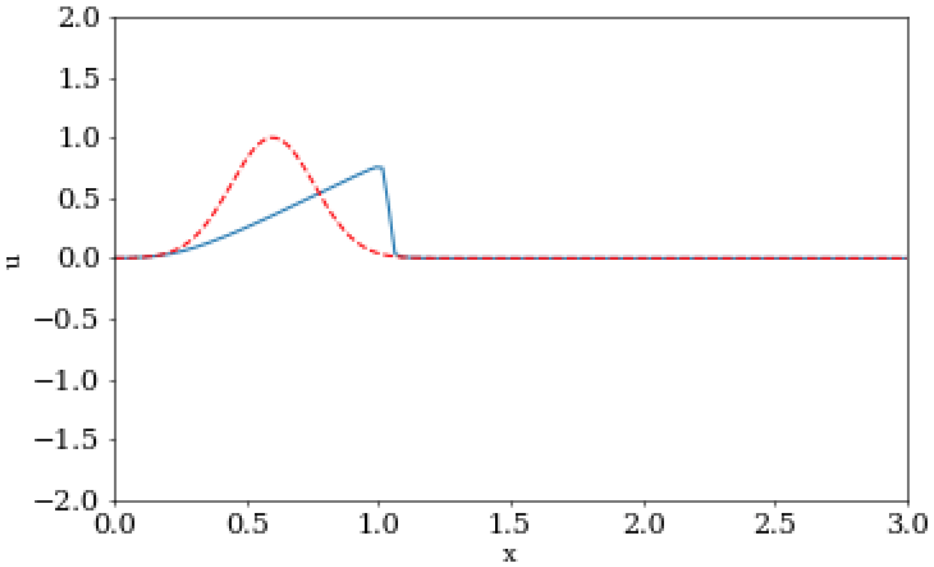


Figure 2.4: Upwind scheme applied to the 1D Burgers Equation. Broken red line is the initial condition while the continuous blue one is the numerical solution at each time step. The snapshot represent the solution after $t = 1[s]$

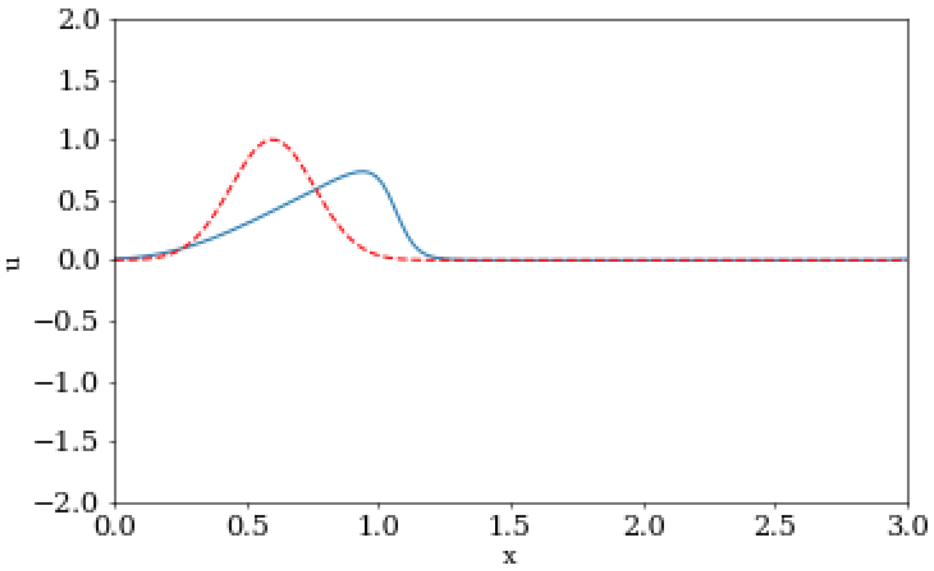


Figure 2.5: Spectral Scheme applied to the 1D Burgers Equation. Broken red line is the initial condition while the continuous blue one is the numerical solution at each time step. The snapshot represent the solution after $t = 1[s]$

2.2.3 2D Advection Equation

$$\partial_t u + 3\partial_x u + 2\partial_y u = \nu(\partial_{xx} u + \partial_{yy} u) \tag{2.21}$$

As explained before, this equation represent the transportation of substance and its properties by bulk motion of a fluid using a two dimensional scalar field $u(x,y,t)$. This equation and the following ones will not be solved by upwind method but will be solved directly by a spectral one. Moreover they are implemented with a semi-implicit spectral scheme which will demonstrate the ease with which the nonlinear equation can be made implicit.

After the definition of space and time domains the initial condition is defined as the following Gaussian

$$u(x,y,0) = e^{-\frac{(x-1)^2+(y-1)^2}{2\sigma^2}} \tag{2.22}$$

where σ is equivalent to 0.25.

Here there are two ways of proceeding in solving the PDE: convert the discretized two dimensional domain into a single vector of ordered grid points and therefore working with vectors or convert the outputs of the various functions used to arrays. The mode used in this thesis is the second one because of the suitability of regular domain, their regularity, generality and clarity of the procedure and the efficiency of the computations involved. The algorithm for spectral scheme is the same used until now with minor changes due to the increase of degree of freedom; the code is reported in the appendix.

To resolve in time this equation a partial implicit-explicit time scheme was chosen where the diffusive term is treated implicitly while the convective one is solved explicitly.

Once the discretisations have been realised, an initial condition has been defined and the wave numbers has been computed then the cycle iteration starts with a FFT performed on the initial condition. The solution at the next time step, in the frequency domain, is computed as stated below:

$$\hat{u}^{n+1} = \frac{\hat{u}^n(\frac{1}{\Delta t} - 3jk_x - 2jk_y)}{\frac{1}{\Delta t} + \nu(k_x^2 + k_y^2)} \tag{2.23}$$

Performing the iFFT gives the solution in the spatial domain which is then displayed and saved; this end the cyclical iteration. To solve this equation using a partially implicit scheme, non recurring to spectral methods, a non linear equation, whose formulation is more difficult that a simple fraction, must be solved at each iteration.

The following figures show the solution of the bi-dimensional advection equation at two different time steps. The nature of the equation makes the solution translate in the *xy* direction while being subjected to diffusion. The nature of the diffusion is only physical. It is clearly

shown above as the semi-implicit method is expressed simply as a fraction and therefore it is extremely easy to implement.

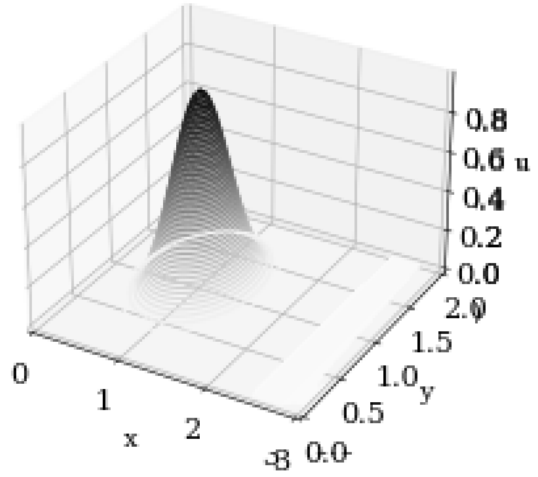


Figure 2.6: Spectral Scheme of the 2D Advection Equation

2.2.4 2D Burgers Equation

$$\begin{cases} \partial_t u + u\partial_x u + v\partial_y u = \nu(\partial_{xx}u + \partial_{yy}u) \\ \partial_t v + u\partial_x v + v\partial_y v = \nu(\partial_{xx}v + \partial_{yy}v) \end{cases} \tag{2.24}$$

As stated before this set of equations mimic the Navier-Stokes equations through its fluidlike expressions for nonlinear advection and diffusion terms. Again, to solve the system of equations the fundamental algorithm presented above is used. The initial condition used in this case is the same used in the other cases therefore:

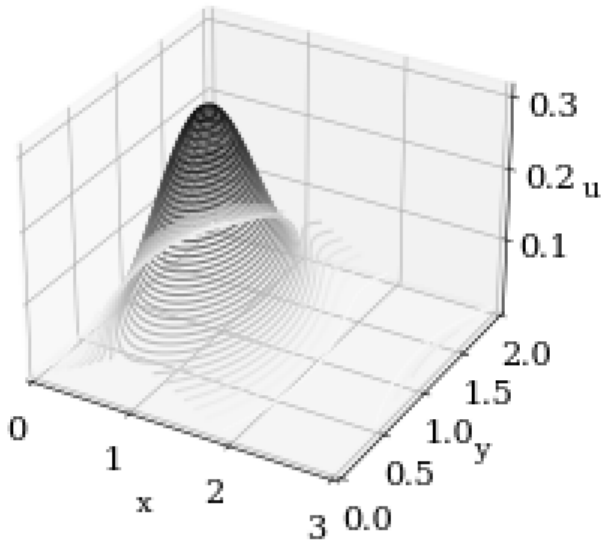


Figure 2.7: Spectral Scheme of the 2D Advection Equation

$$\begin{cases} u(x,y,0) = u_0(x,y) \\ v(x,y,0) = v_0(x,y) \end{cases} \tag{2.25}$$

The main part of the algorithm coded for the computation of this system of equations is presented in the appendix.

The system is solved using two different methods: a fully explicit and a partially explicit-implicit method for spatial discretization. Then the solution at the next time step is computed using Euler explicit method.

Explicit Method

Firstly it is needed to define the domain in time and space then the initial condition should be stated. After that the FFT of the initial condition is performed and wave numbers are computed. Next the derivatives are calculated in the frequency domain as stated in the algorithm presented in the section above. Follows the inverse Fast Fourier transformation on the derivatives calculated before, then the following system is computed:

$$\begin{cases} \frac{u^{n+1}-u^n}{\Delta t} = -u^n \partial_x u^n - v^n \partial_y u^n + \nu(\partial_{xx} u^n + \partial_{yy} u^n) \\ \frac{v^{n+1}-v^n}{\Delta t} = -u^n \partial_x v^n - v^n \partial_y v^n + \nu(\partial_{xx} v^n + \partial_{yy} v^n) \end{cases} \tag{2.26}$$

And lastly the next time step is calculated using the Explicit Euler method therefore:

$$\begin{cases} u^{n+1} = u^n + \Delta t(-u^n \partial_x u^n - v^n \partial_y u^n + \nu(\partial_{xx} u^n + \partial yy u^n)) \\ v^{n+1} = v^n + \Delta t(-u^n \partial_x v^n - v^n \partial_y v^n + \nu(\partial_{xx} v^n + \partial_{yy} v^n)) \end{cases} \tag{2.27}$$

After 700 time steps the solution has the following form:

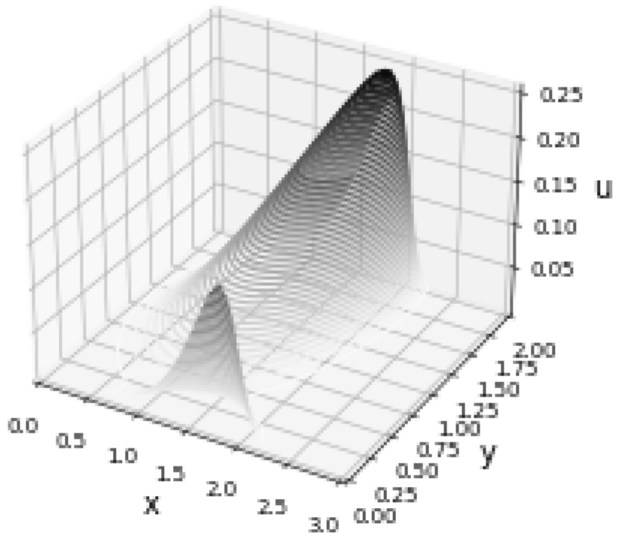


Figure 2.8: Fully explicit Spectral Scheme of the 2D Burgers Equations

The physical diffusivity considered is 0.01 which is extremely low therefore the system can be considered inviscid. As it has been seen before the inviscid Burgers equations presents a shock wave that travels in the *xy* direction. In fact the time step considered has capture the instant in which the wave has re-entered the domain; this basically shows that in spectral schemes the space is infinite.

This method is stable and present a satisfactory accuracy because a suitable time step, 0.005, is considered. It should be remembered that the spectral methods are used because of their low computational cost therefore in order to take advantage of this strength point a suitable method for time integration is required. Explicit Euler method is extremely easy to apply but it is highly unstable and therefore it requires a high temporal accuracy, which means a smaller time step. Summarising then, the computational cost saved using the spectral method

is less effective due to the cost requested for the implementation of Euler explicit method.

Implicit - Explicit Method

This method is different from the algorithm used above only for the discretization of the system of the equations, in particular, the diffusive term is computed explicitly while the convective one is computed implicitly.

This is again a practical example of what has been said at the beginning of this chapter, which is in regard to the easiness of the mathematical treatment needed to be performed in order to make implicit some term of the non linear equation. It is shown in this test case where Burgers equations once transformed into the frequency domain by a FFT becomes easily manageable and the implicitation of the non linear term becomes extremely easy. This therefore leads to a significant simplification of the final relations.

The system 2.24 is discretized as follows:

$$\begin{cases} \partial_t u = -u^n \partial_x u^{n+1} - v^n \partial_y u^{n+1} + \nu(\partial_{xx} u^n + \partial_{yy} u_{n)} \\ \partial_t v = -u^n \partial_x v^{n+1} - v^n \partial_y v^{n+1} + \nu(\partial_{xx} v^n + \partial_{yy} v^n) \end{cases} \tag{2.28}$$

performing a FFT of the Right Hand Side the system becomes

$$\begin{cases} \frac{u^{n+1}-u^n}{\Delta t} = -\hat{u}^n i k_x \hat{u}^{n+1} - \hat{v}^n i k_y \hat{u}^{n+1} + \nu((ik_x)^2 \hat{u}^n + (ik_y)\hat{u}_{n)} \\ \frac{v^{n+1}-v^n}{\Delta t} = -\hat{u}^n i k_x \hat{v}^{n+1} - \hat{v}^n i k_y \hat{v}^{n+1} + \nu((ik_x)^2 \hat{v}^n + (ik_y)^2 \hat{v}^n) \end{cases} \tag{2.29}$$

Using the commutative property on the non linear terms, executing an iFFT and re-arranging the system, it becomes

$$\begin{cases} u^{n+1}\left(\frac{1}{\Delta t} + \partial_x u^n + \partial_y v^n\right) = \frac{u^n}{\Delta t} + \nu(\partial_{xx} u^n + \partial_{yy} u^n) \\ v^{n+1}\left(\frac{1}{\Delta t} + \partial_x u^n + \partial_y v^n\right) = \frac{v^n}{\Delta t} + \nu(\partial_{xx} v^n + \partial_{yy} v^n) \end{cases} \tag{2.30}$$

Therefore the solution of the next time step is:

$$\begin{cases} u^{n+1} = \frac{\frac{u^n}{\Delta t} + \nu(u_{xx} + u_{yy})}{\frac{1}{\Delta t} + u_x + v_y} \\ v^{n+1} = \frac{\frac{v^n}{\Delta t} + \nu(v_{xx} + v_{yy})}{\frac{1}{\Delta t} + u_x + v_y} \end{cases} \tag{2.31}$$

The solution after 700 time steps is shown in the figure 2.9.

The physical diffusivity considered is 0.01 which means that the system can be considered inviscid. There are no changes in the physical behaviour with respect to the previous case.

The main difference is shown in the numerical method; the partially implicit-explicit method even though is more complex is more efficient so it is faster even though it is not appreciable in this particular case because the time integration is still computed utilising Euler explicit method which present the same problem expressed above.

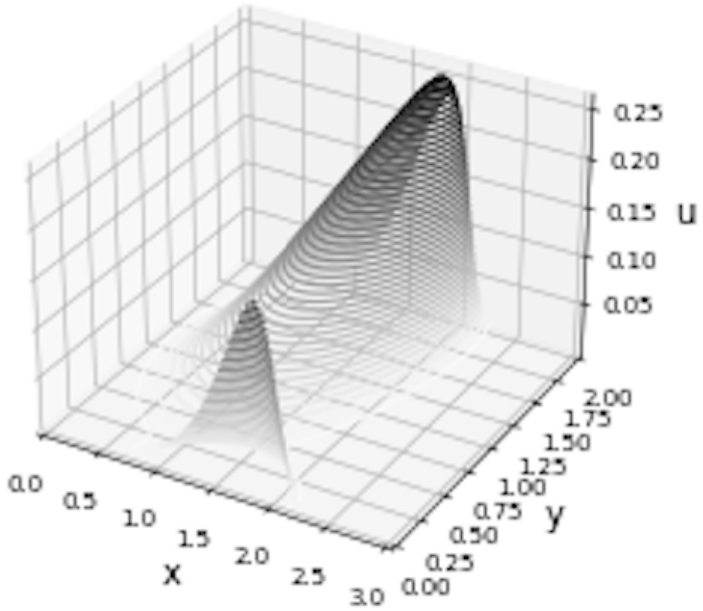


Figure 2.9: Implicit-Explicit Spectral Scheme of the 2D Burgers Equations

CHAPTER

3 Jet Wiping Equations

As presented in the abstract and sub sequentially in the introduction an efficient coating technique is the jet wiping process which uses impinging gas jet to control the thickness of a liquid layer dragged along a moving metal strip. However it is affected by instabilities born from the interaction between the gas jet and the liquid film create wavy pattern in the coating which results in a worsening homogeneity of the final product.

To understand the dynamic of the wave formation the classic laminar boundary layer models for falling films are extended to jet wiping problem. The models presented in this dissertation are known as integral boundary layer models, they are currently used to simulate the dynamic of the problem and they are extensively treated in [1], however, these are only a group of models, among others, studied which though resulted appropriate for the physical representation. Among such other models there are weighted integral boundary layer models (WIBL) and transition and turbulence boundary layer models (TTBL) used in order to extend the theory to large Reynolds numbers and analysing the impact of the modelling strategy. The mathematical theory behind the jet wiping phenomenon is well presented and extensively treated in [2] therefore hereby it will briefly explained.

3.1 Physical Problem

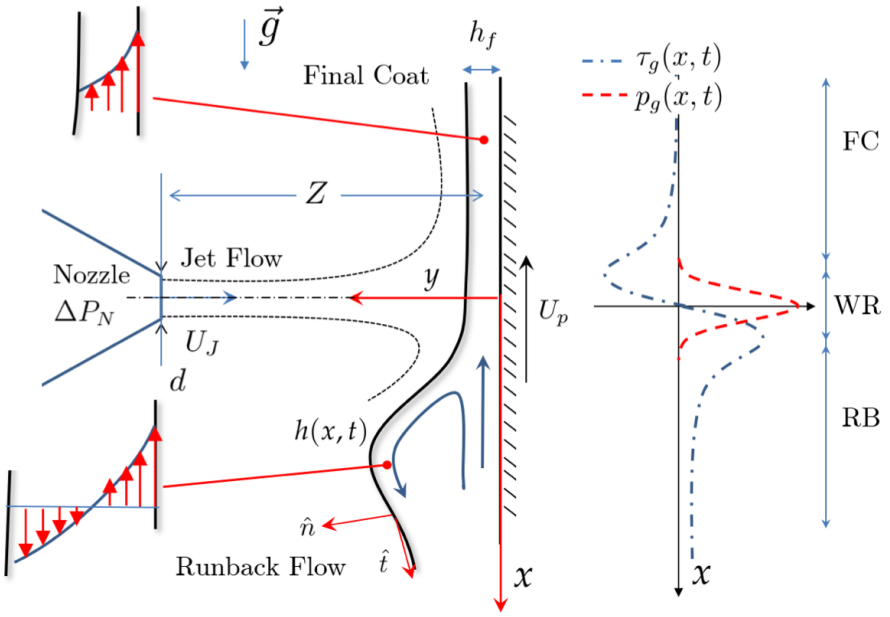


Figure 3.1: Schematic of the jet wiping process

The Jet Wiping Process is presented in the figure 3.1 where a metal strip moving upwards with constant velocity U_p drags a layer of liquid film which is impinged by a gas jet. The configuration is assumed two-dimensional, with incompressible liquid flow bounded by the plate at $y = 0$, and the dynamic liquid interface at $y = h(x,t)$.

As it is shown in the figure 3.1 the nozzle of the gas jet is positioned at $x = 0$ which extends downward counter the substrate velocity and along the gravitational acceleration. The impinging jet flow positioned at a standoff distance Z produces a pressure distribution, $p_g(x,t)$, and shear stress distribution, $\tau_g(x,t)$, that identify three different regions: Wiping region (WR), the RunBack flow region (RB) and the Final Coating region (FC). In the first region, WR, the pressure gradient forces part of the liquid film to reverse direction resulting in a wiping meniscus which then falls downward due to gravitational forces creating then RB region while

the thinner liquid film generated by the pressure gradient remain attached to the metal strip moving upward creating the FC region.

In a one-way coupling formulation, the assumption is that the presence of the liquid film does not have any impact on the gas jet, whereas the gas jet does affect the liquid film. This assumption has been extensively verified for predicting the average final coating thickness (Lacanette et al. 2006; Gosset Buchlin 2007). However, it is not capable of simulating the intricate interaction between the two flows studied by Gosset et al. (2019) and Mendez et al. (2019). The presented formulation, thoroughly addressed in reference [1], aims to decouple the behaviour of the liquid from that of the gas jet. It facilitates the analysis of the liquid film's frequency response and potential mechanisms of undulation formation. Under this formulation, it is assumed that both the pressure and shear stress generated by the jet depend solely on the nozzle gauge stagnation pressure (ΔP), the nozzle opening (d), the discharge coefficient (Cd), and the standoff distance (Z).

The physical problem of interest in not though two dimensional but three dimensional one therefore the problem of interest is presented in the figure below

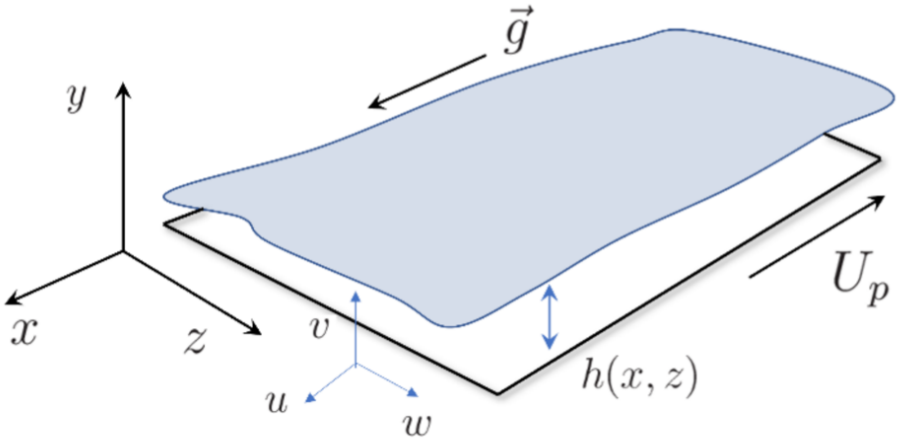


Figure 3.2: 3D Schematic of the jet wiping process

3.2 Long wave formulation

The integral models investigated rely on the Navier-Stokes equation, which in case of the jet wiping process are defined as follows :

$$\begin{cases} \partial_x u + \partial_y v = 0 \\ \rho_l(\partial_t u + u\partial_x u + v\partial_y u) = -\partial_x p_l + \mu_l(\partial_{xx} u + \partial_{yy} u) + \rho_l g \\ \rho_l(\partial_t v + u\partial_x v + v\partial_y v) = -\partial_y p_l + \mu_l(\partial_{xx} v + \partial_{yy} v) \end{cases} \tag{3.1}$$

And they rely on the boundary conditions in the "long wave" formulation. This formulation is derived by scaling the cross streamwise direction with a reference length $[h]$, which is much smaller than the streamwise reference length $[x]$.

3.2.1 Scaling laws

The scaling laws are defined following the physical problem and they transform the the Navier-Stokes equation into their dimensionless form. This section is discussed in both [1] and especially well in [7] where two different case are treated, in particular the falling film and the moving substrate, therefore hereby only the keys concept inherent to the moving substrate, topic of interest, are reported.

The liquid is assumed to be incompressible with kinematic viscosity ν, density ρ, dynamic viscosity $\mu = \rho\nu$ and surface tension σ. The problem is set as shown in the figure 3.2 where p denotes the pressure in the liquid while (u, v, w) are the velocity components. The flow is bounded at the surface of the metal strip while the dynamic liquid interface is at $y = h(x, z, t)$. The square brackets will denote the reference values while the hat will denote the scaling of a certain quantity respect its reference value and it is dimensionless, that is to say $\hat{a} = a/[a]$. As

stated before the substrate is moving at the imposed velocity U_p in the opposite direction to the gravity and therefore it is logical to assume that $[u] = U_p$ meanwhile the reference values for the thickness and the flow rate are respectively: $[h] = \sqrt{\nu U_p/g}$ and $[q] = [u][h] = \sqrt{\nu U_p^3/g}$. Accordingly the Reynolds number is defined as $Re = [q]/\nu = \sqrt{U_p^3/(g\nu)}$. As requested from the "long wave" formulation the x direction should be scaled such that $\varepsilon = [h]/x << 1$ meanwhile the capillary forces, $(\sim \sigma[h]/[x]^3)$, are similar to the gravitational ones, $(\sim \rho g)$. This scaling laws are known as Shkadov's scaling, leads to

$$\varepsilon = (\frac{[h]^2 g\rho}{\sigma})^{1/3} = Ca^{1/3} \tag{3.2}$$

where $Ca = \mu U_p/\sigma$ is the capillary number for the moving substrate. Finally the reduced Reynolds number,$\delta = \varepsilon Re$, and the dimensionless Kaptiza number which weighs the importance of surface tension over viscosity depending only on the liquid properties , $Ka = \sigma/(\rho\nu^{4/3}g^{1/3})$, are introduced. For the sake of clarity, simplicity and visibility the scaling laws are reported in the table below 3.1.

Reference Quantity	Definition	Expression
$[h]$	$(\nu_l[u]/g)^{1/2}$	$(\nu_l U_p/g)^{1/2}$
$[x]$	$[h]/\varepsilon$	$(\nu_l U_p/g)^{1/2} Ca^{-1/3}$
$[u]$	U_p	U_p
$[v]$	εU_p	$U_p Ca^{1/3}$
$[p]$	$\rho_l g[x]$	$(\mu_l \rho_l g U_p)^{1/2} Ca^{-1/3}$
$[\tau]$	$\mu_l[u]/[h]$	$(\mu_l \rho_l g U_p)^{1/2}$
$[t]$	$[x]/[u]$	$(\nu_l/U_p g)^{1/2} Ca^{-1/3}$

Table 3.1: Shkadov's Scaling laws

3.3 Integral Boundary Layer Models

As stated before the integral boundary layer models can be derived from the Navier-Stokes equations reported in 3.1 scaled accordingly to the scaling laws presented in the previous section. The boundary layer equations obtained are:

$$\partial_{\hat{x}}\hat{u} + \partial_{\hat{y}}\hat{v} + \partial_{\hat{z}}\hat{w} = 0 \tag{3.3}$$

$$\delta(\partial_{\hat{t}}\hat{u} + \hat{u}\partial_{\hat{x}}\hat{u} + \hat{v}\partial_{\hat{y}}\hat{u} + \hat{w}\partial_{\hat{z}}\hat{u}) = -\partial_{\hat{x}}\hat{p}_x + \partial^2_{\hat{y}\hat{y}}\hat{u} + 1 \tag{3.4}$$

$$\partial_{\hat{y}}\hat{p}_y = 0 \tag{3.5}$$

$$\delta(\partial_{\hat{t}}\hat{w} + \hat{u}\partial_{\hat{x}}\hat{w} + \hat{v}\partial_{\hat{y}}\hat{w} + \hat{w}\partial_{\hat{z}}\hat{w}) = -\partial_{\hat{z}}\hat{p}_z + \partial^2_{\hat{y}\hat{y}}\hat{w} \tag{3.6}$$

Where the equation 3.3 is the continuity equation while the other three, respectively 3.4,3.5 and 3.6, are the (x, y, z) momentum equations. For the sake of completeness below are stated the boundary conditions, however it is needed to be remembered that the spectral methods do not require them since they are implicitly verified in the initial condition.

$$\overrightarrow{\hat{\mathrm{v}}}\,|_{\hat{y}=0} = (\hat{u}, \hat{v}, \hat{w})|_{\hat{y}=0} = (-1, 0, 0) \tag{3.7}$$

$$\hat{v}|_{\hat{y}=h} = \partial_{\hat{t}}\hat{h} + \hat{u}|_{\hat{y}=h}\partial_{\hat{x}}\hat{h} + \hat{w}|_{\hat{y}=h}\partial_{\hat{z}}\hat{h} \tag{3.8}$$

The dynamic boundary conditions formulating the force balance at the free surface is:

$$\hat{p}|_{\hat{y}=\hat{h}} = \hat{p}_g - (\partial_{\hat{x}\hat{x}}\hat{h} + \partial_{\hat{z}\hat{z}}\hat{h}) \tag{3.9}$$

$$\partial_{\hat{y}}\hat{u}|_{\hat{y}=\hat{h}} = \hat{\tau}_{g,x} \tag{3.10}$$

$$\partial_{\hat{y}}\hat{w}|_{\hat{y}=\hat{h}} = \hat{\tau}_{g,z} \tag{3.11}$$

where $\hat{p}_g$, $\hat{\tau}_{g,x}$ and $\hat{\tau}_{g,z}$ are respectively gas pressure and the shear stress components along x and z respectively, imposed by an external air flow.

To derive the integral model the equations from 3.3 to 3.6 are integrated along y assuming a self similar parabolic velocity profile for both the streamwise $\hat{u}$ and spanwise $\hat{w}$ velocity components. Using the local flow rate definitions, the substrate motion and the interface shear stress, the profiles for the MS case read:

$$\hat{u}(\hat{h},\hat{q}_x,\hat{q}_z) = \frac{3}{4\hat{h}^3}(\hat{\tau}_{g,x}\hat{h}^2 - 2\hat{h} - 2\hat{q}_x)\hat{y}^2 + \frac{6\hat{h} + 6\hat{q}_x - \hat{\tau}_{g,x}\hat{h}^2}{2\hat{h}^2}\hat{y} - 1 \tag{3.12}$$

$$\hat{w}(\hat{h},\hat{q}_x,\hat{q}_z) = \frac{3}{4\hat{h}^3}(\hat{\tau}_{g,z}\hat{h}^2 - 2\hat{q}_z)\hat{y}^2 + \frac{6\hat{q}_z - \hat{\tau}_{g,z}\hat{h}^2}{2\hat{h}^2}\hat{y} \tag{3.13}$$

It is relevant that the balance of the viscosity and gravitational forces are not altered by inertial and surface tension. The integration results in a system of nonlinear partial differential equations for the liquid film height $\hat{h}$, the streamwise $\hat{q}_x$ and spanwise $\hat{q}_z$ flow rates. In conservative form, this reads:

$$\partial_{\hat{t}}\hat{U} + \bigtriangledown \cdot \mathbf{F} = \overrightarrow{\mathrm{S}} \tag{3.14}$$

where the state vector is defined as $\hat{U} = (\hat{h}, \hat{q}_x, \hat{q}_z)^T$, the source vector is $\overrightarrow{\mathrm{S}} = (S_1, S_2, S_3)$ and **F** is the rectangular flux matrix of dimension $2X3$. In particular The sources terms are defined as :

$$\begin{cases} S_1 = 0 \\ S_2 = \frac{1}{\delta}(\hat{h}(\partial_{\hat{x}}\hat{p}_x + \partial^3_{\hat{x}\hat{x}\hat{x}}\hat{h} + \partial^3_{\hat{x}\hat{z}\hat{z}}\hat{h} + 1) + \Delta\hat{\tau}_x) \\ S_3 = \frac{1}{\delta}(\hat{h}(\partial_{\hat{z}}\hat{p}_z + \partial^3_{\hat{z}\hat{z}\hat{z}}\hat{h} + \partial^3_{\hat{z}\hat{x}\hat{x}}\hat{h}) + \Delta\hat{\tau}_z) \end{cases} \tag{3.15}$$

the third derivatives correspond to capillary pressure gradient while the terms $\Delta\hat{\tau}$s are obtained from the integration of equations 3.4 and 3.6 and they corresponds to the difference between shear stresses at interface of the liquid film and the shear stresses at the wall. Using the self similar assumption the shear stresses at the wall are given by the following formulations:

$$\begin{cases} \hat{\tau}_{w,x} = \frac{1}{2}\hat{\tau}_{g,x} - 3\frac{\hat{q}_x}{\hat{h}^2} - \frac{3}{\hat{h}} \\ \hat{\tau}_{w,z} = \frac{1}{2}\hat{\tau}_{g,z} - 3\frac{\hat{q}_z}{\hat{h}^2} \end{cases} \tag{3.16}$$

meanwhile the flux matrix is

$$\mathbf{F} = \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \end{bmatrix} \tag{3.17}$$

where every element it is as follows :

$$
\begin{cases}
F_{11} = \hat{q}_x \\
F_{21} = \hat{q}_z \\
F_{12} = \frac{1}{120\hat{h}}(144\hat{q}_x^2 + 6\hat{\tau}_{g,x}\hat{h}^2\hat{q}_x + \hat{\tau}_{g,x}\hat{h}^4 + 48\hat{h}\hat{q}_x + 6\hat{\tau}_{g,x}\hat{h}^3 + 24\hat{h}^2) \\
F_{22} = \frac{1}{120\hat{h}}(144\hat{q}_x\hat{q}_z + 3\hat{\tau}_{g,x}\hat{h}^2\hat{q}_z + 3\hat{\tau}_{g,z}\hat{h}^2\hat{q}_x + \hat{\tau}_{g,x}\hat{\tau}_{g,z}\hat{h}^4 + 24\hat{h}\hat{q}_z + 3\hat{\tau}_{g,z}\hat{h}^3) \\
F_{13} = F_{22} \\
F_{23} = \frac{144\hat{q}_z^2 + 3\hat{\tau}_{g,z}\hat{h}^2\hat{q}_z + \hat{\tau}_{g,z}^2\hat{h}^2}{120\hat{h}}
\end{cases}
\tag{3.18}
$$

This concludes the introduction to the integral boundary layer model. For a deeper understanding the reader is redirected to [7] and [1].

CHAPTER

4 BLEW in Python

The studies regarding the realisation of a mathematical model that describes the jet wiping process and the formation of instabilities have already taken more that 10 years during which studies, experiments and general research have been carried out extensively in order to formulate a numerical model able to predict the thickness of the final coating and control the generation of instabilities.

The work done until today has culminated in the realisation of the software Boundary LayEr Wiping, BLEW. Therefore the first phase of the research can be considered finished since the problem has been understood and model implementation has already been carried out.

Henceforth an optimisation problem shall be resolved since finite volume schemes present huge computational cost which limits the applicability of the software developed; so in order to make it industrially practical spectral methods are developed to reduce the time and increase the accuracy of the model.

4.1 General aspects of the BLEW software

Arcelor Mittal more than a decade ago proposed to the Von Karman Institute the study about an industrial optimisation process since the goal is to understand the dynamic of liquid film and the arise of instabilities in the coating processes in order to reduce costs and increase the quality of the goods.

After years of studies and experimentation to get the data needed to implement mathematical models a software in *Python* was realised. The software *"BLEW"* is a project made of different ".py" files; most of them are callable functions used in the main script. The choice of Python is based on some points which are: the easiness of usage, huge amount of packages already well developed, algorithms for artificial intelligence already developed, implemented and well documented and the flexibility given.

Firstly different packages are called and an environment is created then, on company requests, the inputs are passed by a file *Excel* linked to the main script. It takes in input three sheets of parameters respectively: *wiping*, *actuators*, *numerical* and *temporal* parameters; gas perturbation and *liquid perturbation*; and other parameters.

Secondly the interpolation matrix is computed and passed to the liquid perturbation function. This is based on the scalar laws presented above.

Thirdly the spatial and temporal domain are defined.

Next the solutions are initialised and the Reinforcement Learning algorithm is implemented. Within the *"RL"* algorithm the numerical scheme chosen is applied. The scheme used until now is the finite volume scheme which is based on the blended Lax-Wendroff (high-order) and Lax-Friedrichs (low-order) scheme, regulated by a *"limiter"* function, on a dimensionless integral model for a liquid film on a moving substrate developed by Gabriele Gamba and

presented in [2]. Immediately afterwards the boundary conditions are defined. Then the gas perturbations are implemented and used to obtain in return *observations* and *reward.*

Finally the environment is reset at the end of the episode.

4.2 Simplified BLEW equations

Since this is the first work of its kind the approach adopted is "bottom-up" therefore the first BLEW equations implemented are simplified; they are originated from the equations (3.14), characterised by (3.15) and (3.18), with the assumption of gas shear stresses, $\tau_{g_x} \quad \tau_{g_z}$, and pressure gradient, ∇p, null. If these conditions are applied while making explicit the fluxes (3.18) the following system of equations is obtained :

$$\begin{cases} \partial_t \hat{h} + \partial_x \hat{q}_x + \partial_y \hat{q}_z = 0 \\ \partial_t \hat{q}_x + \partial_x \frac{144\hat{q}_x^2 + 48\hat{h}\hat{q}_x + 24\hat{h}^2}{120\hat{h}} + \partial_z \frac{144\hat{q}_x\hat{q}_z + 24\hat{h}\hat{q}_z}{120\hat{h}} = S_2 \\ \partial_t \hat{q}_z + \partial_x \frac{144\hat{q}_x\hat{q}_z + 24\hat{h}\hat{q}_z}{120\hat{h}} + \partial_z \frac{144\hat{q}_z^2}{120\hat{h}} = S_3 \end{cases} \tag{4.1}$$

Using the derivative's properties the system can be re-written as

$$\begin{cases} \partial_t \hat{h} + \partial_x \hat{q}_x + \partial_y \hat{q}_z = 0 \\ \partial_t \hat{q}_x + \frac{24}{120}\left(6\partial_x \frac{\hat{q}_x^2}{\hat{h}} + 2\partial_x \hat{q}_x + \partial_x \hat{h} + 6\partial_z \frac{\hat{q}_x\hat{q}_z}{\hat{h}} + \partial_z \hat{q}_z\right) = S_2 \\ \partial_t \hat{q}_z + \frac{24}{144}\left(6\partial_x \frac{\hat{q}_x\hat{q}_z}{\hat{h}} + \partial_x \hat{q}_z + 6\partial_z \frac{\hat{q}_z^2}{\hat{h}}\right) = S_3 \end{cases} \tag{4.2}$$

Below are presented two different implementation of the system of equations (4.2): Explicit Euler, the basic implementation, and an implementation of a partially implicit-explicit scheme. Those are coded together in the last file presented in the appendix. In that file is also present an intermediate scheme used to improve the performances of the base solver however for the sake of clarity and brevity only the best and ultimate result is presented in this thesis

work.

To optimise the performance of the solver the Finite Impulse Response, FIR, filter theory is adopted for a low pass filtering process. In the filtering process some small oscillations are introduced near the extremes of the range of values filtered and this is done in order to reduce the numerical oscillations of the final solution.

4.2.1 Explicit Euler Scheme

The Explicit Euler scheme is the easiest scheme to implement and it is to be considered the base which is stable for $dt \leq dx/500$, where $1/500$ is the velocity value and it was empirically obtained. The system of equations reported above becomes:

$$\begin{cases} \partial_t \hat{h} = -\partial_x \hat{q}_x - \partial_z \hat{q}_z \\ \partial_t \hat{q}_x = S_2 - \frac{24}{120}\left(6\partial_x \frac{\hat{q}_x^2}{\hat{h}} + 2\partial_x \hat{q}_x + \partial_x \hat{h} + 6\partial_z \frac{\hat{q}_x \hat{q}_z}{\hat{h}} + \partial_z \hat{q}_z\right) \\ \partial_t \hat{q}_z = S_3 - \frac{24}{144}\left(6\partial_x \frac{\hat{q}_x \hat{q}_z}{\hat{h}} + \partial_x \hat{q}_z + 6\partial_z \frac{\hat{q}_z^2}{\hat{h}}\right) \end{cases} \tag{4.3}$$

The derivatives are computed as follows

- The Linear term : $\partial_i \hat{q}_j = iFFT(jk_i \hat{\hat{q}}_j)$ where $i, j = (x, z)$ and $\hat{\hat{q}}_j = FFT(\hat{q}_j)$
- The Non Linear term : if $a = \frac{\hat{q}_x^2}{\hat{h}}$, $b = \frac{\hat{q}_x \hat{q}_z}{\hat{h}}$ and $c = \frac{\hat{q}_z}{\hat{h}}$ then $\partial_i(a,b,c) = iFFT(jk_i(\hat{a},\hat{b},\hat{c}))$ in where $(\hat{a},\hat{b},\hat{c}) = FFT((a,b,c))$
- The temporal term : $\partial_t(\hat{h},\hat{q}_x,\hat{q}_z) = \frac{(\hat{h},\hat{q}_x,\hat{q}_z)^{k+1} - (\hat{h},\hat{q}_x,\hat{q})^k}{\Delta t}$

Therefore the final system to be implemented is

$$\begin{cases} \hat{h}^{k+1} = \hat{h}^k - \Delta t(\partial_x \hat{q}_x + \partial_z \hat{q}_z) \\ \hat{q}_x^{k+1} = \hat{q}_x^k + \Delta t \left(S_2 - \frac{24}{120} \left(6\partial_x \frac{\hat{q}_x^2}{\hat{h}} + 2\partial_x \hat{q}_x + \partial_x \hat{h} + 6\partial_z \frac{\hat{q}_x \hat{q}_z}{\hat{h}} + \partial_z \hat{q}_z \right) \right) \\ \hat{q}_z^{k+1} = \hat{q}_z^k + \Delta t \left(S_3 - \frac{24}{144} \left(6\partial_x \frac{\hat{q}_x \hat{q}_z}{\hat{h}} + \partial_x \hat{q}_z + 6\partial_z \frac{\hat{q}_z^2}{\hat{h}} \right) \right) \end{cases} \tag{4.4}$$

The main problem encountered is related to the definition of a wrong initial condition. The initial condition used at the beginning was defined as

$$\begin{cases} \hat{h} = e^{\frac{(x-x_0)^2+(z-z_0)^2}{2\sigma^2}} \\ \hat{q}_x = \frac{\hat{h}^3}{3} - \hat{h} \\ \hat{q}_z = 0 \end{cases} \tag{4.5}$$

This caused immediately the solution to overflow due to the many non linear term divided by h which was zero almost everywhere.

The problem is solved introducing a base state for the initial film height which basically avoided the division by zeros thus solving the problem of overflow. The initial condition used then are defined as :

$$\begin{cases} h = 0.5e^{\frac{(x-x_0)^2+(z-z_0)^2}{2\sigma^2}} + h_0 \ \text{ with } \ h_0 = 0.1 \\ \hat{q}_x = \frac{\hat{h}^3}{3} - \hat{h} \\ \hat{q}_z = 0 \end{cases} \tag{4.6}$$

Once the problem of overflow was solved different simulation with different spatial discretisation are done in order to verify the degree of the numerical dispersion and the computational time of the two schemes depending on the grid refinement.

Figures 4.1, 4.2 and 4.3 show the solution obtained by grid refinement in x and z direction at sequential times, in particular, at $t = 13, 26, 54\ [s]$. The grid refinement is defined as : $n_x = mL_x$

and $n_z = mL_z$, where m is a *multiplier* equal to $[1,2,3]$, while the spatial discretisation is : $dx = L_x/n_x$ and $dz = L_z/n_z$. The physical time is 60 $[s]$. Increasing the multiplier at each simulation increases the number of grid points which then reduce the grid spacing creating a more refined spatial discretisation.

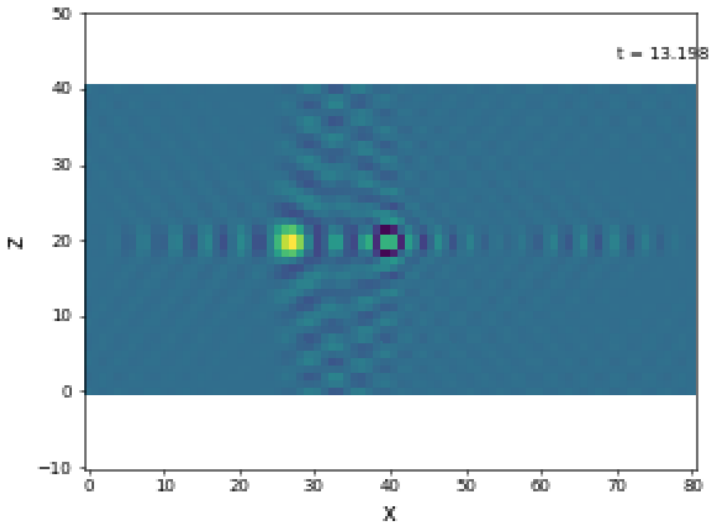


Figure 4.1: Solution of simplified BLEW equations characterised by $n_x = L_x$ and $n_z = L_z$ at 13 seconds of simulation

Figure 4.1 shows the solution characterised by a *multiplier* of 1 after 13 $[s]$. This means that the number of grid points is equal to the length of the domain in its respective direction. In order to obtain a simulation of 60 $[s]$ the solver has employed almost 197 $[s]$ of computational time.

The initial condition is travelling from right, starting at $(x,z) = (40,20)$ which is basically the center of the spatial domain, to the left and once it leaves the domain from the left edge it will re-enter from the right one. The waves generated from the moving Gaussian are visual representation of the numerical dispersion born from a poor spatial discretisation which leads to an inappropriate approximation of the solution across all the studied field.

Figure 4.2 shows the solution characterised by a *multiplier* of 2 after 26 $[s]$. Now the number of grid points is two times the respective domain's length. In order to obtain the same

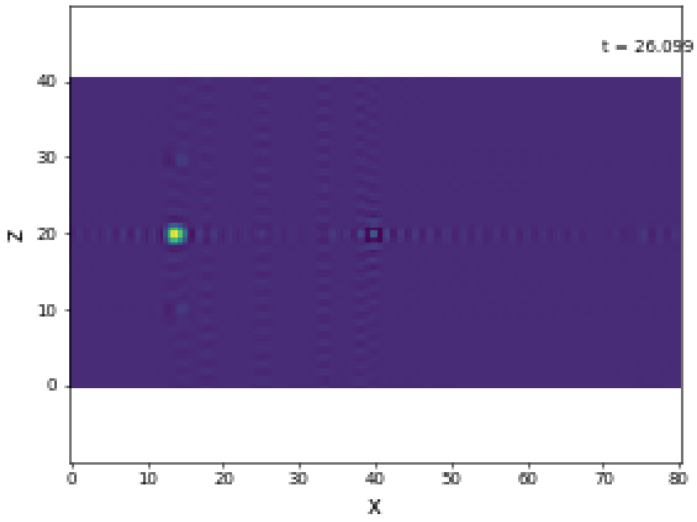


Figure 4.2: Solution of simplified BLEW equations characterised by $n_x = 2L_x$ and $n_z = 2L_z$ at 26 seconds of simulation

60 $[s]$ of simulation the solver has employed almost 806 $[s]$ of computational time. As it was said before the Gaussian is travelling from left to right but in this case since the grid is refined then the numerical dispersion, shown as waves propagating from the moving initial condition, is much less although it is still present and relevant.

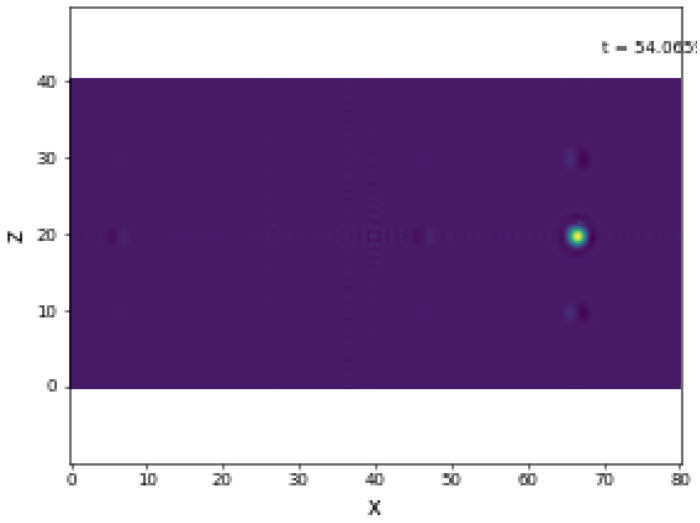


Figure 4.3: Solution of simplified BLEW equations characterised by $n_x = 3L_x$ and $n_z = 3L_z$ at 54 seconds of simulation

Lastly figure 4.3 reports the solution characterised by a *multiplier* of 3 after 54 $[s]$. This plot show the most refined grid used for the Explicit Euler scheme in fact the number of grid points is three times the respective domain's length. In this case the simulation lasted about 2410 $[s]$ which is basically three times slower that the previous case. Although the solution is much more accurate with almost no numerical dispersion, almost absolute absence of travelling waves, the computational time employed is no small amount therefore for practical uses a trade off needs to be found.

The solver works properly however it is not conveniently applicable to real cases because of its high computational cost. To improve the solution and reduce the computational cost a partially implicit-explicit scheme is introduced in the sub-section below.

4.2.2 Partially Implicit - Explicit Scheme

To solve the problem of the high computational cost of the Explicit-Euler scheme a partially implicit-explicit scheme is introduced. This scheme, as it will be shown below, work properly for $dt = dx/50$ which means that it is 10 time faster than the explicit-Euler without relevant changes in the solution stability.

As stated in the title the partially implicit-explicit scheme computes some terms explicitly which means it uses the solution of the variable at the current time step, while other are treated implicitly which means that they are unknown. From the system of equations (4.2) the non linear terms in the flow rate equations along x and z can be rewritten, using the derivatives' properties, as

- $\partial_i \frac{\hat{q}_j^2}{\hat{h}} = \frac{2\hat{q}_j}{\hat{h}} \partial_i \hat{q}_j + \left(\frac{\hat{q}_j}{\hat{h}}\right)^2 \partial_i \hat{h}$ where $i, j = (x, z)$
- In case of the flow rate equation along x the non linear term $\frac{\hat{q}_x \hat{q}_z}{\hat{h}}$ can be written as $a * \hat{q}_x$ where $a = \frac{\hat{q}_z}{\hat{h}}$

- Meanwhile in case of the flow rate equation along z a new variable, $b = \frac{\hat{q}_x}{\hat{h}}$, can be defined therefore $\frac{\hat{q}_x \hat{q}_z}{\hat{h}} = b * \hat{q}_z$.

Hence the system becomes as stated below

$$\begin{cases} \partial_t \hat{h} + \partial_x \hat{q}_x + \partial_y \hat{q}_z = 0 \\ \partial_t \hat{q}_x + \frac{24}{120}\left(6\left(2b\partial_x \hat{q}_x + b^2 \partial_x \hat{h}\right) + 2\partial_x \hat{q}_x + \partial_x \hat{h} + 6\left(\hat{q}_x \partial_z a + a\partial_z \hat{q}_x\right) + \partial_z \hat{q}_z\right) = S_2 \\ \partial_t \hat{q}_z + \frac{24}{120}\left(6\left(\hat{q}_z \partial_x b + b\partial_x \hat{q}_z\right) + \partial_x \hat{q}_z + 6\left(2a\partial_z \hat{q}_z - a^2 \partial_z \hat{h}\right)\right) = S_3 \end{cases} \tag{4.7}$$

And then

$$\begin{cases} \partial_t \hat{h} + \partial_x \hat{q}_x + \partial_y \hat{q}_z = 0 \\ \partial_t \hat{q}_x + \frac{24}{120} 6\left(2b\partial_x \hat{q}_x + a\partial_z \hat{q}_x + \hat{q}_x \partial_z a\right) = S_2 - \frac{24}{120}\left(\partial_x \hat{h}(6b^2 + 1) + 2\partial_x \hat{q}_x + \partial_z \hat{q}_z\right) \\ \partial_t \hat{q}_z + \frac{24}{120} 6\left(2a\partial_z \hat{q}_z + b\partial_x \hat{q}_z + \hat{q}_z \partial_x b\right) = S_3 + \frac{24}{120}\left(a^2 \partial_z \hat{h} - \partial_x \hat{q}_z\right) \end{cases} \tag{4.8}$$

Now the following procedure can be performed to modify each equation in the most suitable way :

- Every term should be passed from the spatial domain to the frequency domain using the FFT.
- Rewrite derivative terms as $\partial_i(\hat{h}, \hat{q}_j) = ik_i(\hat{\hat{h}}, \hat{\hat{q}}_j)$ with $i, j = (x, z)$
- Now that every term is in the frequency domain, the derivatives are written as stated above then the commutative property can be used on suitable targets. In this case the variable that will be treated implicitly is the $\hat{q}_j$ of the non linear term of the respective flow rate equation.

This basically means that in case of the x direction the term treated implicitly will be the variable $\hat{q}_x$ being part of a non linear term meanwhile in the z direction it will be $\hat{q}_z$ part of the non linear term. This practically means that :

$$\hat{q}_x\partial_x a + a\partial_x\hat{q}_x = iFFT(\hat{\hat{q}}_x^{k+1}(ik_x\hat{a}^k) + \hat{a}^k(ik_x\hat{\hat{q}}_x^{k+1})) = iFFT(2\hat{\hat{q}}_x^{k+1}(ik_x\hat{a})) \tag{4.9}$$

- Finally every term must be switched back to the spatial domain.
  As in the previous example the terms that will be switched back are :$\hat{q}_x^{k+1} = iFFT(\hat{\hat{q}}_x^{k+1})$ and $a^k = iFFT(ik_x\hat{a}^k)$

Therefore the final system implemented in the code states

$$\begin{cases} \hat{h}^{k+1} = \hat{h}^k - \Delta(\partial_x\hat{q}_x^k + \partial_z\hat{q}_z^k) \\ \hat{q}_x^{k+1} = \frac{S_2 - \frac{24}{120}\left(\partial_x h(6b^{2^k}+1)\hat{+}2\partial_x\hat{q}_x^k + \partial_z\hat{q}_z^k\right)}{\frac{1}{\Delta t} + \frac{12}{5}(\partial_x b^k + \partial_z a^k)} \\ \hat{q}_z^{k+1} = \frac{S_3 + \frac{24}{120}\left(a^{2^k}\partial_z\hat{h}^k - \partial_x\hat{q}_z^k\right)}{\Delta t + \frac{12}{5}(\partial_x b^k + \partial_z a^k)} \end{cases} \tag{4.10}$$

This practical case is the perfect example of the flexibility of the spectral schemes. In order to use other implicit schemes a complex system of non linear equations needs to be generated which then needs to be solved every time step in order to find the solution of the next one; however the spectral schemes give more freedom in the non linear term management which then simplify greatly the mathematical treatment which leads to generate easier relations to implement, in fact what would have been a system of non linear equations become a system of three fractions with a certain degree of similarity.

Moreover the code shows a third scheme which is however an intermediate step. It is also a partially implicit-explicit scheme characterised by a different choice of implicit variable. The implicit variable was chosen based on the flow rate equation that was solved and it was

considered only from the non linear term : $\hat{q}_x\hat{q}_z/h$.

The implementation of this algorithm is shown in the appendix and it is the last proposed to the reader while the results are reported in : 4.4, 4.5, 4.6 and 4.7 below. As in the previous cases the number of grid points is equal to the length of the respective domain times a *multiplier* value, m, which is equal to $(1,2,3,4,5)$ therefore it can be summarised that $dx = dz = 1/m$ thus increasing the *multiplier*, m, refines the mesh. The physical time is, again, the same 60 $[s]$.

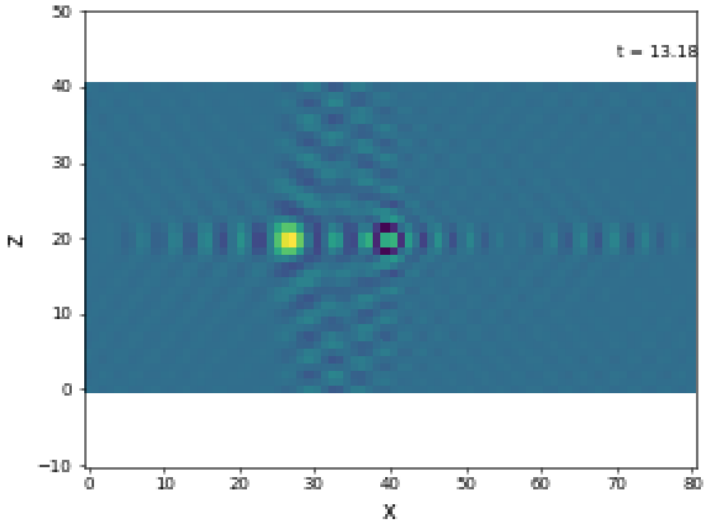


Figure 4.4: Solution of simplified BLEW equations characterised by $m = 1$ at 13 seconds of simulation

As in the corresponding Explicit Euler case, here, at the same physical time the figure 4.4 report the solution characterised by a multiplier of one which means that the grid is extremely coarse and therefore the approximation of the solution at the common surface of two adjacent cells is inappropriate which is being shown as wave propagation from the moving Gaussian. Although the solution is affected by high numerical dispersion the computational time employed by the solver is just 88 $[s]$ which is already faster that the corresponding test case.

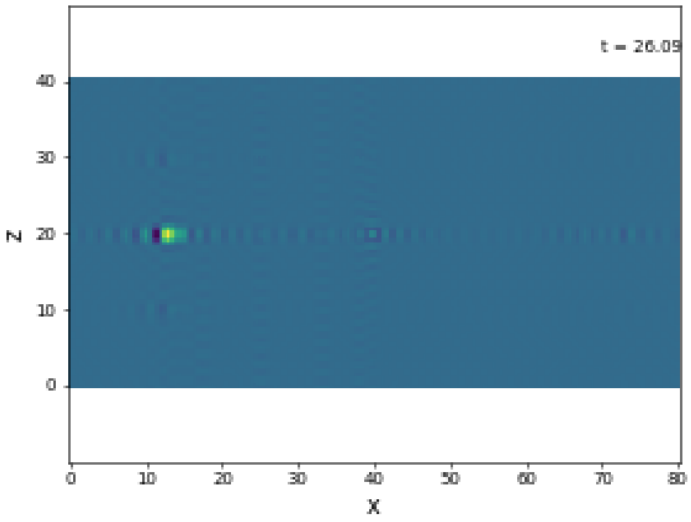


Figure 4.5: Solution of simplified BLEW equations characterised by $m = 2$ at 26 seconds of simulation

Figure 4.5 represents the solution at a physical time of 26 $[s]$ and a number of grid points characterised by a *multiplier*, $m = 2$, therefore the grid spacing is half of the previous one thus improving the spatial discretisation. The computational time employed for this simulation is 258 $[s]$. The numerical dispersion is greatly reduced even though it is still present

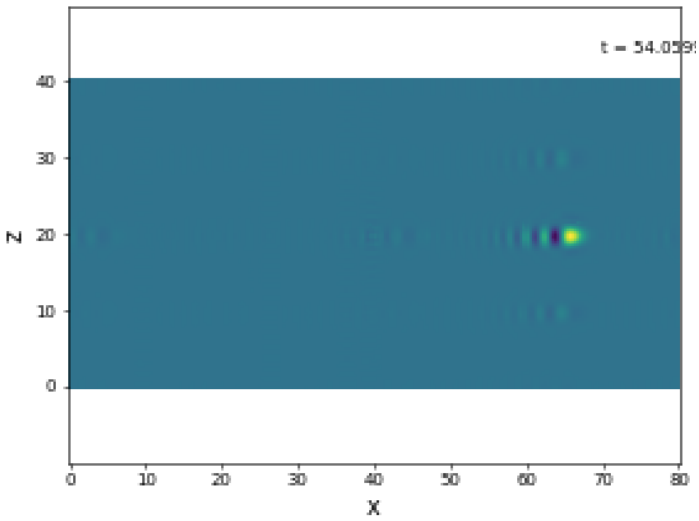
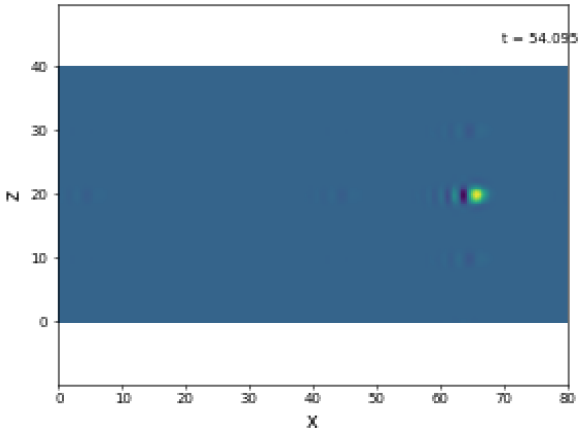


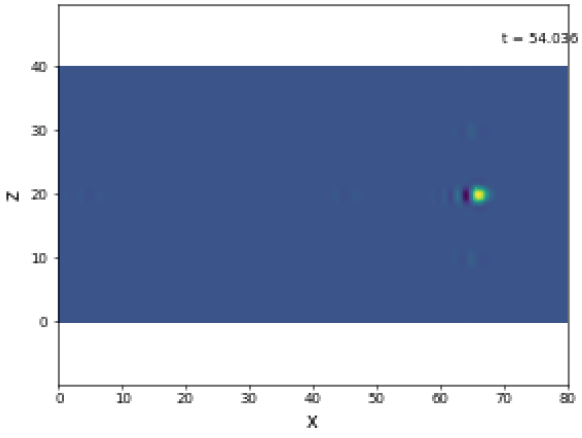
Figure 4.6: Solution of simplified BLEW equations characterised by $m = 3$ at 54 seconds of simulation

The figure 4.6 reports the solution characterised by a *multiplier*, m, of 3 at the physical time of 54 $[s]$. The time employed to realise this simulation is almost 738 $[s]$, three time slower than the previous case, with a relevant reduction in numerical dispersion. However increasing the number of the grid points, as shown in figure 4.7 and 4.8, does not reduce relevantly the numerical dispersion even though the computational time increases respectively to $1451 - 2491$ $[s]$.



0.5

Figure 4.7: Solution of simplified BLEW equations characterised by $m = 4$ at 54 seconds of simulation



0.5

Figure 4.8: Solution of simplified BLEW equations characterised by $m = 5$ at 54 seconds of simulation

4.2.3 Comparison

This sub-section summarises what reported in the previous two sub-sections. The results are reported in table 4.1 and are shown in figure 4.9

m	Explicit Euler t_{comp} $[s]$	Partially Implicit Scheme t_{comp} $[s]$
1	197	88
2	806	258
3	2410	738
4	–	1451
5	–	2491

Table 4.1: computational cost for different grid spacing

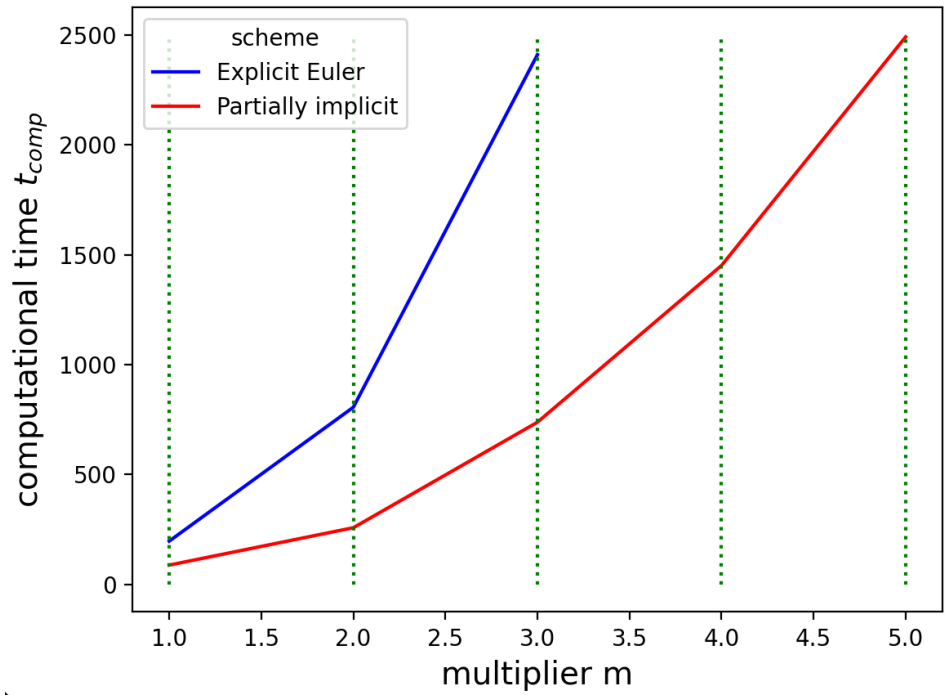


Figure 4.9: Relation between the computational time, seconds, and the grid refinement expressed in number of grid points by the the *multiplier*, m.

Figure 4.9 shows the trend for the computational time depending on the grid refinement expressed by the *multiplier*, m, recall that $dx = L_x/n_x = L_x/(mL_x) = 1/m$.

It is clearly shown that the increase in the number of grid points causes a steep increase in the computational time in both cases however Explicit Euler Scheme has a higher computational cost with respect to the partially implicit scheme. Alternatively with the same computational time considered the partially implicit scheme allows for a better mesh grid refinement than the other scheme.

Lastly considering the figures inherent to the explicit Euler scheme and to the partially implicit scheme it could be said, from a visual interpretation, that although the more refined grid of the explicit Euler is more accurate with respect to the partially implicit scheme. A good trade off between accuracy and computational cost is proposed by the other scheme especially if a finer grid, $m = 4$, is considered.

CHAPTER

5 Conclusions

The work done until now and the one presented in this thesis is just a single step towards the final goal.

This consists in the realisation of a software, backed up by an efficient reinforcement learning algorithm, able to observe the state of the coating and modifying it by varying the velocity output of the gas jet. To do this efficiently the software should be able to process a huge amount of calculations and data while taking the right decisions therefore the goal can be divided into two main objectives:

- Realisation of an efficient solver which is able to speed up the calculations and therefore reduce the computational cost without losing accuracy.
- Realisation of a smart Reinforcement Learning algorithm which is able to accomplish the desired thickness by using the least amount of actuators possible.

Both subjects are being studied and the work is carried forward.

This thesis is the preliminary study of spectral methods and has the goal of presenting to the readers the rough idea of the inner working of these algorithms in order to gain a quick

understanding of the problem and the solution chosen to solve it.

The next step that should be carried out in order to complete the work and create the final product requested by the company are :

- Development of more computationally efficient spectral schemes.
- Introduction of gas perturbation
- Modification of the solver in order to represent the reality of the phenomenon which means that the wave should not re-enter from the input.
- Validation of the efficiency of the scheme chosen regarding the new, more completed, system of equations.

To create an industrially applicable solver the computational cost should be greatly reduced without incurring in losses of stability and accuracy. To reach this goal semi implicit scheme could be a great starting point since it gives a certain degree of versatility in mathematical treatment which could potentially lead to formulate a variety of equations each characterised by different values of stability and accuracy. Thus a good method could be found with a reduced computational cost.

Appendix

This section presents, in the following order, python codes of : 1D advection equation, 1D Burgers equation, 2D advection equation and 2D Burgers equations.

In addition a final code containing three different spectral solver of the simplified BLEW equations is reported in the latest part of the appendix.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jan 30 11:31:32 2023

@author: aleksandr
"""
# This algorithm solves 1D advection equation with BC utilizing upwind scheme w
# and spectral scheme


#%%% LIBRARIES

import numpy as np
import matplotlib.pyplot as plt
import os
import imageio

from scipy import sparse as sp
from scipy.integrate import solve_ivp
#from General_Functions import ANIMATE_D

#%%% NUMERICAL DOMAIN

L     = 3                         # Length of the domain
n_x   = 101                       # Number of grid points
x     = np.linspace(0,L,n_x)      # Domain discretization
dx    = float(round(L/n_x,3))     # Space step
U     = 1                         # Flow velocity
CFL   = 0.8                       # CFL = U * dt / dx condition
dt    = CFL * dx / U              # Time step
t_i   = 0                         # Initial time
t_f   = 4                         # Final time
times = np.arange(t_i,t_f,dt)     # Time discretization

#%%% INITIAL CONDITION

u0 = np.exp(-(x-0.6)**2/0.05)

#%%% UPWIND SCHEME

FD = np.zeros((n_x,n_x))   # Matrix of coefficients

for j in range(n_x):
    FD[j,j] = 1
    try:
        FD[j,j-1] = -1
    except:
        pass

FDs = sp.csr_matrix(FD)    # Sparse matrix

def upwind1D(t,u,FD,dx):
    dudx = FD.dot(u)/dx
    return -dudx
```

```
sol_upwind = solve_ivp(upwind1D,[t_i,t_f],u0,args=(FDs,dx),t_eval=times,dense_o
Sol_upwind = sol_upwind.y

#Y = ANIMATE_D(x,times,sol_upwind,'Upwind1D.gif')

#%%% SPECTRAL SCHEME

k = np.fft.fftfreq(n_x,dx)  # Wavenumber

# dudx = -1j k u_hat

def spectral(t,u,k):
    u_hat = np.fft.fft(u)
    dudx_hat = 1j*k*u_hat
    dudx = np.real(np.fft.ifft(dudx_hat))
    return -dudx

sol_spectral = solve_ivp(spectral,[t_i,t_f],u0,t_eval=times,args=(k,),dense_out
Sol_spectral = sol_spectral.y
time=sol_spectral.t



#%%% PLOTTING

plt.rc('text', usetex=False)
plt.rc('font', family='serif')
plt.rc('xtick',labelsize=16)
plt.rc('ytick',labelsize=16)

def animate(x,t,Sol,GIFNAME):
    Fol_Out='Video_Images_temp'
    n_t=len(t)
    if not os.path.exists(Fol_Out):
        os.mkdir(Fol_Out)

    for k in range(1,n_t):
        fig, ax1 = plt.subplots(figsize=(8, 5)) # This creates the figure
        plt.plot(x,Sol[:,k])
        plt.plot(x,Sol[:,0],'r--')
        ax1.set_xlabel('x',fontsize=14)
        ax1.set_ylabel('u',fontsize=14)
        ax1.set_xlim([0,np.max(x)])
        ax1.set_ylim(-2,2)
        Name=Fol_Out+ os.sep +'Step_'+str(k)+'.png'
        MEX= 'Exporting Im '+ str(k)+' of ' + str(n_t)
        print(MEX)
        plt.savefig(Name, dpi=50)
        plt.close()

    print('Temporary images exported')
    images=[]

    for k in range(1,n_t,1):
```

```
            MEX= 'Mounting Im '+ str(k)+' of ' + str(n_t)
            print(MEX)
            FIG_NAME=Fol_Out+os.sep+'Step_'+str(k)+'.png'
            images.append(imageio.imread(FIG_NAME))

        imageio.mimsave(GIFNAME, images,duration=0.05)

        import shutil
        shutil.rmtree(Fol_Out)

        MEX='Animation'+GIFNAME+' Ready'
        print(MEX)

        return MEX


# Make the video of the solutions
animate(x,time,Sol_spectral,'Test_Spectra.gif')



# Make the video of the solutions
animate(x,time,Sol_upwind,'Test_FD.gif')
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jan 31 13:42:06 2023

@author: aleksandr
"""

# This script has the goal of solving the 1D viscous Burgers equation
    # u_t + u * u_x = nu * u_xx
# Upwind method and spectral method will be used to compute the equation

#%%% LIBRARIES

import numpy as np
import matplotlib.pyplot as plt
import os
import imageio

from scipy import sparse as sp
from scipy.integrate import solve_ivp
from scipy.sparse import csr_matrix

#%%% PLOTTING

plt.rc('text', usetex=False)
plt.rc('font', family='serif')
plt.rc('xtick',labelsize=16)
plt.rc('ytick',labelsize=16)

def animate(x,t,Sol,GIFNAME):
    Fol_Out='Video_Images_temp'
    n_t=len(t)
    if not os.path.exists(Fol_Out):
        os.mkdir(Fol_Out)

    for k in range(1,n_t):
        fig, ax1 = plt.subplots(figsize=(8, 5))
        plt.plot(x,Sol[:,k])
        plt.plot(x,Sol[:,0],'r--')
        ax1.set_xlabel('x',fontsize=14)
        ax1.set_ylabel('u',fontsize=14)
        ax1.set_xlim([0,np.max(x)])
        ax1.set_ylim(-2,2)
        Name=Fol_Out+ os.sep +'Step_'+str(k)+'.png'
        MEX= 'Exporting Im '+ str(k)+' of ' + str(n_t)
        print(MEX)
        plt.savefig(Name, dpi=50)
        plt.close()

    print('Temporary images exported')
    images=[]

    for k in range(1,n_t,1):
        MEX= 'Mounting Im '+ str(k)+' of ' + str(n_t)
```

```
        print(MEX)
        FIG_NAME=Fol_Out+os.sep+'Step_'+str(k)+'.png'
        images.append(imageio.imread(FIG_NAME))

    imageio.mimsave(GIFNAME, images,duration=0.01)

    import shutil
    shutil.rmtree(Fol_Out)

    MEX='Animation'+GIFNAME+' Ready'
    print(MEX)
    return MEX

#%%% DOMAIN

L     = 3                                   # Length of the domain
n_x   = 128                                 # Number of grid points
dx    = round(L/(n_x),3)                    # Space step
x     = np.linspace(0,L,n_x)                # Domain discretization

n_t   = 256                                 # Number of temporal points
t_i   = 0                                   # Initial time
t_f   = 4                                   # Final time
dt    = round((t_f-t_i)/(n_t),3)            # Time step
t     = np.arange(t_i,t_f,dt)               # Time discretization

nu    = 0.1                                 # Dynamic viscosity

#%%% INITIAL CONDITION

u0  = np.exp(-(x-0.6)**2/0.05)              # Initial condition

#%%% UPWIND METHOD

# First derivative (u_i-u_(i-1))/dx
FD = np.diagflat(np.ones((n_x,1)),0) + np.diagflat(-1*np.ones((n_x-1,1)),-1)
FD[0,-1] = -1
FDs = sp.csr_matrix(FD)/dx
#if nu>=.5:
# Second derivative (u_(i+1)-2u_i+u_(i-1))/dx**2
SD = np.diagflat(-2*np.ones((n_x,1)),0) + np.diagflat(np.ones((n_x-1,1)),-1) +
SD[0,-1] = SD[-1,0] = 1
SDs = sp.csr_matrix(SD)/dx**2
# else:
#     SD = np.zeros((n_x,n_x))
#     SDs = sp.csr_matrix(SD)

def Burgers1D_UpWind(t,u,FDs,SDs):
    u_t = - u * (FDs.dot(u)) + nu * (SDs.dot(u))
    return u_t

Sol_UpWind = solve_ivp(Burgers1D_UpWind,[t_i,t_f],u0,method='Radau',t_eval=t,de
u_uw = Sol_UpWind.y
animate(x,t,u_uw,'Burgers1D_Upwind.gif')
#%%% SPECTRAL METHOD
```

```
k = np.fft.fftfreq(n_x, dx)

def Burgers1D_SPECtral(t,u,nu,k):

    u_hat = np.fft.fft(u)
    u_hat_x = 1j * k * u_hat
    u_hat_xx = -k**2 * u_hat

    u_x = np.real(np.fft.ifft(u_hat_x))
    u_xx = np.real(np.fft.ifft(u_hat_xx))

    u_t = - u * u_x + nu * u_xx
    return u_t

Sol_SPECtral = solve_ivp(Burgers1D_SPECtral,[t_i,t_f],u0,t_eval=t,args=(nu,k),d
u_spec = Sol_SPECtral.y
time_spec = Sol_SPECtral.t
animate(x,time_spec,u_spec,'Burgers1D_Spectral.gif')
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 21 14:43:15 2023

@author: aleksandr
"""

# This script ha the goal to solve the 2D Advection equation with spectral meth
# u_t + 3*u_x + 2*u_y = nu + (u_xx + u_yy)

#%%% LIBRARIES

import numpy as np
import matplotlib.pyplot as plt
import os
import imageio

from scipy import sparse as sp
from scipy.integrate import solve_ivp
from scipy.sparse import csr_matrix

#%%% DOMAIN

Lx  = 3
Ly  = 2
ti  = 0
tf  = 5
nx  = 300
ny  = 200
nt  = 600
nu  = 0.1

x0    = 1
y0    = 1
sigma = 0.25

dx  = round(Lx/nx,3)
x   = np.linspace(0,Lx,nx)
dy  = round(Ly/ny,3)
y   = np.linspace(0,Ly,ny)
dt  = round((tf-ti)/nt,3)
t   = np.linspace(ti,tf,nt)

#%%% INITIAL CONDITION

(xx,yy) = np.meshgrid(x,y)
u0 = np.exp(-((xx-x0)**2 + (yy-y0)**2)/2/sigma**2)

plt.ioff()
fig = plt.figure(num=1)
ax = plt.axes(projection='3d')
ax.contour3D(xx, yy, u0, 50, cmap='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
```

```
ax.set_zlabel('u')

#%%% SPECTRAL METHOD

kx_vect = 2*np.pi*np.fft.fftfreq(nx,dx)
ky_vect = 2*np.pi*np.fft.fftfreq(ny,dy)

kx = np.zeros((ny,nx))
ky = np.zeros((ny,nx))

for j in range(nx):
    for i in range(ny):

        kx[i,j] = kx_vect[j]
        ky[i,j] = ky_vect[i]


u_hat = np.fft.fft2(u0)

Fol_Out='Video_Images_temp'
n_t=len(t)
if not os.path.exists(Fol_Out):
   os.mkdir(Fol_Out)

for n in range(1,nt):

    u_hat = u_hat * (1/dt -3j*kx -2j*ky)
    u_hat = u_hat / (1/dt + nu * (kx**2 + ky**2))
    u = np.real(np.fft.ifft2(u_hat))
    u_hat = np.fft.fft2(u)

    fig = plt.figure(num=n)
    ax = plt.axes(projection='3d')
    ax.contour3D(xx, yy, u, 50, cmap='binary')
    ax.set_xlabel('x',fontsize=14)
    ax.set_ylabel('y',fontsize=14)
    ax.set_zlabel('u',fontsize=14)
    ax.set_xlim([0,Lx])
    ax.set_ylim([0,Ly])
    Name=Fol_Out+ os.sep +'Step_'+str(n)+'.png'
    MEX= 'Exporting Im '+ str(n)+' of ' + str(nt)
    print(MEX)
    plt.savefig(Name, dpi=50)
    plt.close()

print('Temporary images exported')
images=[]

GIFNAME = '2D_AdvEq.gif'

for k in range(1,nt,1):
    MEX= 'Mounting Im '+ str(k)+' of ' + str(nt)
    print(MEX)
    FIG_NAME=Fol_Out+os.sep+'Step_'+str(k)+'.png'
    images.append(imageio.imread(FIG_NAME))
```

```
 # Now we can assembly the video
imageio.mimsave(GIFNAME, images,duration=0.05)

import shutil  # nice and powerfull tool to delete a folder and its content
shutil.rmtree(Fol_Out)

MEX='Animation'+GIFNAME+' Ready'
print(MEX)
```

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Feb 23 13:18:24 2023

@author: aleksandr
"""

# This script solve 2D burgers equation with Spectral Methods
    # u_t + uu_x + vu_y = nu * (u_xx + u_yy)
    # v_t + uv_x + vv_y = nu * (v_xx + v_yy)
# The non linear term will be calculated implicitly

#%%% LIBRARIES

import numpy as np
import matplotlib.pyplot as plt
import os
import imageio.v2 as imageio
import time

#%%% DOMAIN

Lx  = 3
Ly  = 2
tf  = 5
nx  = 60
ny  = 40
nt  = 5000
nu  = 0.01

x0    = 1
y0    = 1
sigma = 0.20

dx  = Lx/nx
dy  = Ly/ny
dt  = 5e-3

x   = np.linspace(0,Lx,nx)
y   = np.linspace(0,Ly,ny)

#%%% INITIAL CONDITION

(xx,yy) = np.meshgrid(x,y)
u = np.exp(-((xx-x0)**2 + (yy-y0)**2)/2/sigma**2)
v = np.exp(-((xx-x0)**2 + (yy-y0)**2)/2/sigma**2)

# plt.ioff()
# fig = plt.figure(num=1)
# ax = plt.axes(projection='3d')
# ax.contour3D(xx,yy,u,50,cmap='binary')
# #ax.contour3D(xx,yy,v,50,cmap='binary')
# ax.set_xlabel('x')
# ax.set_ylabel('y')
```

```
#%%% SPECTRAL METHOD

start = time.perf_counter()

# If it is needed add to the function input the filter which then would be used
# ux_hat = 1j *(kx *filtr_x) *u_hat

def Burgers2D(dt,u,v,kx,ky):

    u_hat = np.fft.fft2(u)
    v_hat = np.fft.fft2(v)

    ux  = np.real(np.fft.ifft2(1j *kx *u_hat))
    uy  = np.real(np.fft.ifft2(1j *ky *u_hat))
    uxx = np.real(np.fft.ifft2(-kx**2 *u_hat))
    uyy = np.real(np.fft.ifft2(-ky**2 *u_hat))

    vx  = np.real(np.fft.ifft2(1j *kx *v_hat))
    vy  = np.real(np.fft.ifft2(1j *ky *v_hat))
    vxx = np.real(np.fft.ifft2(-kx**2 *v_hat))
    vyy = np.real(np.fft.ifft2(-ky**2 *v_hat))

    # Fully explicit scheme

    RHS_u = -u * ux - v *uy +nu *(uxx +uyy)
    RHS_v = -u * vx - v *vy +nu *(vxx +vyy)

    # Partially explicit-implicit scheme

    # RHS_u = (u/dt +nu *(uxx + uyy)) /(1/dt +ux +vy)
    # RHS_v = (v/dt +nu *(vxx + vyy)) /(1/dt +ux +vy)

    return RHS_u, RHS_v

def forward_euler(dt,u,v,RHS_u,RHS_v):

    u = u +dt *RHS_u
    v = v +dt *RHS_v

    return u,v

kx_vect = 2*np.pi*np.fft.fftfreq(nx,dx)
ky_vect = 2*np.pi*np.fft.fftfreq(ny,dy)

kx = np.zeros((ny,nx))
ky = np.zeros((ny,nx))

for j in range(nx):
    for i in range(ny):

        kx[i,j] = kx_vect[j]
        ky[i,j] = ky_vect[i]
```

```
# FILTERING PROCESS----------------------------------------------------###

# filtr_x = np.ones_like(kx)
# filtr_y = np.ones_like(ky)
# max_kx  = np.max(np.abs(kx[0,:]))
# max_ky  = np.max(np.abs(ky[:,0]))
# filtr_x[np.where(np.abs(kx)>max_kx*2/3)] = 0
# filtr_y[np.where(np.abs(ky)>max_ky*2/3)] = 0

# ---------------------------------------------------------------------###

plt.ioff()
Fol_Out='Video_Images_temp'

if not os.path.exists(Fol_Out):
   os.mkdir(Fol_Out)

for n in range(1,nt+1):

    (RHS_u,RHS_v) = Burgers2D(dt, u, v, kx, ky)
    (u,v) = forward_euler(dt, u, v, RHS_u, RHS_v)

    # (u,v) = Burgers2D(dt, u, v, kx, ky)

    fig = plt.figure()
    ax  = plt.axes(projection='3d')
    ax.contour3D(xx, yy, u, 50, cmap='binary')
    ax.set_xlabel('x',fontsize=14)
    ax.set_ylabel('y',fontsize=14)
    ax.set_zlabel('u',fontsize=14)
    ax.set_xlim([0,Lx])
    ax.set_ylim([0,Ly])
    Name=Fol_Out+ os.sep +'Step_'+str(n)+'.png'
    MEX= 'Exporting Im '+ str(n)
    print(MEX)
    plt.savefig(Name, dpi=50)
    plt.close()

print('Temporary images exported')
images=[]

GIFNAME = '2D_BurgersEq.gif'

for k in range(1,nt,1):
    MEX= 'Mounting Im '+ str(k)
    print(MEX)
    FIG_NAME=Fol_Out+os.sep+'Step_'+str(k)+'.png'
    images.append(imageio.imread(FIG_NAME))

imageio.mimsave(GIFNAME, images,duration=0.001)

import shutil
shutil.rmtree(Fol_Out)

MEX='Animation'+GIFNAME+' Ready'
```

```
print(MEX)

finish = time.perf_counter()

print(f'Time FINISH : {finish-start}')
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May  3 09:59:17 2023

This code solve spectrally the following equations

    \partial_t h  + \partial_x F11 + \partial_z F21 = S1
    \partial_t qx + \partial_x F12 + \partial_z F22 = S2
    \partial_t qz + \partial_x F13 + \partial_z F23 = S3

where h is the coating thickness while qx and qz are flow rates.
Fij are flux terms while Sij are the source terms.

Hp : dpX,dpZ = 0 & dtX,dtZ = 0
    pressure gradient & gas shear stress are null therefore

@author: aleksandr
"""

#%% LABRIERIES

import numpy as np
import matplotlib.pyplot as plt
import os
import imageio.v2 as imageio
import time

#%% DOMAIN

Lx  = 80                    # Streamwise length
Lz  = 40                    # Spanwise length
tf  = 4                     # Time Finish phenomenon
nx  = 80                    # Number of streamwise grid points
nz  = 40                    # Number of spanwise grid points
nt  = 100000                # Number of grid points for time discretisation

dx  = Lx/nx                 # Grid Spacing streawise
dz  = Lz/nz                 # Grid spacing spanwise
dt  = dx/50                 # Grid spacing for temporal discretisation

x   = np.linspace(0,Lx,nx)  # Streawise spatial discretisation
z   = np.linspace(0,Lz,nz)  # Spanwise spatial discretisation

x0  = 40                    # Origin of the Gaussian
z0  = 20                    # Origin of the Gaussian

(zz,xx) = np.meshgrid(z,x)  # Matrices of coordinate for Spatial Discretisatior

#%% PARAMETERS (Zinc properties)

U_p   = 1                   # m/s^2 Metal Strip Velocity
rho_l = 6500                # kg/m^3 Density
mu_l  = 0.003               # kg/m/s Dynamic viscosity
nu_l  = mu_l/rho_l          # m^2/s Kinematic viscosity
```

```
g     = 9.8                # m/s^2 Gravitational accelaration
sigma = 0.78               # N/m

#%% SCALING LAWS

u      = 2*U_p             # Wave travel speed (conservative estimate for CFL)
Ca     = (mu_l*U_p/sigma)          # Capillary number
Re     = np.sqrt(U_p**3/g/nu_l)    # Reynolds number
epsilon = Ca**(1/3)                # Parameter given by Shkadov's scaling

#%% IC

sig = 0.5                                      # variance
h0=0.1
h = 0.05*np.exp(-((xx-x0)**2 + (zz-z0)**2)/2/sig**2)+h0  # initial film hight
qx = (1/3)*h**3-h                              # initial streamwise flow rate
qz = np.zeros((nx,nz))                         # initial spanwise flow rate

fig = plt.figure(num=1)
plt.pcolor(xx,zz,h)
plt.axis('equal')
plt.xlabel("x")
plt.ylabel("z")
plt.colorbar()


#%% EXPLICIT EULER Spectral Scheme

from scipy import signal

# Approach using FIR filter theory
def low_PASS_H(kx_vect,k_hat_lim=0.8):
    '''create transfer function for low pass filtering'''
    # Create the transfer function
    kernel = signal.firwin(len(kx_vect)//2, k_hat_lim, window = 'hamming')
    # You could the transfer function like this if you want:
    w, H_T  =  signal.freqz(kernel,worN=len(kx_vect),whole=True)
    H_z_p=np.conj(H_T)*H_T
    return H_z_p

kx_vect = 2*np.pi*np.fft.fftfreq(nx,dx)
kz_vect = 2*np.pi*np.fft.fftfreq(nz,dz)
# Grid of wave numbers
kz,kx=np.meshgrid(kz_vect,kx_vect)

# Create the filters along x and z
H_x=np.real(low_PASS_H(kx_vect,k_hat_lim=0.5)).reshape((len(kx_vect),1))
H_z=np.real(low_PASS_H(kz_vect,k_hat_lim=0.5)).reshape((len(kz_vect),1))
# Create the full transfer function
H_m=H_x.dot(H_z.T) # build the 2D filter from 1D filter

start = time.perf_counter()                    # Start timing

# Definition of a function for the spectral scheme applied to BLEW
# Spatial approximation
```

```
def sources(h,qx,qz):

    h_hat = np.fft.fft2(h)*H_m    # (filtered) film height in frequency domain

    tau_w_x = -3*qx/h**2 -3/h     # Shear stress streamwise at the wall
    tau_w_z = -3*qz/h**2          # Shear stress spanwise at the wall


    # h_ijk are capillary pressure gradients

    h_xxx = np.real(np.fft.ifft2((1j*kx)**3*h_hat))
    h_xzz = np.real(np.fft.ifft2((1j*kx)*(1j*kz)*h_hat))

    S2 = 1/epsilon/Re*(h*(h_xxx+h_xzz+1)+tau_w_x)

    h_zzz =  np.real(np.fft.ifft2((1j*kz)**3*h_hat))
    h_xxz =  np.real(np.fft.ifft2((1j*kx)**2*(1j*kz)*h_hat))

    S3 = 1/epsilon/Re*(h*(h_zzz+h_xxz)+tau_w_z)

    return S2,S3

def fluxes(h,qx,qz):

    h_hat = np.fft.fft2(h)*H_m  # Filtered h in freq domain
    qx_hat = np.fft.fft2(qx)*H_m # Filtered qx in freq domain
    qz_hat = np.fft.fft2(qz)*H_m  # Filtered q_Z in freq domain

    # F11 = qx
    # F21 = qz

    F11_x = np.real(np.fft.ifft2(1j*kx*qx_hat))
    F21_z = np.real(np.fft.ifft2(1j*kz*qz_hat))

    # F12 = (144*qx**2+48*h*qx+24*h**2)/120/h
    # F22 = (144*qx*qz+24*h*qz)/120/h

    # Non Linear terms
    a = qx**2/h
    b = qx*qz/h

    a_hat = np.fft.fft2(a)*H_m
    b_hat = np.fft.fft2(b)*H_m

    a_x = np.real(np.fft.ifft2(1j*kx*a_hat))
    b_z = np.real(np.fft.ifft2(1j*kz*b_hat))
    qx_x = np.real(np.fft.ifft2(1j*kx*qx_hat))
    h_x = np.real(np.fft.ifft2(1j*kx*h_hat))
    qz_z = np.real(np.fft.ifft2(1j*kz*qz_hat))

    F12_x = 1/5*(6*a_x+2*qx_x+h_x)
    F22_z = 1/5*(6*b_z+qz_z)

    # F13 = F22
```

```
    # F23 = 144*qz**2/120/h

    c = qz**2/h

    c_hat = np.fft.fft2(c)*H_m

    b_x = np.real(np.fft.ifft2(1j*kx*b_hat))
    c_z = np.real(np.fft.ifft2(1j*kz*c_hat))
    qz_x = np.real(np.fft.ifft2(1j*kx*qz_hat))

    F13_x = 1/5*(6*b_x+qz_x)
    F23_z = 6/5*c_z

    return F11_x,F21_z,F12_x,F22_z,F13_x,F23_z

def Explicit_Euler(dt,h,qx,qz,S2,S3,F11_x,F21_z,F12_x,F22_z,F13_x,F23_z):

    h = h +dt *(-F11_x-F21_z)
    qx = qx +dt *(S2-F12_x-F22_z)
    qz = qz +dt *(S3-F13_x-F23_z)

    return h,qx,qz

#%% IMPLICIT - EXPLICIT Spectral Scheme :
    # (qx*qz/h) is the only implicit term

def Imp_Exp_SS(dt,h,qx,qz,S2,S3):

    qx_hat = np.fft.fft2(qx)*H_m  # Filtered qx in freq domain
    qz_hat = np.fft.fft2(qz)*H_m  # Filtered q_Z in freq domain

    # First equation : \partial_t h  + \partial_x F11 + \partial_z F21 = S1

    qx_x = np.real(np.fft.ifft2(1j*kx*qx_hat))
    qz_z = np.real(np.fft.ifft2(1j*kz*qz_hat))

    h = h -dt *(qx_x +qz_z)

    # Second equation : \partial_t qx  + \partial_x F12 + \partial_z F22 = S2

    a = qx**2/h
    a_hat = np.fft.fft2(a)*H_m
    a_x = np.real(np.fft.ifft2(1j*kx*a_hat))

    qx = (S2+qx/dt-24/120*(6*a_x+2*qx_x+qz_z))/(1/dt+144/120*qz_z/h)

    # Third equation : \partial_t qz  + \partial_x F13 + \partial_z F23 = S3

    qz_x = np.real(np.fft.ifft2(1j*kx*qz_hat))

    b = qz**2/h
    b_hat = np.fft.fft2(b)*H_m
    b_z = np.real(np.fft.ifft2(1j*kz*b_hat))

    qz = (S3+qz/dt-24/120*(qz_x+6*b_z))/(1/dt+144/120*qx_x/h)
```

```
    return h,qx,qz

#%%% IMPLICIT - EXPLICIT Spectral Scheme 2 : in addition to the (qx*qz/h) the
    # following term is treated implicitly 2*h*qx*qx_x/h**2
    # obtained from qx**2/h = (2*h*qx*qx_x+qx**2h_x)/(h**2)

def Imp_Exp_SS2(dt,h,qx,qz,S2,S3):

    h_hat  = np.fft.fft2(h)*H_m  # Filtered h in freq domain
    qx_hat = np.fft.fft2(qx)*H_m # Filtered qx in freq domain
    qz_hat = np.fft.fft2(qz)*H_m # Filtered qz in freq domain

    # First equation

    qx_x = np.real(np.fft.ifft2(1j*kx*qx_hat))
    qz_z = np.real(np.fft.ifft2(1j*kz*qz_hat))

    h = h -dt *(qx_x +qz_z)                                # film height equation

    # Second equation

    a     = qz/h
    a_hat = np.fft.fft2(a)*H_m
    a_z   = np.real(np.fft.ifft2(1j*kz*a_hat))

    b     = qx/h
    b_hat = np.fft.fft2(b)*H_m
    b_x   = np.real(np.fft.ifft2(1j*kx*b_hat))

    it = 1/dt+12/5*(b_x+a_z)                               # Implicit term

    h_x  = np.real(np.fft.ifft2(1j*kx*h_hat))
    qx_x = np.real(np.fft.ifft2(1j*kx*qx_hat))
    qz_z = np.real(np.fft.ifft2(1j*kz*qz_hat))

    qx = (S2+qx/dt-1/5*(h_x*(6*b**2+1)+2*qx_x+qz_z))/it # x-Flow rate equation

    # Third equation

    h_z = np.real(np.fft.ifft2(1j*kz*h_hat))
    qz_x = np.real(np.fft.ifft2(1j*kx*qz_hat))

    qz = (S3+qz/dt+1/5*(a**2*h_z-qz_x))/it             # z-Flow rate equation

    return h,qx,qz

#%% COMPUTATION

plt.ioff()
Fol_Out='Video_Images_temp'

if not os.path.exists(Fol_Out):
   os.mkdir(Fol_Out)
```

```
X=100


for n in range(1,nt+1):

    (S2,S3) = sources( h, qx, qz)
    #(F11_x,F21_z,F12_x,F22_z,F13_x,F23_z) = fluxes( h, qx, qz)
    #(h,qx,qz) = Explicit_Euler(dt, h, qx, qz, S2, S3, F11_x, F21_z, F12_x,
    #                           F22_z, F13_x, F23_z)
    #(h,qx,qz) = Imp_Exp_SS(dt, h, qx, qz, S2, S3)
    (h,qx,qz) = Imp_Exp_SS2(dt, h, qx, qz, S2, S3)

    if n%X == 0: # To speed up, save every X steps
     fig = plt.figure()
     plt.pcolor(xx, zz, h)
     plt.xlabel('x',fontsize=14)
     plt.ylabel('z',fontsize=14)
     plt.axis('equal')
     Name=Fol_Out+ os.sep +'Step_'+str(n)+'.png'
     MEX= 'Exporting Im '+ str(n)
     print(MEX)
     plt.savefig(Name, dpi=50)
     plt.close()

print('Temporary images exported')
images=[]

GIFNAME = 'BLEW_Imp_Exp_SS2_dx_SU_50.gif'

for k in range(1,nt,1):
    if k%X == 0:
     MEX= 'Mounting Im '+ str(k)
     print(MEX)
     FIG_NAME=Fol_Out+os.sep+'Step_'+str(k)+'.png'
     images.append(imageio.imread(FIG_NAME))

imageio.mimsave(GIFNAME, images,duration=0.001)

import shutil
shutil.rmtree(Fol_Out)

MEX='Animation'+GIFNAME+' Ready'
print(MEX)

finish = time.perf_counter()

print(f'Time FINISH : {finish-start}')
```

Bibliography

[1] M.A Mendez A. Gosset B. Scheid M. Balabane J.-M. Buchlin. “Dynamics of the Jet Wiping Process via Integral models”. In: *Journal of fluid mechanics* (28/04/2020). DOI: 2004.13400.

[2] Gabriele Gamba. “Development of a fine volume code for simulating liquid films instabilities in hot dip galvanisation”. MA thesis. Politecnico di Torino, 2022.

[3] A. Gosset and J. M. Buchlin. “Jet wiping in hot-dip galvanization”. In: *journal of Fluids Engineering* 129(4).466 (2007). DOI: 10.1115/1.2436585.

[4] David A. Kopriva. *Implementing Spectral Methods for Partial Differential Equations Algorithms for Scientists and Engineers*. Springer Dordrecht, 2009.

[5] Francesco Mancini. “Machine Learning Control of 3D liquid film”. MA thesis. Politecnico di Torino, 2023.

[6] Lloyd N. Trefethen. *Spectral Methods in Matlab*. SIAM, 2020. ISBN: 978-0-89871-465-4.

[7] Benoit Scheid Tsvetelina Ivanova Fabio Pino and Miguel A. Mendez. “Evolution of waves in liquid films on moving substrates”. In: (19/01/2023). DOI: 2203.08201.