Computational Thermo-Fluid Dynamics
Summer Semester 2023-2024

Thermo-Fluid analysis and simulation of Ice-Cream Machine

Niki Balestrieri[1], Bojun Zhao[2]

Abstract

In this discussion we are investigating the 2-Dimensional Thermo-Fluid Dynamics of ice-cream machines.

Our work is aimed to show the complexity of such a commonly known process, studying the optimal parameters, conditions and path for obtaining one of the most popular food industries' product.

For better clarity and understanding, the analysis will be divided in two major parts: a rheological discussion and a thermal one, showing how the two influence one another. Dealing with a non-newtonian fluid, in particular a shear-thinning one, adds complexity to the model and allows us to find a correlation between viscosity and temperature, crucial for the computations. The geometry used recalls a Taylor-Couette reactor in which is inserted a measurement device aimed to break symmetry. The ice-cream optimal temperature is reached through a cooling external cylinder, kept at constant temperature and in direct contact with the fluid. For the computation a finite difference method is used and RK4 time steppig scheme.

Keywords

Heat conduction and convection — Non-Newtonian fluid — Finite difference

[1] *Master Student, TUM, Munich, Germany*

[2] *Master Student, TUM, Munich, Germany*

*__e-mails__: niki.balestrieri@tum.de, bojun.zhao@tum.de

Contents

1. Introduction

Before starting a technical analysis of the process, is useful to give an intuitive description of the phenomenon on a macroscopic and visual point stand.
In the artisanal and industrial production of ice-cream, the same steps are involved: gathering the semi-liquid ingredients at a starting temperature of $T = 20 - 25°C$, inserting them in the machine that would **mix** them and eventually **pasteurizing** them at a temperature of $T = 85°C$, in order to kill any harmful bacteria, and finally **cool down** the mixture at a final temperature of $T = -20°C$. At the end, the final costumer is able to enjoy a viscous and cold product that we all know as ice-cream.
We can observe how this phenomenon concerns two major variations over time: a **viscosity** increasing and a **temperature** decreasing. In particular, we are dealing with a **Non-Newtonian fluid**, meaning that viscosity and shear stress are not linearly dependent. In fact being a **pseudoplastic**, or shear-thinning, fluid translates in the fact that viscosity decreases when shear stress is applied. We can observe this when scooping the ice-cream: it becomes less thick until is set again in position on the cone. The viscosity is also related to the change of temperature, in fact it increases with the cooling process.
We are dealing both with **thermal conduction and convection**, having the outer cylinder setting the operational temperature and the inner one, which approximates the paddles, mixing the fluid for better mechanical agitation and enhancing the heat transfer from the outer wall.
The flow is studied through the **Navier-Stokes equations**, formulation derived through the dependence from the radial and angular dimensions, all in cylindrical coordinates for compatibility with the geometry.
With all the information gathered from the mathematical derivations, we will then set up the computation. First imposing a circular grid with an obstacle in the middle, representing the thermometer inside the cavity, then setting up our solving method for the PDEs and time discretization: chosing an implicit time discretization method, that converges better respect an explicit one in our case, and finite difference method as solving approach.

The discussion will be shaped firstly in a qualitative way in **Chapter 2 - "Physical Analysis"**, where the fundamentals will be explained putting the basis for the mathematical studies in **Chapter 3 - "Numerical methods"**. We will be then discussing the results in **Chapter 4 - "Results and Discussion"** and for better understanding, we will show parts of our code in **Chapter 6 - "Code Appendix"**.

2. Physical Analysis

For starting, is of fundamental importance defining the geometry we are dealing with for then associating the viscosity and temperature changes on that spacial domain.

2.1 Geometry of the machine

The machine will be approximated as a Taylor-Couette reactor, having a cylindrical rotor as paddles and a fixed external cylinder. Between the two cylinders there is an annular cavity in within the ice-cream is mixed and treated. The height of the cylinder is considered unitary since is not taken in account for our computation, in fact the section analyzed is on the horizontal plane.

As mentioned, in order to break the symmetry in the geometry, a **thermometer** is inserted (Figure 2).

For simplicity, the shape and dimension of this device corresponds to a "slice" of the grid built for the spacial discretization (Figure 3).

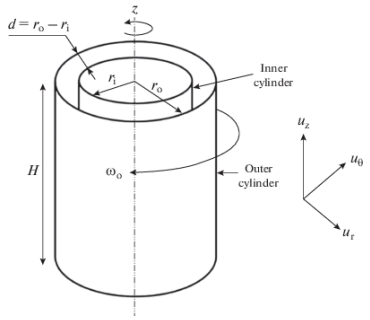

Figure 1. Taylor Couette reactor with an inner rotating cylinder

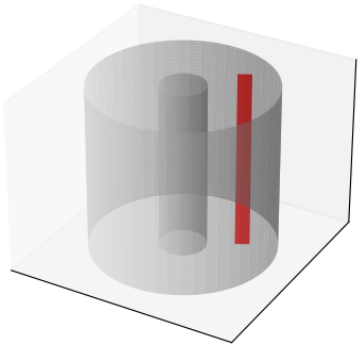

Figure 2. Geometry highlighting the thermometer in red

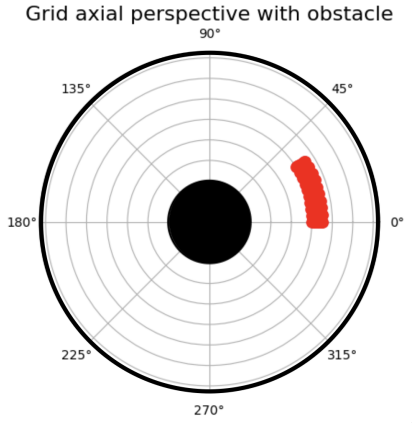

Figure 3. Grid representation

2.2 Rheology of Ice-Cream

The complexity of this model, as previously mentioned, is majorly related to the nature of this fluid: a **Non-Newtonian pseudoplastic** (or shear-thinning) **mixture**. The evaluation of the viscosity is done through the **Power Law Model** as following:

$$\eta = K \cdot \gamma^{n-1} \qquad \text{for } n < 1 \tag{1}$$

where η is the apparent viscosity, K consistency index, γ shear rate and n flow index of a shear thinning fluid. In order to find a correlation between this formulation and the

variation of temperature, we implement the Arrhenius formulation for $K(T)$ and $n(T)$ [1]:

$$K(T) = A_K \cdot exp\left(\frac{E_K}{T(K)}\right) \qquad \text{for } A_K = 3.5754 \text{ and } E_K = -160.4 \tag{2}$$

$$n(T) = A_n \cdot exp\left(\frac{E_n}{T(K)}\right) \qquad \text{for } A_n = 4.4965 \text{ and } E_K = -602.16 \tag{3}$$

We can then visualize the relation between viscosity and shear rate accounting for different temperatures:

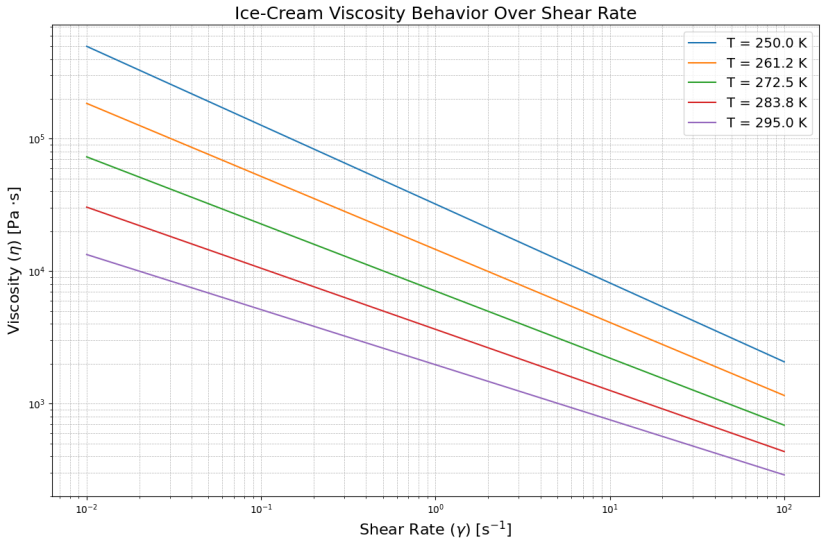

Figure 4. Viscosity behavior with Shear rate in logarithmic form

Other parameters to take in account in this discussion are the **storage modulus** G', representing the storage of energy, and the viscous contribution G'', **loss modulus**. We evaluate this parameters in function of G^*, the total material stiffness, and the phase angle δ, which in our case $0° \leq \delta \leq 90°$ since we are dealing with a viscoelastic fluid.
For a $\delta = 70°$ and G^*, G', G'' defined as:

$$G^* = \frac{\sigma_{MAX}}{\gamma_{MAX}}; \qquad G' = G^* cos\delta; \qquad G'' = G^* sin\delta; \tag{4}$$

Where σ_{MAX} and γ_{MAX} are obtained through mathematical derivation using the Generalized Reynolds number [2]. We obtain a function of the temperature that will be divided in three major zones, for better understanding and analysis. As we can observe in **Figure (5)**, we have two stable zones (Zone 1 and Zone 3), where the modules, minus minor oscillations, are stable. We notice a different behavior in Zone 2, where we can identify the **critical temperature** as $T_C = -2.8°C$, where we have a steep peak. From that point on the viscosity

[1] The values of A_K, A_n, E_K, E_n are found in literature and are result of experimental tests.
[2] This would be explained in detail later on.

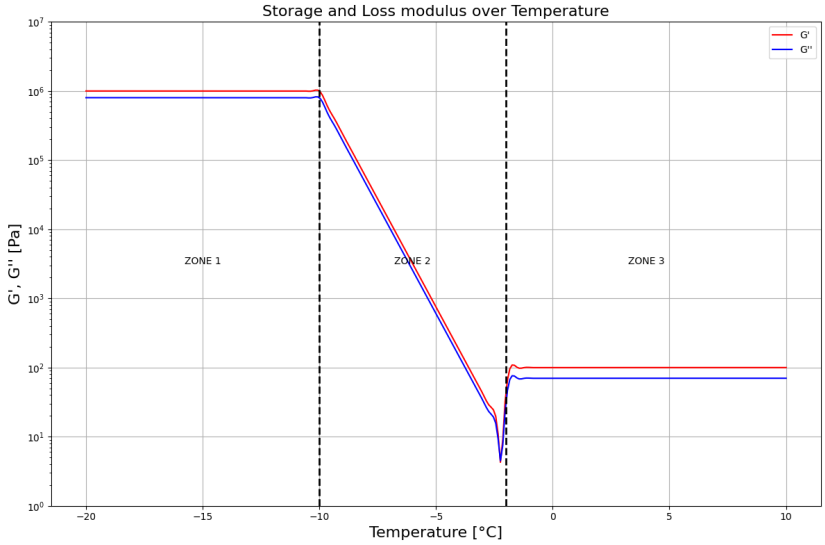

Figure 5. Storage and loss modulus dependency over temperature

stabilizes, making our analysis simpler. We can then determine that the temperatures of interest, where major changes occur, vary from a range of $-10°C \leq T \leq -2.8°C$. This information allow us to visualize the dependency of the viscosity from the temperature as **Figure (6)**, in this case keeping a constant shear rate $\gamma = 65 s^{-1}$.

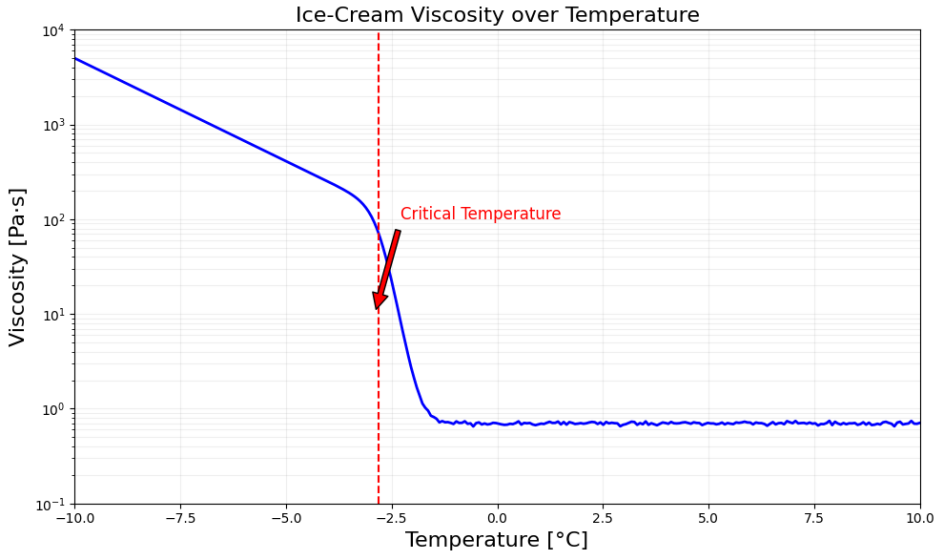

Figure 6. Viscosity dependency over temperature

2.3 Flow stability

We proceed the discussion analyzing the flow and its equations. First, we define the **Generalized Reynolds number**, only suitable formulation for our pseudoplastic fluid:

$$Re = \frac{d_e^n \cdot v^{(2-n)} \cdot \rho}{K \cdot 8^{(n-1)} \left(\frac{(3n+1)}{4n}\right)^n} \qquad \text{with} \quad d_e = d_o - d_i \tag{5}$$

Where d_e is the equivalent diameter [3], v is the angular velocity of the rotor and ρ is the density. Ice-cream is a dispersion and the density will vary based on the dimension of the ice particles in the mixture: for simplicity, we will consider a constant density of $\rho = 1160 kg/m^3$. We have to ensure that the Generalized Reynolds number falls in a range $Re < 2100$ in order to have laminar flow and keep stability. Through this calculation, we can derive the optimal parameters for the computation. Keeping constant the inner and outer radius and the angular velocity, we obtain the following:

r_i [m]	r_o [m]	n	v [rad/s]	T[°C]	K	Re
2.00	0.50	0.42	1.00	-20.00	22432.13	657.91
2.00	0.50	0.47	1.00	-2.80	7213.62	676.44
2.00	0.50	0.49	1.00	0.00	6078.25	679.44
2.00	0.50	0.50	1.00	5.00	4515.02	684.99

Table 1. Reynolds number values for different temperatures

First, we notice how the flow index (n) respects the condition for **Equation (1)** for the range of examined temperatures. We also notice how the Reynolds number always falls in the wanted range for laminar flow, being majorly influenced by the angular velocity and slightly by the temperature. The borderline condition for stability would be:

r_i [m]	r_o [m]	n	v [rad/s]	T[°C]	K	Re
2.00	0.50	0.51	2.10	5.00	4515.02	2048.59

Table 2. Highest values for flow stability at T_{max}

2.4 Thermal analysis

From the previous flow and parameters analysis, we have established the thermal range of interest, such as $-20°C < T < 5°C$. For the pasteurization process, the mixture will be heated for 30 seconds at a temperature of $T = 85°C$, phenomenon that will increase the Reynolds number to $Re = 1839.02$, keeping a constant velocity of 1 rad/s.

[3] For a Taylor-Couette reactor, the considered diameter in the Reynolds number calculation is the one of the internal cavity.

For this specific process, we have to analyze **conduction** and **convection**[4], which both concur to a well distributed heat transfer mechanism.
We are dealing with **thermal conduction** because the outer cylinder transfers heat to the mixture through contact and the heat flows naturally along the temperature gradient until thermal equilibrium is reached. Therefore, we also have the movement of the inner cylinder, which generates **heat convection** due to the motion. With these two mechanisms we are avoiding the accumulation of heat on a specific surface, which allows the ice-cream to have an **homogeneous temperature** all around.

3. Numerical methods

Finally, we get to the heart of the discussion, looking at the dominant equations and integrating the previous knowledge into mathematical form.

3.1 Velocity field equations

For the velocity field we are dealing with **Navier-Stokes equations** in cylindrical coordinates. Since the symmetry is broken by the presence of the obstacle, we cannot reduce our model in a 1D study. Instead, the analysis translates in a **2D model** with the dependency from the radial direction (r) and angular one (θ). We analized the **continuity equation**:

$$\frac{1}{r}\frac{\partial(r\,u_r)}{\partial r}+\frac{1}{r}\frac{u_\theta}{\partial\theta}=0 \tag{6}$$

And the **momentum** for both r-axis and θ-axis:

$$\rho\left(u_r\frac{\partial u_r}{\partial r}+\frac{u_\theta}{r}\frac{\partial u_r}{\partial\theta}-\frac{u_\theta^2}{r}\right)=-\frac{\partial P}{\partial r}+\mu\left[\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u_r}{\partial r}\right)-\frac{u_r}{r^2}+\frac{1}{r^2}\frac{\partial^2 u_r}{\partial\theta^2}-\frac{2}{r^2}\frac{\partial u_\theta}{\partial\theta}\right] \tag{7}$$

$$\rho\left(u_r\frac{\partial u_\theta}{\partial r}+\frac{u_\theta}{r}\frac{\partial u_\theta}{\partial\theta}+\frac{u_r u_\theta}{r}\right)=-\frac{1}{r}\frac{\partial P}{\partial\theta}+\mu\left[\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u_\theta}{\partial r}\right)-\frac{u_\theta}{r^2}+\frac{1}{r^2}\frac{\partial^2 u_\theta}{\partial\theta^2}+\frac{2}{r^2}\frac{\partial u_r}{\partial\theta}\right] \tag{8}$$

Since we are dealing with **pressure** as well, it is required to couple it with the velocity component. For accomplishing this we are using the **SIMPLE algorithm**, where we decompose the matrix into diagonal and off-diagonal part:

$$\mathscr{M}\mathbf{U}=\mathscr{A}\mathbf{U}-\mathscr{H} \tag{9}$$

and implement the following:

1. Solve the pressure gradient coupling function: $\mathscr{M}\mathbf{U}=-\nabla p$
2. Calculate: $\mathscr{H}=\mathscr{A}\mathbf{U}-\mathscr{M}\mathbf{U}$

[4]The mathematical expressions will be deepened in **Chapter 3.2 - "Heat equation"**.

3. Update the pressure: $\nabla \cdot (A^{-1}\nabla p) = \nabla \cdot (A^{-1}\mathscr{H})$
4. Update the velocity field: $\mathbf{U} = A^{-1}\mathscr{H} - A^{-1}\nabla p$
5. Repeat from step 1 until converge

3.2 Heat Transfer equations

In this matter, we are dealing with the **convective-diffusive equation**, or transport equation, which states:

$$\Big(\rho c_p\Big)\frac{\partial T}{\partial t} = \Big(\rho c_p\Big)\nabla \cdot \Big(\nu T\Big) - \nabla \cdot (k\nabla T) + S \tag{10}$$

where c is the specific heat capacity, k the material thermal conductivity, ν the velocity, T the temperature field, ρ the material density, ∇T is the temperature gradient and S the heat source energy that will be not considered in this case. So in order, we see the **unsteady** term, a **convection** term and a **diffusion** term.
The temperature distribution $T(r,\theta,t)$ in cylindrical coordinates is then written as:

$$\rho c_p \frac{\partial T}{\partial t} = \rho c_p \left[\frac{1}{r}\frac{\partial (r\, v_r\, T)}{\partial r} + \frac{1}{r}\frac{\partial (v_\theta T)}{\partial \theta}\right] - \left[\frac{1}{r}\frac{\partial}{\partial r}\left(r k \frac{\partial T}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta}\right] \tag{11}$$

Here is noticeable once again how the heat transfer and velocity field are strictly correlated in our model. Since k and ρ are kept constant, the equation simplifies:

$$\frac{\partial T}{\partial t} = \left(v_r \frac{\partial T}{\partial r} + \frac{v_\theta}{r}\frac{\partial T}{\partial \theta}\right) - \alpha\left(\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2}\right) \tag{12}$$

where $\alpha = \frac{k}{\rho c_p}$ is the thermal diffusivity.
As values, we will be using the following, assuming them constant for approximation:

$k\left[\frac{W}{mK}\right]$	$c_p\left[\frac{J}{KgK}\right]$	$\alpha\left[\frac{mm^2}{s}\right]$	$\rho\left[\frac{kg}{m^3}\right]$
0.2366	1646.0	0.1239	1160

Table 3. Parameters for heat transfer

3.3 Boundary conditions

We impose two different sets of boundary conditions, in order to describe the behavior of the inner and outer cylinder in respect to the velocity, pressure (which is coupled with the velocity [5]) and the temperature. What we also take in account are the conditions around the thermometer.

Velocity field boundaries

For inner and outer boundaries:

- u_r satisfy with Dirichlet boundary condition on both boundaries:
  $(u_r)_{in} = 0$, $(u_r)_{out} = 0$
- u_θ satisfy with Dirichlet boundary condition on both boundaries:
  $(u_\theta)_{in} = 2.1$, $(u_\theta)_{out} = 0$
- p satisfy with Neumann boundary condition on both boundaries:
  $\nabla p \cdot n_{in} = 0$, $\nabla p \cdot n_{out} = 0$

For the boundaries of the obstacle:

- u_r satisfy with Dirichlet boundary condition on 4 boundaries:
  $(u_r)_{smallr} = 0$, $(u_r)_{bigr} = 0$, $(u_r)_{small\theta} = 1$, $(u_r)_{big\theta} = -1$
- u_θ satisfy with Dirichlet boundary condition on 4 boundaries:
  $(u_\theta)_{smallr} = -1$, $(u_\theta)_{bigr} = 1$, $(u_\theta)_{small\theta} = 0$, $(u_\theta)_{big\theta} = 0$
- p satisfy with Neumann boundary condition on 4 boundaries:
  $\nabla p \cdot n = 0$ on all boundaries

Heat transfer boundaries

For inner and outer boundaries:

- T satisfy with Dirichlet boundary condition on the outer boundary:
  $T_{out} = -20°\text{C}$
  The outer temperature is arbitrary and fixed.
- T satisfy with Neumann boundary condition on the inner boundary:
  $\nabla T \cdot n = 0$
  T_{in} is initially imposed as the fluid temperature.

For the boundaries of the obstacle:

- T satisfy with Neumann boundary condition on all boundaries:
  $\nabla T \cdot n = 0$
  The obstacle is insulated.

[5] See "SIMPLE algorithm" in **Chapter 3.1 - "Velocity field equations"**

3.4 Finite difference method

For building the model, the spatial discretization has been performed through Finite Difference Method (FDM). The aim is to obtain a **matrix A**, which includes all the needed the information about discretized PDE's, and a **vector b**, representing the right hand side vector. [6]. We can then visualize the systems for the discretization of **velocity**, **pressure** and **temperature** as following:

$$Au = b \qquad \text{where: } \mathbf{A}\text{=2n·2n, } \mathbf{u}\text{=2n, } \mathbf{b}\text{=2n} \tag{13}$$

$$A_p p = b_p \qquad \text{where:}\mathbf{A}\text{=n·n, } \mathbf{p}\text{=n, } \mathbf{b}\text{=n} \tag{14}$$

$$A_T T = b_T \qquad \text{where:}\mathbf{A}\text{=n·n, } \mathbf{T}\text{=n, } \mathbf{b}\text{=n} \tag{15}$$

Where n si the number of grid points.

Flow field discretization

For what concerns the velocity field, we will show just the derivation for the **diffusive** component as exemplary: eventually, the convection and presssure terms will be discretized in the same fashion. Taking **Equation (6)** as reference, we write the Laplace operator:

$$\Delta u = \begin{bmatrix} \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u_r}{\partial r}\right) - \frac{u_r}{r^2} + \frac{1}{r^2}\frac{\partial^2 u_r}{\partial \theta^2} - \frac{2}{r^2}\frac{\partial u_\theta}{\partial \theta} \\ \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u_\theta}{\partial r}\right) - \frac{u_\theta}{r^2} + \frac{1}{r^2}\frac{\partial^2 u_\theta}{\partial \theta^2} + \frac{2}{r^2}\frac{\partial u_r}{\partial \theta} \end{bmatrix} \tag{16}$$

Then we apply the finite difference discretization for first and second order derivatives:

- **Radial Direction:**

$$\frac{\partial u_r}{\partial r} \approx \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta r} \qquad \frac{\partial u_\theta}{\partial \theta} \approx \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta \theta} \qquad \frac{\partial^2 u_r}{\partial \theta^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta\theta)^2}$$

- **Angular Direction:**

$$\frac{\partial u_\theta}{\partial r} \approx \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta r} \qquad \frac{\partial u_r}{\partial \theta} \approx \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta \theta} \qquad \frac{\partial^2 u_\theta}{\partial \theta^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta\theta)^2}$$

For better understanding we combine all the terms together. The following expression accounts for the **radial direction**, since the azimuthal one follows the same structure and logic. We then have, using a re arranged formulation:

$$\frac{\partial^2 u_r}{\partial \theta^2} + \frac{1}{r}\frac{\partial u_r}{\partial r} - \frac{u_r}{r^2} - \frac{2}{r^2}\frac{\partial u_\theta}{\partial \theta} \tag{17}$$

[6] This matrix-vector logic is as well applied to **Chapter 3.5 - "Time discretization"**

$$\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{(\Delta\theta)^2}+\frac{1}{r}\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{(\Delta\theta)^2}-\frac{u_r}{r^2}-\frac{2}{r^2}\frac{u_{i+1,j}-u_{i-1,j}}{2\Delta\theta} \tag{18}$$

After computing the inner nodes, the same machinery is applied to the boundary conditions.

Tempreture field discretization

In an analogous way, from **Equation (12)**, we analyze the advective and diffusive terms, both for the radial and angular direction.

- **Convection**:

  Radial Convection Term: $\dfrac{\partial T}{\partial r}\approx\dfrac{T_{i+1,j}-T_{i-1,j}}{2\Delta r}$

  Angular Convection Term: $\dfrac{\partial T}{\partial \theta}\approx\dfrac{T_{i,j+1}-T_{i,j-1}}{2\Delta\theta}$

- **Diffusion**:

  Radial Diffusion Term: $\dfrac{\partial T}{\partial r}\approx\dfrac{T_{i+1,j}-T_{i-1,j}}{2\Delta r}$

  Angular Diffusion Term: $\dfrac{\partial^2 T}{\partial \theta^2}\approx\dfrac{T_{i,j+1}-2T_{i,j}+T_{i,j-1}}{(\Delta\theta)^2}$

Altogether, we derive the following discretization:

$$\left(v_r\frac{T_{i+1,j}-T_{i-1,j}}{2\Delta r}+\frac{v_\theta}{r}\frac{T_{i,j+1}-T_{i,j-1}}{2\Delta\theta}\right)-\alpha\left(\frac{1}{r}\frac{T_{i+1,j}-T_{i-1,j}}{2\Delta r}+\frac{1}{r}\frac{T_{i,j+1}-2T_{i,j}+T_{i,j-1}}{(\Delta\theta)^2}\right) \tag{19}$$

3.5 Time discretization

For the time stepping scheme, different methods has been implemented to see which one would have given the best convergence and more accurate results. The methods include **steady-state**, **RK4**, **implicit**, and **Crank-Nicolson (θ method)**. And we chose RK4 method at the end.

- **Steady-state**: the linear system is solved directly.

  $$\mathbf{A}\mathbf{x}_{\text{new}}=\mathbf{b} \tag{20}$$

- **RK4:** approximation of the solution using four intermediate steps to estimate the

next value.

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x} - \mathbf{b} \qquad \mathbf{x}_{new} = \mathbf{x}_n + \frac{1}{6}\Delta t\,(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

For the coefficients :

$$\begin{aligned}
\mathbf{k}_1 &= \mathbf{A}\mathbf{x}_n - \mathbf{b} \\
\mathbf{k}_2 &= \mathbf{A}\left(\mathbf{x}_n + \frac{1}{2}\Delta t\mathbf{k}_1\right) - \mathbf{b} \\
\mathbf{k}_3 &= \mathbf{A}\left(\mathbf{x}_n + \frac{1}{2}\Delta t\mathbf{k}_2\right) - \mathbf{b} \\
\mathbf{k}_4 &= \mathbf{A}\,(\mathbf{x}_n + \Delta t\mathbf{k}_3) - \mathbf{b}
\end{aligned}$$

- **Implicit**: the system is solved at each step.

  $$(I - \Delta t\mathbf{A})\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} - \Delta t\mathbf{b}$$

- **Crank-Nicolson**: combination of both implicit and explicit methods.

  $$(I - \theta\Delta t\mathbf{A})\mathbf{x}_{\text{new}} = (I + (1-\theta)\Delta t\mathbf{A})\mathbf{x}_{\text{old}} - \Delta t\mathbf{b}$$

  where θ is a parameter between 0 and 1. When $\theta = 0$, it corresponds to the explicit method; when $\theta = 1$, it corresponds to the implicit method.

4. Results and Discussion

4.1 Graphic results - Velocity field

First, we implemented a velocity of 1 rad/s in order to visualize and make a comparison between **radial** and **angular velocity**, as shown in **Figure (7)** and **Figure (8)**. We notice

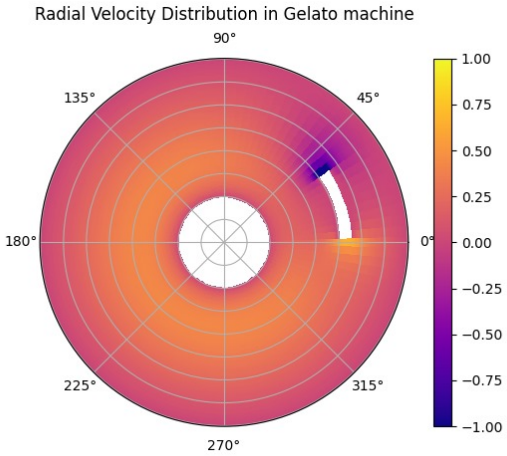

Figure 7. Radial velocity distribution

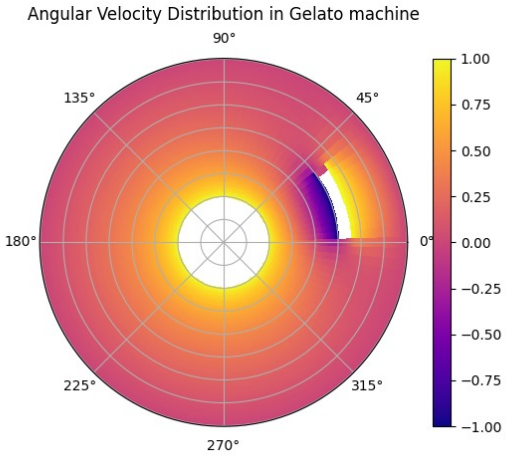

Figure 8. Angular velocity distribution

how the velocity distribution is heavily influenced by the presence of the obstacle, which distrupts the smoothness of the velocity distribution.
Then we investigate further the angular velocity while increasing the rotor's velocity of rotation in **Figure (9)** and **Figure (10)**. Is evident how increasing the velocity of the inner cylinder, the solution looses accurancy.

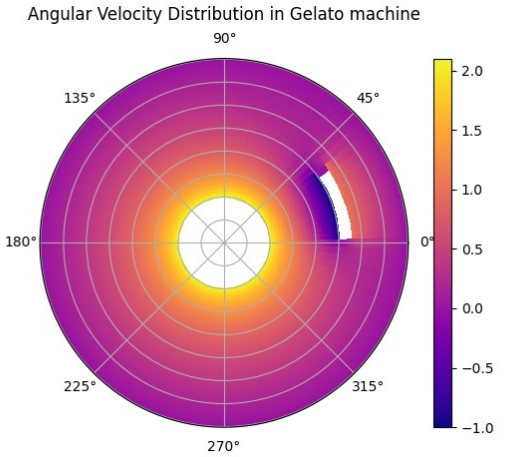

Figure 9. Angular velocity for 2 rad/s

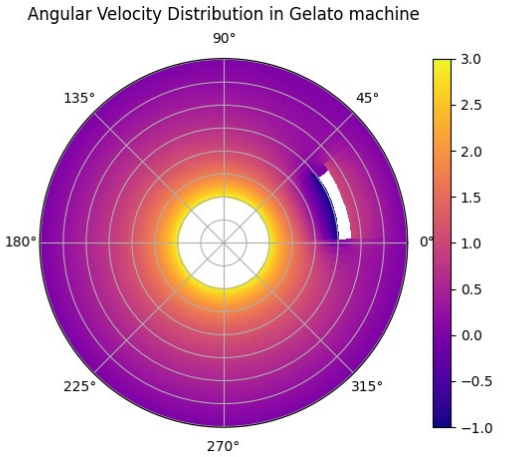

Figure 10. Angular velocity for 3 rad/s

4.2 Graphic results - Temperature field

Now we focus on the analysis of the temperature field. Once again, the obstacle influences not only the velocity distribution but also the temperature, leaving the field not homogeneous.

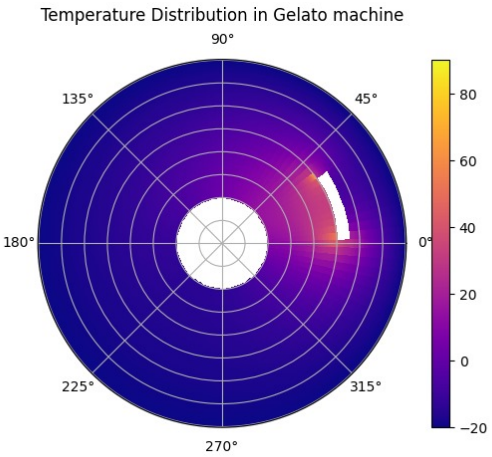

Figure 11. Initial temperature $T = -20°C$

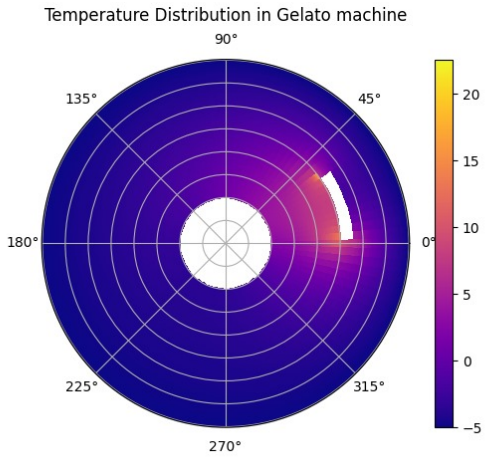

Figure 12. Initial temperature $T = -5°C$

Lastly, we visualize the **temperature** during the pasteurization process.

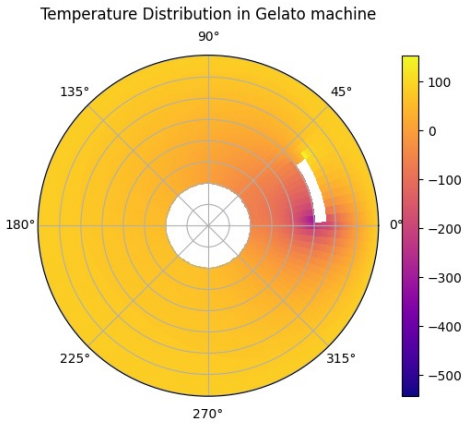

Figure 13. Temperature distribution during pasteurization.

4.3 Time stepping scheme choice

As showed in **Chapter 3.5 - "Time discretization"**, we implemented different options to analyze which one would have given us the best convergence in reasonable computing time. We established that using an implicit time scheme gives a better convergence but the computing time is prohibitive for multiple analysis. We opted for an explicit scheme, such

as **RK4**, because even if the convergence is not extremely accurate, the computation was not as heavy.

4.4 Side Remarks

Due to the complexity of the computation, we implemented two models, one accounting for the pressure and one without. The best results for convergence were given by the code not accounting for the pressure but, for theoretical coherence, the plots were performed accounting for the pressure as well. With further adjustments, the code attached in the archive will able to define the dependency of all the three fields: **pressure**, **velocity** and **temperature**.

5. Conclusions

This analysis was aimed to show the complexity of the production of such a common product and the physics behind it.
We analyzed the **rheological phenomena** and stated the relations between parameters: how the viscosity is influenced by velocity and temperature, for example. We also elaborated an approximation for the **geometry**, extrapolating the functionality of a Taylor-Couette reactor, and implementing a suitable grid. Finally, trough a **numerical analysis**, we established the best time stepping scheme and implemented the model trough finite element discretization and coherent boundary conditions, obtaining a graphical visualisation of the process.

Acknowledgments

We would like to thank Professor Dr. Gregoire Varillon and Dr. Camilo Silva for the insights regarding the theoretical and practical application of our study case. Even if we picked a peculiar analysis to perform, they gave us the background tools to perform a systematic study.

6. Code Appendix

In order to not having a repetitive and redundant list, since the parameters are coded with the same logic, here the more exemplary applications.

6.1 Mesh and define index

```
# Circular Grid
        self.Index = np.zeros((self.dimTheta, self.dimR),dtype=int)
        self.current_number = 0
        for i in range(self.dimTheta):
            for j in range(self.dimR):
                if self.theta_range[0] < i < self.theta_range[1] and self
   .r_range[0] < j < self.r_range[1]:
                    self.Index[i, j] = -1
```

```
                else:
                    self.Index[i, j] = self.current_number
                    self.current_number += 1
     # define a index function: ii represents Theta, jj represents r
    def index(self, n, ii, jj):

        if ii < 0:
            ii = ii + self.dimTheta
        elif ii > self.dimTheta-1:
            ii = ii - self.dimTheta

        if self.Index[ii,jj] == -1:
            print(f'ii is {ii}, jj is {jj}')
            raise ValueError("The point in the obstacle should not be called!")

        index = n*self.current_number + self.Index[ii,jj]

        return index
```

6.2 Momentum conservation equations

```
def set_bulk(self):
    # Loop through all theta indices
    for ii in range(self.dimTheta):
        # Loop through all radial indices except the boundaries
        for jj in range(1, self.dimR - 1):
            # Check if the current point is outside the defined obstacle range
            if (ii < self.theta_range[0] or ii > self.theta_range[1]) or (jj < self.r_range[0] or jj > self.r_range[1]):

                # Diagonal element of matrix A for index (0, ii, jj)
                self.A[self.index(0, ii, jj), self.index(0, ii, jj)] = (
                    2 * self.mu / self.dr ** 2 +
                    2 * self.mu / ((self.R_in + jj * self.dr) * self.dTheta) ** 2 +
                    2 * self.mu / (self.R_in + jj * self.dr) ** 2
                )

                # Right neighbor in radial direction for index (0, ii, jj)
                self.A[self.index(0, ii, jj), self.index(0, ii, jj + 1)] = (
                    -self.mu / self.dr ** 2 -
                    self.mu / (2 * (self.R_in + jj * self.dr) ** 2) +
                    self.rho * self.u[self.index(0, ii, jj)] / (2 * self.dr)
```

```
                )

                # Left neighbor in radial direction for index (0, ii, jj)
                self.A[self.index(0, ii, jj), self.index(0, ii, jj - 1)] = (
                    -self.mu / self.dr ** 2 +
                    self.mu / (2 * (self.R_in + jj * self.dr) ** 2) -
                    self.rho * self.u[self.index(0, ii, jj)] / (2 * self.dr)
                )

                # Upper neighbor in angular direction for index (0, ii, jj)
                self.A[self.index(0, ii, jj), self.index(0, ii + 1, jj)] = (
                    -self.mu / ((self.R_in + jj * self.dr) * self.dTheta) ** 2 +
                    self.rho * self.u[self.index(1, ii, jj)] / (2 * self.dTheta * (self.R_in + jj * self.dr))
                )

                # Lower neighbor in angular direction for index (0, ii, jj)
                self.A[self.index(0, ii, jj), self.index(0, ii - 1, jj)] = (
                    -self.mu / ((self.R_in + jj * self.dr) * self.dTheta) ** 2 -
                    self.rho * self.u[self.index(1, ii, jj)] / (2 * self.dTheta * (self.R_in + jj * self.dr))
                )

                # Diagonal element of matrix A for the second index component (1, ii, jj)
                self.A[self.index(0, ii, jj), self.index(1, ii + 1, jj)] = (
                    -self.mu / ((self.R_in + jj * self.dr) ** 2 * self.dTheta)
                )
                self.A[self.index(0, ii, jj), self.index(1, ii, jj)] = (
                    -self.rho * self.u[self.index(1, ii, jj)] / (self.R_in + jj * self.dr)
                )
                self.A[self.index(0, ii, jj), self.index(1, ii - 1, jj
```

6.3 Boundary conditions - Inner cylinder

```
# Inner cylinder - ROTOR
   def set_inner(self, u_in=0.0):
```

```
        for ii in range(self.dimTheta):
            self.A[self.index(1,ii,0), self.index(1,ii,0)] = 1
            self.b[self.index(1,ii,0)] = u_in
            self.u[self.index(1,ii,0)] = u_in

            self.A[self.index(0,ii,0), self.index(0,ii,0)] = 1
            self.b[self.index(0,ii,0)] = 0

            f = 1/(2*self.dr)
            self.At[self.index(0,ii,0), self.index(0,ii,0)] = -3*f
            self.At[self.index(0,ii,0), self.index(0,ii,1)] = 4*f
            self.At[self.index(0,ii,0), self.index(0,ii,2)] = -1*f
```

6.4 Boundary conditions - Outer cylinder

```
# Outer cylinder - STATOR
   def set_outer(self, T_out):
        for ii in range(self.dimTheta):
            self.A[self.index(1,ii,self.dimR-1),self.index(1,ii,self.dimR
   -1)] = 1
            self.b[self.index(1,ii,self.dimR-1)] = 0

            self.A[self.index(0,ii,self.dimR-1),self.index(0,ii,self.dimR
   -1)] = 1
            self.b[self.index(0,ii,self.dimR-1)] = 0

            f = 1/(2*self.R_out*self.dTheta)
            self.Ap[self.index(0,ii,self.dimR-1),self.index(0,ii,self.
   dimR-1)] = -3*f
            self.Ap[self.index(0,ii,self.dimR-1),self.index(0,ii,self.
   dimR-2)] = 4*f
            self.Ap[self.index(0,ii,self.dimR-1),self.index(0,ii,self.
   dimR-3)] = -1*f

            self.At[self.index(0,ii,self.dimR-1),self.index(0,ii,self.
   dimR-1)] = 1
            self.bt[self.index(0,ii,self.dimR-1)] = T_out
            self.T[self.index(0,ii,self.dimR-1)] = T_out
```

6.5 Time stepping scheme - RK4

```
        for t in np.arange(0,self.t_end,self.dt):
            u1 = u_old + 1/2*self.dt*(self.A @ u_old - self.b)
            u2 = u_old + 1/2*self.dt*(self.A @ u1 - self.b)
            u3 = u_old + self.dt*(self.A @ u2 - self.b)
            u_new = u_old + 1/6*self.dt*(self.A @ u_old + 2*self.A @ u1 +
    2*self.A @ u2 + self.A @ u3) - self.dt*self.b
```

6.6 Obstacle implementation

```
def set_obs(self, u_o=0):
    # Loop through the radial range of the obstacle
    for jj in range(self.r_range[0], self.r_range[1] + 1):
        # Set boundary conditions at the start and end of the angular range for index (0, theta_range, jj)
        self.A[self.index(0, self.theta_range[0], jj), self.index(0, self.theta_range[0], jj)] = 1
        self.A[self.index(0, self.theta_range[1], jj), self.index(0, self.theta_range[1], jj)] = 1
        self.b[self.index(0, self.theta_range[0], jj)] = u_o
        self.b[self.index(0, self.theta_range[1], jj)] = -u_o
        self.u[self.index(0, self.theta_range[0], jj)] = u_o
        self.u[self.index(0, self.theta_range[1], jj)] = -u_o

        # Set boundary conditions at the start and end of the angular range for index (1, theta_range, jj)
        self.A[self.index(1, self.theta_range[0], jj), self.index(1, self.theta_range[0], jj)] = 1
        self.A[self.index(1, self.theta_range[1], jj), self.index(1, self.theta_range[1], jj)] = 1

        # Setting up the differentiation matrix At for the obstacle at start and end of the angular range
        self.At[self.index(0, self.theta_range[0], jj), self.index(0, self.theta_range[0], jj)] = -3 / (2 * (self.R_in + jj * self.dr) * self.dTheta)
        self.At[self.index(0, self.theta_range[0], jj), self.index(0, self.theta_range[0] - 1, jj)] = 4 / (2 * (self.R_in + jj * self.dr) * self.dTheta)
        self.At[self.index(0, self.theta_range[0], jj), self.index(0, self.theta_range[0] - 2, jj)] = -1 / (2 * (self.R_in + jj * self.dr) * self.dTheta)
        self.At[self.index(0, self.theta_range[1], jj), self.index(0, self.theta_range[1], jj)] = -3 / (2 * (self.R_in + jj * self.dr) * self.dTheta)
        self.At[self.index(0, self.theta_range[1], jj), self.index(0, self.theta_range[1] + 1, jj)] = 4 / (2 * (self.R_in + jj * self.dr) * self.dTheta)
        self.At[self.index(0, self.theta_range[1], jj), self.index(0, self.theta_range[1] + 2, jj)] = -1 / (2 * (self.R_in + jj * self.dr) * self.dTheta)

    # Loop through the angular range of the obstacle
    for ii in range(self.theta_range[0], self.theta_range[1] + 1):
        # Set boundary conditions at the start and end of the radial range for index (0, ii, r_range)
        self.A[self.index(0, ii, self.r_range[0]), self.index(0, ii, self
```

```
    .r_range[0])] = 1
28          self.A[self.index(0, ii, self.r_range[1]), self.index(0, ii, self
    .r_range[1])] = 1
29
30          # Set boundary conditions at the start and end of the radial
    range for index (1, ii, r_range)
31          self.A[self.index(1, ii, self.r_range[0]), self.index(
```

6.7 Energy conservation equations

```
1   def set_t(self):
2     # Loop through all theta indices
3     for ii in range(self.dimTheta):
4         # Loop through all radial indices except the boundaries
5         for jj in range(1, self.dimR - 1):
6         # Check if the current point is outside the defined obstacle
    range
7             if (ii < self.theta_range[0] or ii > self.theta_range[1]) or
    (jj < self.r_range[0] or jj > self.r_range[1]):
8
9         # Diagonal element of matrix At
10            self.At[self.index(0, ii, jj), self.index(0, ii, jj)] = (
11                    2 * self.alpha / (self.dr) ** 2 +
12                    2 * self.alpha / (self.dTheta * (self.R_in + jj *
    self.dr)) ** 2
13                )
14
15                # Element to the right of the diagonal in matrix At
16                self.At[self.index(0, ii, jj), self.index(0, ii, jj + 1)]
     = (
17                    self.u[self.index(0, ii, jj)] / (2 * self.dr) -
18                    self.alpha / (self.dr) ** 2 -
19                    self.alpha / (2 * self.dr * (self.R_in + jj * self.dr
    ))
20                )
21
22                # Element to the left of the diagonal in matrix At
23                self.At[self.index(0, ii, jj), self.index(0, ii, jj - 1)]
     = (
24                    -self.u[self.index(0, ii, jj)] / (2 * self.dr) -
25                    self.alpha / (self.dr) ** 2 +
26                    self.alpha / (2 * self.dr * (self.R_in + jj * self.dr
    ))
27                )
28
29                # Element above the diagonal in matrix At (next theta
    index)
30                self.At[self.index(0, ii, jj), self.index(0, ii + 1, jj)]
```

```
 = (
                    self.u[self.index(1, ii, jj)] / (2 * self.dTheta * (
self.R_in + jj * self.dr)) -
                    self.alpha / (self.dTheta * (self.R_in + jj * self.dr
)) ** 2
                )

                # Element below the diagonal in matrix At (previous theta
 index)
                self.At[self.index(0, ii, jj), self.index(0, ii - 1, jj)]
 = (
                    -self.u[self.index(1, ii, jj)] / (2 * self.dTheta * (
self.R_in + jj * self.dr)) -
                    self.alpha / (self.dTheta * (self.R_in + jj * self.dr
)) ** 2
```

References

[1] Wikipedia. Convection–diffusion equation. *Wikipedia*, 2024.

[2] Alejandro De la Cruz Martínez et al. Estimation of ice cream mixture viscosity during batch crystallization in a scraped surface heat exchanger. *Processes*, 2020.

[3] H. Douglas Goff and Valerie J. Davidson. Flow characteristics and holding time calculations of ice cream mixes in htst holding tubes. *Journal of Food Protection*, 1992.

[4] Alan Leighton and O. E. Williams. The basic viscosity of ice-cream mixes. *The Journal of Physical Chemistry*, 1927.

[5] N. J. Rodríguez et al. Numerical modelling of unsteady convective–diffusive heat transfer with a control volume hybrid method. *Applied Mathematical Modelling*, 2009.

[6] Qammar Rubbab et al. Numerical simulation of advection–diffusion equation with caputo-fabrizio time fractional derivative in cylindrical domains: Applications of pseudo-spectral collocation method. *Alexandria Engineering Journal*, 2021.

[7] Muhammad Saqib et al. Computational solutions of two dimensional convection diffusion equation using crank-nicolson and time efficient adi. *American Journal of Computational Mathematics*, 2017.

[8] Jiri Blazek. Chapter 8 - boundary conditions. In Jiri Blazek, editor, *Computational Fluid Dynamics: Principles and Applications (Third Edition)*, pages 253–281. Butterworth-Heinemann, 2015.

[9] Claudia Cogné et al. Experimental data and modelling of thermal properties of ice creams. *Journal of Food Engineering*, 58(4):331–341, Aug 2003.

[10] Luis F. Zambrano-Mayorga et al. Influence of the formulation on the thermophysical properties and the quality parameters of dairy ice cream. *DYNA*, 86(208):117–125, 2019.