Natural Convection

Lawrence Smith | Ninad Mehta 05/06/2020

1 Introduction

This report details our analysis of problems involving fluid flow coupled to thermal energy transport through temperature-dependent buoyant forces. This type of two-way coupled flow problem is called *natural convection*, and is an excellent example of buoyancy driven flow. Buoyancy driven flow arises when a fluid with temperature dependent density is heated, causing local temperature increases and density mismatches. Under the influence of a gravitational field, this causes buoyant forces which vary spatially over the fluid, driving flow. This flow accelerates the transfer of energy between a heated object and its surroundings (as compared to pure diffusion) and is called natural convection. This is a fascinating type of flow because the flow fields do arise "naturally" from the temperature boundary conditions and geometry of the problem (rather than being prescribed by a constant body force or velocity boundary condition).

Natural convection is considered in three main contexts: practical, day to day engineering calculations, experimental studies, and numerical studies. In practical analysis the specification of the heat transfer rate between a surface and a fluid is typically handled by assuming a convective flux at the boundary, and the modeling the rate of thermal energy exchange here using a convection coefficient – a parameter which determines the rate at which energy is transferred between the solid and the fluid. Thermal solutions can be very sensitive to the choice of this convection coefficient, and it can be a source of large error in heat transfer analyses. However the time savings of using a simplification like a constant convection coefficient are large as compared to actually simulating the buoyancy driven flow for a given geometry, and this is still a widespread technique.

Many experimental studies have been conducted to guide the choice of convection coefficients for various geometries (e.g. heated cylinders, heated or cooled plates with various inclination angles, ect). These experimental studies produce volumes of data that correlate the heat transferred between a solid and fluid to the temperature difference between them through various dimensionless numbers such as Grashof and Prandtl. By creating best-fits to these datasets, many phenomenological relationships have been developed, such this nonlinear expression published by Churchill for computing the heat transfer coefficient for a heated sphere: which holds for Prandtl numbers $Pr > 0.5$ and Rayleigh

$$\overline{Nu}_D = 2 + \frac{0.589\, Ra_D^{0.25}}{\left[1 + \left(\frac{0.469}{Pr}\right)^{9/16}\right]^{4/9}} \qquad \overline{Nu}_D = \frac{\bar{h}\, D}{k}$$

numbers $Ra < 1e11$. While experimental studies have been invaluable in building natural convection relationships for simple geometries, they are inherently limited. For example, if the geometry of a particular problem cannot be decomposed into simple shapes that can each be modeled using an existing relationship like the one above another technique is needed.

This brings us to the final context in which natural convection is studied - using numerical models. Given the generality and flexibility of the finite element method, we can directly compute an (approximate) solution to a coupled fluid flow / heat transfer problem for any geometry or set of fluid properties. Numerical solutions to natural convection problems have been extensively studied, both to corroborate and extend physical experiments as well as to solve new, previously unstudied problems. The current cutting edge of this research is more advanced than what we will show in this paper, and

some possible extensions of our work are discussed in the Conclusions section. Along the way, we will identify points where our approach differs from existing numerical work.

2 Background: The Physics of Natural Convection

We choose to model the balance of fluid momentum, stress, pressure, and viscous forces using the Navier Stokes Equations, along with the conservation of mass:

$$\rho_0 \frac{Du}{Dt} = -\nabla p + b + \mu \nabla^2 u; \nabla u = 0$$

The body force b represents the spatially varying buoyant force, which naturally is temperature dependent. We choose to model this force as directly proportional to the local fluid temperature through a constant called the coefficient of thermal expansion α.

$$b = \rho_0(1 - \alpha(T - T_0))g$$

Notably, the non-constant, temperature-dependent density does *not* show up in the conservation of mass equation or in the conservation of momentum terms of the the Navier Stokes Equations. This is referred to as the Boussinesq approximation - we *only* consider the effects of nonuniform density on the body force term. In other recent works, this assumption is not made, and the compressible Navier-Stokes Equations are solved using the finite volume method.

To model the conservation of thermal energy , we use the heat diffusion-advection equation:

$$\frac{DT}{Dt} = \kappa \nabla^2 T$$

At the start of the project, we derived the weak form of this coupled system of partial differential equations in both the steady and unsteady form, under the assumption of zero Neumann boundary conditions. We could not find a detailed derivation of the weak form, so we have included one here in our Appendix. We have also included the exact syntax for representing this weak form in a FEnics-compatible code, since we could not find this in any references or online resources either.

Significant additional complexity is introduced when implementing the full, unsteady version of our system of equations numerically, due the multiple possible methods for time integration and several operator splitting techniques which may be applied. In fact this became the focus of our project, once we had validated the implementation of our code with steady examples. We will treat this topic separately

3 Validation: Recirculating Flow in Square Cavity

In order to verify our implementation of the governing equations of natural convection, we conducted a classical buoyancy-driven flow simulation involving a square chamber with deferentially heated vertical walls, as shown in Figure 1. The setup of this problem is described in the texbook [5], as well as in many papers including [1]. We used triangular elements which are quadratic in velocity and linear in pressure and temperature (for this and all other simulations present in this report) - which is a notable difference from our reference [5], which uses higher order quadratic elements. We generated solutions to this problem at identical conditions to those presented in our reference, in order to make a direct comparison for accuracy. We found very good agreement between the output of our simulations and published results, as seen in Figures 2 and 3.

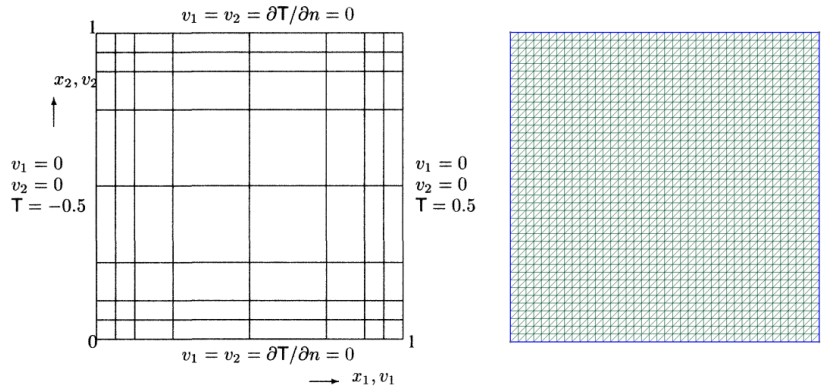

Figure 1: Description of domain and boundary conditions for buoyancy-driven flow in square cavity problem (left) from [5], and the structured mesh we used for simulation of our problem.

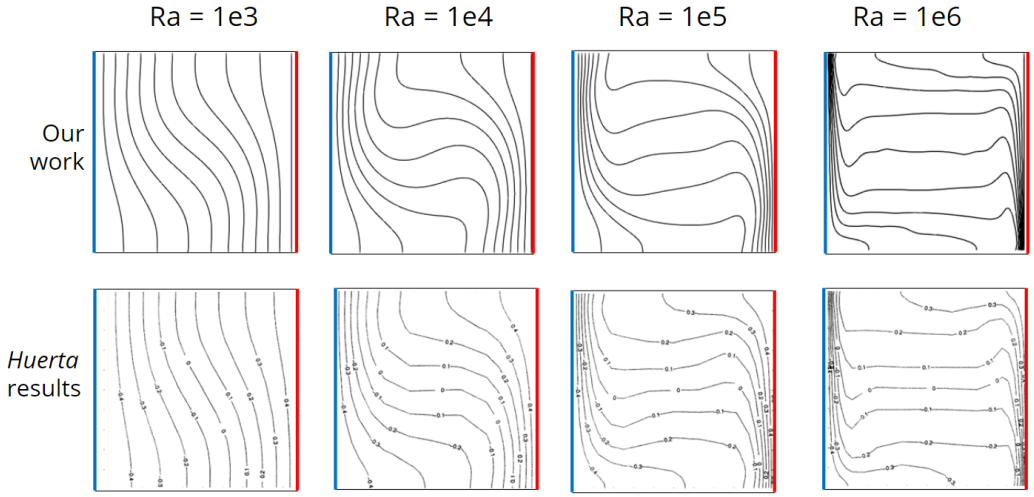

Figure 2: Representation of Isotherms from our analysis and Donea Huerta

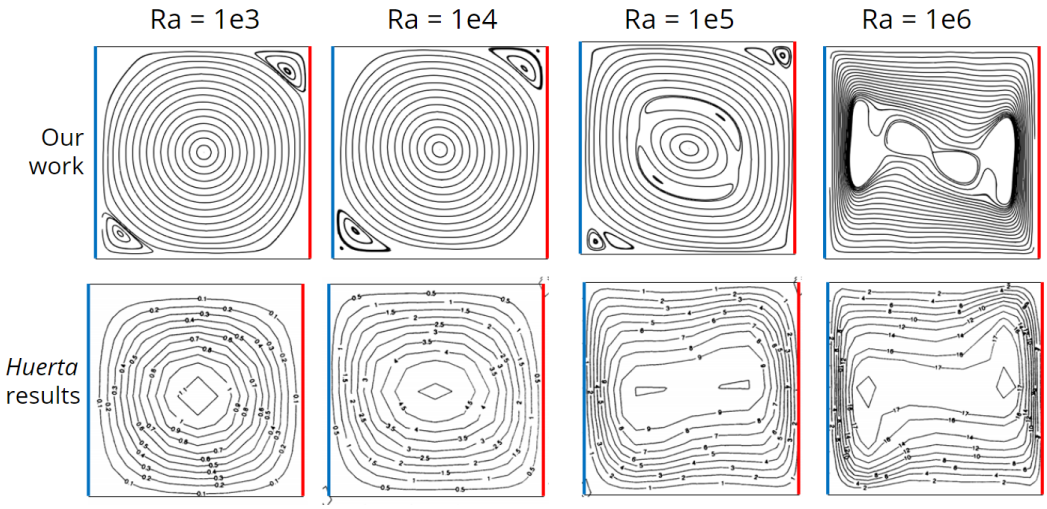

Figure 3: Comparison of vortex flow obtained with vortex flow of Donea Huerta using velocity streamlines

Additionally, we conducted a mesh refinement study to determine the mesh required to provide a useful solution to our problem. We generated a series of structured meshes similar to the example shown in Figure 1, with mesh edge lengths equal to $L/10$, $L/20$, $L/40$, and $L/80$. We solved for the flow profile and temperature in the square cavity, with $Ra = 1e5$ and $Pr = 1.0$. We chose to examine two quantities: Y-component of velocity along the horizontal centerline of the chamber,

and the X-component of the temperature gradient along the vertical left wall of the chamber. Both measures showed convergences as well as accuracy - they appear to approach a stable solution, and they resemble the results presented in the paper [1].

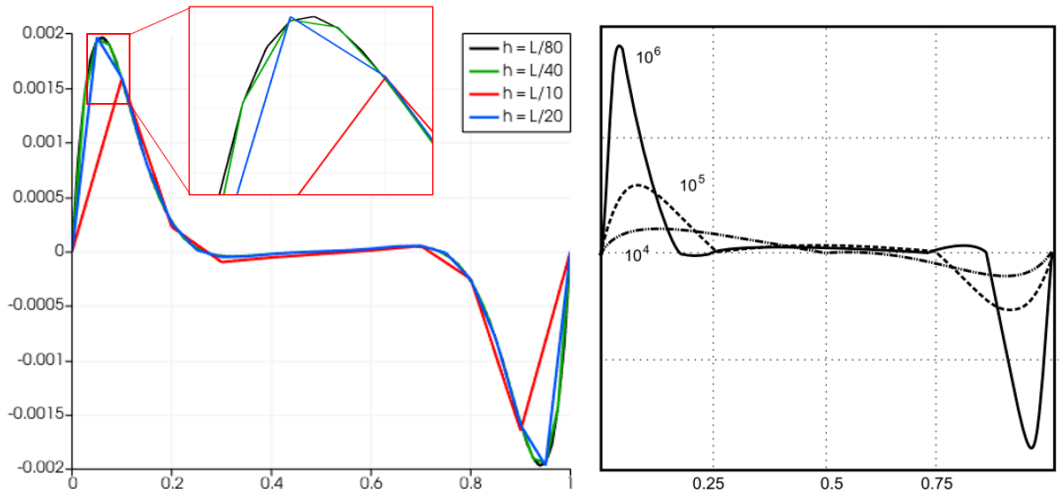

Figure 4: Mesh refinement study on the classical problem of natural convection in a square cavity. Y-component of velocity is plotted along the horizontal centerline of the cavity for four different mesh densities

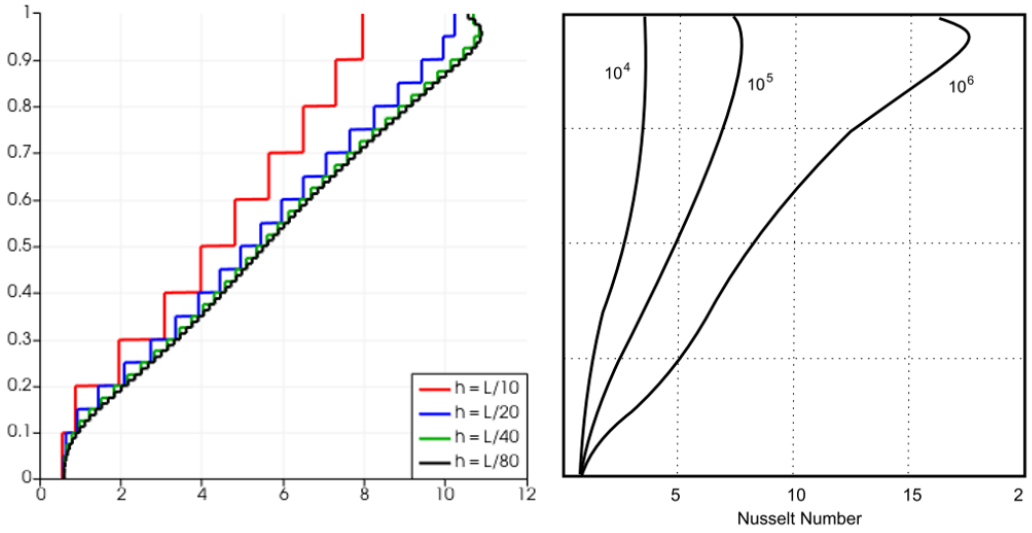

Figure 5: Mesh refinement study on the classical problem of convection in a square cavity. Dimensionless local Nusselt number $\frac{\delta T}{\delta x}$ is plotted along the left wall of the cavity, and compared to results from [1]. Above element size of $L/40$, the characteristic shape of the Nusselt number curve (especially the local maximum very near to the upper corner) is not recovered. We used element sizes no larger than $L/40$ for the remainder of this project.

Based on our comparisons to published data and mesh refinement study, we chose to call our simulation framework verified, and moved on to simulating more complex problems for which less verification data exist.

4 Results

4.1 Steady Convection from an Inclined Plate

We turned our now-validated natural convection tool towards a practical engineering application: analyzing the natural convective flow around a heated plated plate geometry.

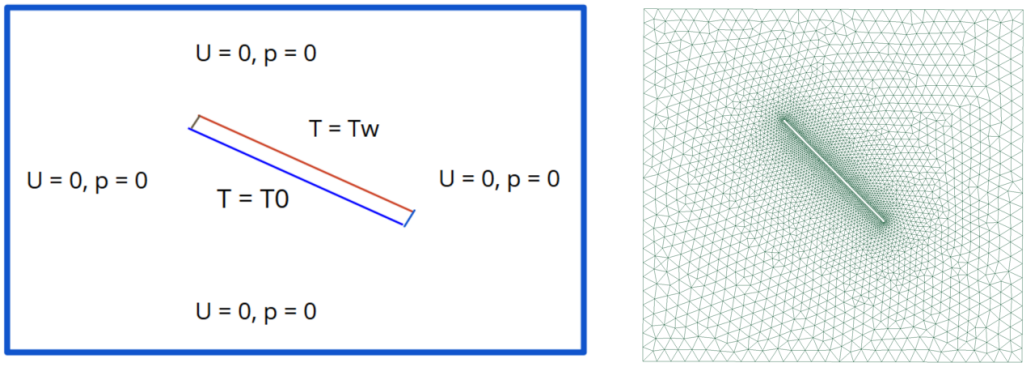

Figure 6: Model problem and discretization, with 45°inclination angle

This problem has been studied in both experimental and numerical contexts, and has many direct applications to practical engineering problems: for example, estimating the convection from inclined solar panel. Experimental results for the natural convection around the boundary of inclined surfaces have been reported by [6] and [2]. Studies also exist on the natural convection around an inclined plate with non-uniform temperature [2]. Dimensionless Nusselt number Nu, Grashof Gr number, and Prandtl number Pr are all critical for this problem, as they help describe the characteristics and flow regime of the stable flow field. Prandtl number signifies the thickness ratio of velocity and thermal boundary layers, while Grashof and Rayleigh numbers describe the ratio of driving forces (buoyancy) to dissipative forces (viscosity). For natural convection, it is found that the velocity and thermal boundary layers are of similar thickness for $Pr < 1$ and $Pr = 1$. The velocity boundary layer is always thicker than the thermal boundary layer for high Pr numbers [2].

The inclination angle of the plate also directly affects the flow field that develops, even for identical temperature boundary conditions. As reported by researcher Abhijit Guha, for a study on thermo-fluid dynamics of a vertical plate[3], the convective motion of the fluid is a direct result of buoyancy force. But for a horizontal plate[4], the buoyancy force generates a pressure gradient along the surface. This pressure gradient along the surface drives the laminar convective flow. This process is different from that of a vertical plate and is referred to as 'indirect natural convection'.

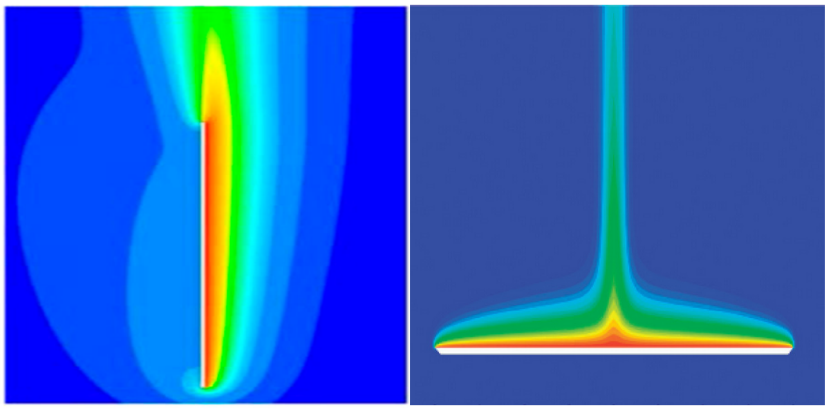

Figure 7: Contours of temperature for a vertical and horizontal plate, from [3] (left) and [4] (right)

In our problem, the right side of the plate is held at an elevated temperature and the other sides of the plate are held to ambient temperature. We apply no slip, no permeation boundary conditions to all surfaces, and a zero pressure boundary condition to the cavity walls. In order to model natural convection effectively, we chose a cavity size equal to twice the length of the plate, in order to limit the effects of the walls on the flow immediately around the plate. We hold the temperature and flow boundary conditions constant, and vary the plate inclination angle between 0°and 90°. We have excerpted from our full results four inclination angles, which capture the extremes of the observed flow behavior.

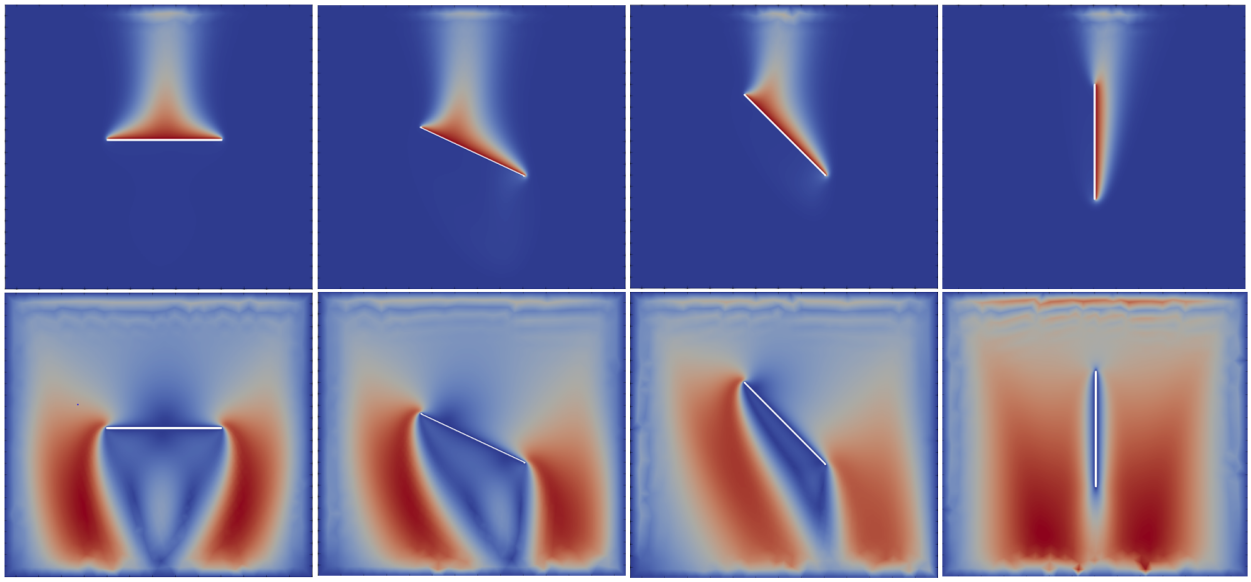

Figure 8: Temperature (upper) and velocity (lower) contours for the isothermally heated inclined plates, with inclination angle (from left to right) of $0°$, $25°$, $45°$, and $90°$

We observe several trends in the temperature and velocity data. For all inclination angles, a thermal boundary layer and a buoyant plume form on the heated side of the plate. The plume bends as it rises from the surface of the plate, and orients vertically no matter the plate inclination angle. The temperature inside the plume decreases as the distance from the plate increases. The temperature difference and inclination dictate the detachment point of the buoyant plume as it lifts off before reaching the edge of the plate. This detachment point moves towards the center of the plate as the inclination angle tends to 0°. For the horizontal plate case, the plume is exactly above the point of detachment.

For all angles, we also see zero movement of the fluid around the boundary of the computational domain, and maximum velocity around the corners of the plate where the air is pulled into the buoyant plume. Notably, we present results for velocity magnitude - the directional aspect of velocity is not covered in this paper. For a horizontal plate, there is symmetric nature seen in the temperature and velocity profile.

Our results agree qualitatively with those in [3] and [4], and the flow patterns we see match our fluid dynamics intuition. This study shows the utility of our simulation tool in steady natural convection problems, and could prove useful in practical engineering analysis of thermal systems. However, during this exercise we were continually faced with convergence issues, especially for simulations with high Rayleigh number. Interestingly, for high Ra, we saw inconsistent convergence trends across the inclination angles we tested. Simulations with smaller inclination angles were more likely to converge, even for identical Ra. Additionally, simulations with identical inclination and Ra were more likely to converge for higher Pr. For the results presented here we chose modest values of Ra and Pr to guarantee convergence across all inclination angles ($Ra = 1e4$, $Pr = 0.7$).

4.2 Unsteady Convection in Differentially Heated Cavity

Literature sources agree that solutions to the natural convection problem are not unique for high Ra [7][1]. Unfortunately, high Ra problems are perhaps the most interesting from a practical standpoint, as they involve the highest levels of actual heat transfer. The only recourse is to include the time derivatives of temperature and velocity that appear in our system of PDEs and solve the unsteady problem. For this effort, we have slightly modified the model problem described in Figure 1 to induce more complex and unsteady flow patterns. We have used air as the working fluid in this model, which gives us $Ra = 1.5e6$ and $Pr = 0.7$ - these conditions mean that we are likely to see significant flow effects but the flow will stay in the laminar regime.

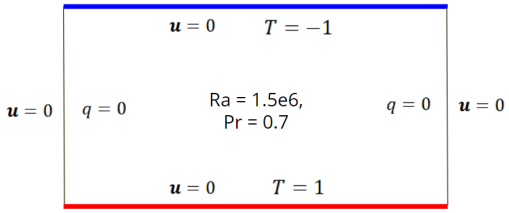

Figure 9: Model problem for unsteady convection: rectangular chamber with heated floor and cooled ceiling

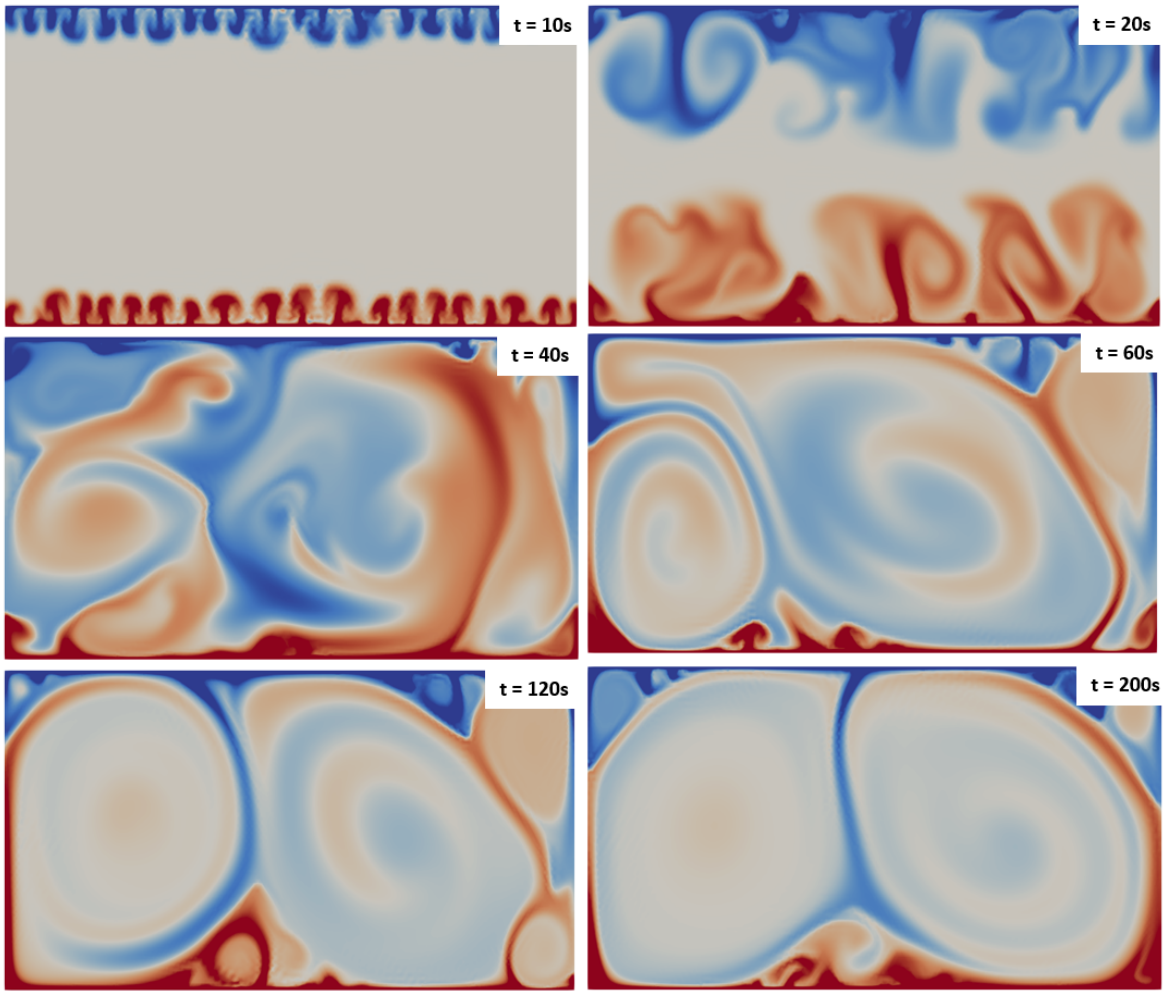

Figure 10: Temperature contour plots from unsteady simulation of free convection between two horizontal surfaces. Initially, small plumes erupt towards the center of the cavity. These collapse and merge into larger plumes, which meet in the center of the cavity and mix chaotically for a period of time. Eventually, metastable cells form and recirculate thermal energy from the top to the bottom of the cavity. These alternate in re circulation direction and persist with little change once they are formed.

The results from our unsteady simulations as visualized in Figure 10 show that we can simulate complex, unsteady fluid flows driven by natural convection. Our results qualitatively resemble the formation of Rayleigh-Benárd cells, a well-known convection phenomenon that has been extensively studied both empirically and numerically. We can also extract meaningful heat transfer results from the simulations - for example we can compute the rate of heat transfer to the cavity from a boundary by integrating the following expression along the boundary:

$$q = \int_{\Gamma} (\nabla T \cdot n) dx$$

We plot the *rate* of thermal energy released by the lower boundary and absorbed by the upper boundary as a function of time for $200s$ of simulation time.

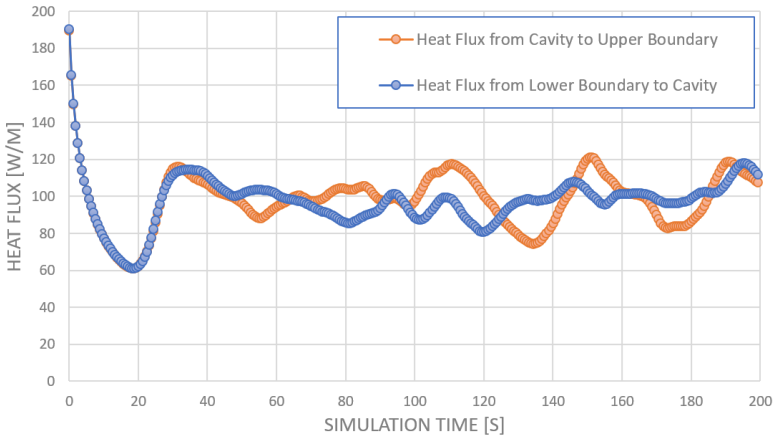

Figure 11: Extraction of heat fluxes at upper and lower boundaries of domain during unsteady simulation of model natural convection problem. Flux directions are defined such that both flows are positive for direct comparison. Peak fluxes occur as plumes erupt during start of simulation, and stabilize toward an average of $100W/m$ (this is a 2D simulation).

4.3 Numerical Implementation

While implementing and testing our unsteady solver, we experienced significant challenges related to convergence and computational cost. In our simulations, flow features of interest such as Rayleigh-Benárd cells did not arise until $100s+$ of simulation time had elapsed. Depending on the timestep size, this could require tens of thousands of incremental solves or more. So for the final phase of our project we turned an eye towards optimal *implementation* of our problem, with the goal of determining which method could deliver converged results in a minimum amount of time.

Convergence of a simulation is linked to many inter-related factors, such as timestep, local element size and order, and local fluid velocity. We can estimate the likelihood of convergence by examining the Courant number and Peclet number:

$$C = \frac{u\Delta t}{h} \qquad Pe = \frac{uh}{\alpha}$$

It's immediately obvious, however, that choosing a timestep which will provide a converged solution in the minimum amount of wall clock time is very challenging to do in a natural convection problem. Both C and Pe depend linearly on the velocity magnitude u - but the velocity field is not prescribed and not known at the outset of the problem; it arises naturally from the boundary conditions and geometry. In order to examine this, we added diagnostic code to our FeniCS implementation to compute C and Pe over the entire domain for each timestep, extract the maximum value, and log it to a .csv file.

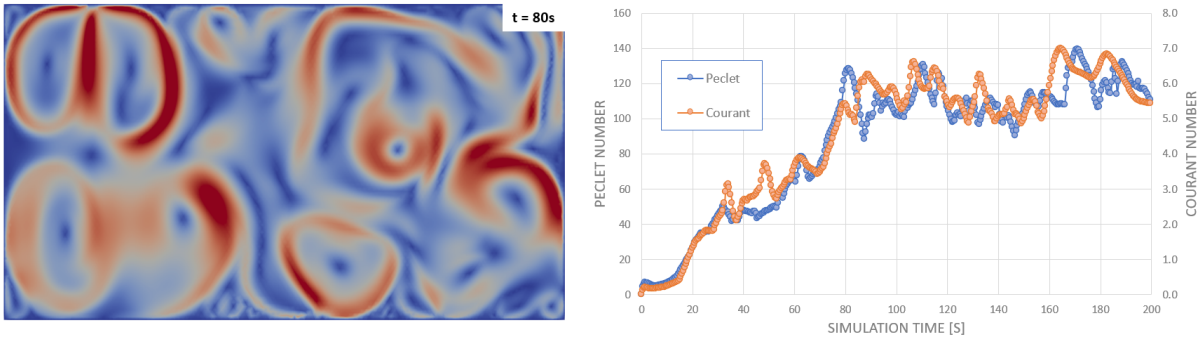

Figure 12: Velocity magnitude in the model problem domain at $t = 80s$ (left), and global maximum C and Pe plotted over 200s of simulation time (right). Clearly, C and Pe vary unpredictably throughout the simulation (as does u) - so choosing a constant Δt is challenging.

Next, we implemented three solution algorithms in FeniCS and benchmarked them by simulating our model unsteady problem with the same boundary conditions and computational mesh, on the same machine. We used the boundary conditions and geometry described in Figure 9 and a mesh of $12k$ nodes ($50k$ DoF), and we simulated using a single core on an AMD Ryzen 3950 processor running at 4.6GHz. We used Python commands to measure and log the duration of each timestep's solve and log this data to a .csv file. A description of the three algorithms we implemented follows.

4.3.1 Monolithic Solve

We implemented a monolithic solve of our system of PDEs by combining the Navier Stokes (NS) and Conservation of Thermal Energy (TH) equations through addition. We used the $\theta - Galerkin$ method to integrate forward in time, with the same θ value applied to both the velocity and temperature derivative (though in theory, differnet θ values could be used for each derivative). We tested both Crank-Nicholson and Backward Euler integration, and selected Backward Euler because it seemed to offer a better tradeoff between stability and solve time. The full weak form derivation and FeniCS syntax is given in Appendix B.

4.3.2 Operator Split Solve

We implemented a four-part operator split solve which was inspired by guidance from [5], and we have broken it into steps 1, 2a, 2b, and 2c. The full weak form derivation and FeniCS syntax is given in Appendix C.

1. Backward Euler step on the full Thermal Conservation of Energy equation. *Huerta*[5] advises splitting this solve into two steps, and performing a 3rd order Taylor Galerkin step on the temperature derivative using only the convective terms in the *TH* equation to compute an incremental temperature distribution, followed by a Crank-Nicholson step using the diffusive TH terms to compute the final temperature. However, based on our initial Solver profiling, we found that this TH step *never* took more than 3% of the total solve time for a given timestep, so we elected to stick with Backward Euler because of its superior stability.

2. Chorin-Temam Operator Split Solve on the full Navier Stokes Equation using Incremental Pressure Correction Scheme (IPCS). This involves 2a) a solve of the incremental velocity profile using the NS momentum terms, 2b) a correction using the pressure terms, and 2c) a final solve for the end-of-step velocity using the computed pressure. We chose this method because of its simplicity and popularity, and because we covered it in class. *Huerta*[5] recommends instead breaking this into two steps: a Crank-Nicholson step on the NS momentum, body force, and viscous terms and a Backward Euler step on the NS pressure and incomprehensibility terms.

4.3.3 Adaptive-Timestep Monolithic Solve

As we analyzed the results from our previous two solution methods, it became clear that neither of them could ever be optimal for an unsteady natural convection problem for one simple reason: they both use a constant value of Δt. In order to prevent the solution from diverging, Δt must be set lower than the lowest value required over the entire duration of the simulation. From Figure 15, it's easy to see that at some points in the simulation, C is large, requiring a small Δt, and at other points C is small - meaning larger Δt would suffice.

We implemented an adaptive-timestepping scheme in our monolithic solver, whereby the value of Δt is dynamically adjusted at the beginning of each timestep based on the maximum value of C computed from the velocity profile solved for in the previous timestep.

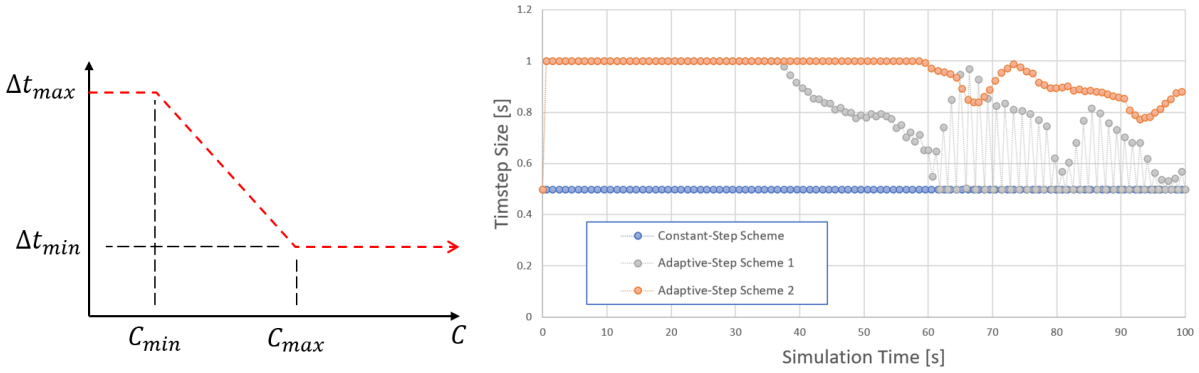

Figure 13: Linear mapping between Courant number and timestep Δt used in our third solution scheme, with saturation applied above and below $C = 5$ and $C = 1$ (left). For our simulation we chose $\Delta t_{min} = 0.5s$ and $\Delta t_{max} = 2.5s$. The choice of these adjustable parameters affects the mapping between C and δt, and therefore the timesteps taken throughout the simulation. At right, we see the effects of changing the adaptive stepping parameters by plotting δt over the first 200s of our simulation. Adaptive-Step Schemes 1 and 2 are two example from the handful that we tried, and differ in the choices of $C_{min} = 1$ and $C_{max} = 5$ only. The settings for Adaptive-Step Scheme 2 (orange) produced a converged result in the shortest amount of time.

We used a very simple method for adjusting the timestep - a linear mapping between C and δt which is saturated to minimum and maximum δt, at user-defined minimum and maximum values of C. We did not conduct an exhaustive study of the ideal values for these parameters, and for our final benchmarking test we selected $C_{min} = 1$, $C_{max} = 5$, $\Delta t_{min} = 0.25s$, and $\Delta t_{max} = 2.5s$. These choices kept C bounded between 2.0 and 4.0 for most of the simulation - compare this to the highly variable C results from a constant-δt solve presented in Figure 15.

One interesting result from experimenting with the adaptive-timestepping monolithic solver is that the solutions obtained are subtly influenced by the timestep size.

4.4 Results of Benchmarking Tests

The results of our testing were surprising, in that our monolithic solvers vastly outperformed the operator-splitting scheme. We found that even with every step of the operator split utilizing Backward Euler time integration, we were still required to take prohibitively small timesteps - roughly 1% the size that could be taken with our monolithic solver. We believe that our results are at least partially reflective of the mesh size of our model problem ($5e4$ DoF), which is relatively small. It's quite likely that for problems with $1e6$ or more DoF, operator splitting techniques would surpass monolithic solves in terms of wall clock time required to simulate one second of simulation time.

We also observed some benefit associated with adaptive timestepping - we were able to reduce the wall clock time required to solve our problem by a factor of about 1.5 when employing an adaptive timestep. We also acknowledge that our implementation of adaptive timestepping is extremely

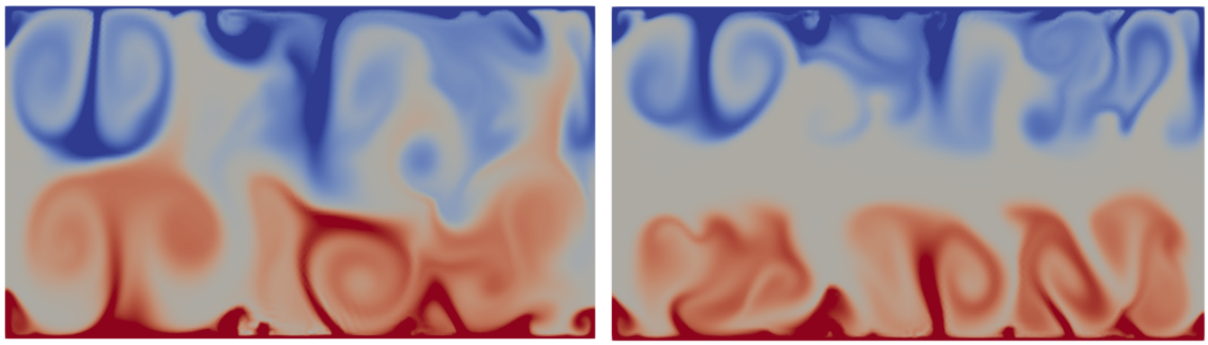

Figure 14: Temperature contour solutions for two adaptive-timestep solves at $t = 60s$, with identical boundary conditions. The results are subtly different. We believe that because the flow profiles we are simulating are so unsteady and unstable, small differences in velocity solutions computed at the initial timestep could propagate forward to produce large differences in solutions many timesteps later.

simplistic and prone to unexpected artifacts like oscillation in δt (see Figure 13, right). There is likely a much more robust method for adjusting the timestep, perhaps based on PID control.

	Monolithic	Monolithic Adaptive	Operator Split	
Max. Stable Δt	0.5	0.25 <Δt <2.5	0.005	
Step 1	7.86 ± 0.24	7.55 ± 0.82	0.071 ± 0.00	Full TH
Step 2	N/A	N/A	1.048 ± 0.03	Adv. NS
Step 3	N/A	N/A	0.071± 0.00	Pressure NS
Step 4	N/A	N/A	1.012 ± 0.01	Diff. NS
Total Sim. Time	3140	2244	89108	
Clock Time per Sim. Time	**15.7**	**11.2**	**445.4**	

Figure 15: Results from benchmarking test for three implementations of our unsteady natural convection problems. Figure of merit for performance is Wall Clock Time per Sim. Time - in other words, the number of seconds of actual time required to simulate one second of unsteady flow. Mesh and boundary conditions were identical for all three implementation. Our adaptive-timestepping monolithic solve outperformed the constant-step monolithic solve by a factor of 1.5, and outperformed the operator-split solve by a factor of 40.

5 Conclusion

This project gave us an opportunity to implement a two-way coupled multiphysics problem in FeniCS and compare our results to published data and our fluid dynamics intuition. We were initially drawn to this topic because of its practical application to heat transfer analysis, but we were surprised to find a huge amount of complexity related to the numerical implementation of the solution. Along the way, we compared our results to published data and found excellent (Figures 2, 4) or reasonable (Figures 5, 8) agreement. We look forward to taking the skills we developed in this class in sharpened during this project to our respective areas of research.

6 References

References

[1] Aytekin Çibik and Songül Kaya. "A projection-based stabilized finite element method for steady-state natural convection problem". In: *Journal of Mathematical Analysis and Applications* 381.2 (2011), pp. 469–484. ISSN: 10960813. DOI: 10.1016/j.jmaa.2011.02.020. URL: http://dx.doi.org/10.1016/j.jmaa.2011.02.020.

[2] Abhijit Guha, Akshat Jain, and Kaustav Pradhan. "Computation and physical explanation of the thermo-fluid-dynamics of natural convection around heated inclined plates with inclination varying from horizontal to vertical". In: *International Journal of Heat and Mass Transfer* 135 (2019), pp. 1130–1151. ISSN: 00179310. DOI: 10.1016/j.ijheatmasstransfer.2019.01.054. URL: https://doi.org/10.1016/j.ijheatmasstransfer.2019.01.054.

[3] Abhijit Guha and Subhajit Nayek. "Thermo-fluid-dynamics of natural convection around a heated vertical plate with a critical assessment of the standard similarity theory". In: *Physics of Fluids* 29.10 (2017). ISSN: 10897666. DOI: 10.1063/1.4990279. URL: http://dx.doi.org/10.1063/1.4990279.

[4] Abhijit Guha and Sayantan Sengupta. "Effects of finiteness on the thermo-fluid-dynamics of natural convection above horizontal plates". In: *Physics of Fluids* 28.6 (2016). ISSN: 10897666. DOI: 10.1063/1.4953382. URL: http://dx.doi.org/10.1063/1.4953382.

[5] Antonio Huerta and Jean Donea. *Finite Element Methods for Flow Problems*. John Wiley & Sons, Ltd, 2005, pp. 318–330. ISBN: 9780470013823. DOI: doi:10.1002/0470013826.ch6.

[6] B R Rich. "An Investigation of Heat Transfer From an Inclined Flat Plate in Free Convection". In: *Trans. ASME* 75 (1953), pp. 489–499.

[7] S. P. Stepanov, M. V. Vasilyeva, and V. I. Vasilyev. "Numerical simulation of the convective heat transfer on high-performance computing systems". In: *AIP Conference Proceedings* 1773 (2016). ISSN: 15517616. DOI: 10.1063/1.4965015.

APPENDIX A: Steady Formulation - Monolithic Solve

Navier Stokes Equations

We begin with the strong form of the Navier Stokes Equations, incorporating the Boussinesq approximattion for density variation

$$\rho_0 \frac{Du}{Dt} = -\nabla p + \rho_0(1 - \alpha(T - T_0))g + \mu\nabla^2 u \qquad \nabla u = 0$$

Breaking up the material derivative and putting into the weak formulation

$$\int_\Omega w \cdot \left[\rho_0 \frac{\partial u}{\partial t} + \rho_0(u \cdot \nabla u) + \nabla p - \mu\nabla^2 u - \rho_0(1 - \alpha(T - T_0))g\right] d\Omega = 0$$

The third term can be expanded by integrating by parts

$$\int_\Omega w \cdot (\nabla p) d\Omega = \int_\Gamma (wp) \cdot n d\Gamma - \int_\Omega (\nabla \cdot w) p d\Omega$$

The fourth term can be expanded by integrating by parts

$$\int_\Omega w \cdot (\mu\nabla^2 u) d\Omega = \int_\Gamma \mu(w\nabla u) \cdot n d\Gamma - \int_\Omega \mu\nabla w : \nabla u d\Omega$$

Collecting all the expressions, and dropping the boundary terms (zero Nuemann condition)

$$\int_\Omega w \cdot \rho_0 \frac{\partial u}{\partial t} d\Omega + \int_\Omega w \cdot \rho_0(u \cdot \nabla u) d\Omega + \int_\Omega w \cdot (\nabla \cdot w) p d\Omega + \int_\Omega w \cdot \mu\nabla w : \nabla u d\Omega - \int_\Omega w \cdot \rho_0(1 - \alpha(T - T_0))g] d\Omega = 0 \quad (1)$$

Conservation of Thermal Energy Equation

The thermal energy equation can be stated as:

$$\frac{DT}{Dt} = \kappa\nabla^2 T$$

Breaking up the material derivative, and putting into weak formulation:

$$\int_\Omega r \cdot (\frac{\partial T}{\partial t} + (u \cdot \nabla T) - \kappa\nabla^2 T) d\Omega = 0$$

The final term can be expanded to leave:

$$\int_\Omega r \cdot (\kappa\nabla^2 T) d\Omega = \int_\Omega r \cdot (\kappa(\nabla r \cdot \nabla T)) d\Omega + \int_\Gamma rT d\Gamma$$

Collecting terms and dropping the boundary terms (zero Neumann condition):

$$\int_\Omega r \cdot \frac{\partial T}{\partial t} d\Omega + \int_\Omega r \cdot (u \cdot \nabla T) d\Omega - \int_\Omega r \cdot \kappa(\nabla r \cdot \nabla T) d\Omega = 0 \quad (2)$$

Listing 1: FeniCS Syntax for Steady Problem

```
# Weak Form of Navier-Stokes
NS=\
rho*fe.inner(v,fe.grad(u)*u)*fe.dx\
+mu*fe.inner(fe.grad(v), fe.grad(u))*fe.dx\
-p*fe.div(v)*fe.dx\
-q*fe.div(u)*fe.dx\
-rho*(1.0-alpha*t)*fe.inner(v,g)*fe.dx\

# Weak Form of Thermal Energy Conservation
TH =\
fe.inner(z,fe.inner(u,fe.grad(t)))*fe.dx\
-kappa*fe.inner(fe.grad(t), fe.grad(z))*fe.dx

# Compose weak form of coupled problem by adding PDEs
CoupledWeakForm = NS + TH

# Set up solution space and compute jacobian
dW = fe.TrialFunction(W)
dFdW = fe.derivative(CoupledWeakForm, W0, dW)

# Set up nonlinear problem and solver
problem = fe.NonlinearVariationalProblem(CoupledWeakForm, W0, bcSet, J↩
    = dFdW)
solver  = fe.NonlinearVariationalSolver(problem)

# execute nonlinear solve
solver.solve()
```

APPENDIX B: Unsteady Formulation - Monolithic Solve

If we combine the two PDEs by addition, we get the following:

$$\int_\Omega w \cdot \left[\frac{\partial u}{\partial t} + \mathcal{L}(u,p,T,t)\right] d\Omega + \int_\Omega r \cdot \left[\frac{\partial T}{\partial t} + \mathcal{M}(u,T,t)\right] d\Omega = 0$$

where

$$\mathcal{L} = (u \cdot \nabla u) + (\nabla \cdot w)\frac{p}{\rho_0} + \frac{\mu}{\rho_0}\nabla w : \nabla u - (1 - \alpha(T - T_0))g$$

and

$$\mathcal{M} = u \cdot \nabla T - \kappa(\nabla r \cdot \nabla T))$$

Using the θ-Galerkin method for approximating time derivatives:

$$\begin{aligned}\int_\Omega w \cdot \left[\frac{u(t_{n+1}) - u(t_n)}{\Delta t}\right] d\Omega + \theta \int_\Omega w \cdot \mathcal{L}(u_{n+1}, p_{n+1}, T_{n+1}) d\Omega + (1-\theta)\int_\Omega w \cdot \mathcal{L}(u_n, p_n, T_n) d\Omega \\ + \int_\Omega r \cdot \left[\frac{T(t_{n+1}) - T(t_n)}{\Delta t}\right] d\Omega + \theta \int_\Omega r \cdot \mathcal{M}(u_{n+1}, T_{n+1}) d\Omega + (1-\theta)\int_\Omega r \cdot \mathcal{M}(u_n, T_n) d\Omega = 0\end{aligned} \tag{3}$$

Collecting terms:

$$\begin{aligned}\frac{1}{\Delta t}\int_\Omega w \cdot [u(t_{n+1}) - u(t_n)] + r \cdot [T(t_{n+1}) - T(t_n)]\, d\Omega \\ + \theta \left[\int_\Omega w \cdot \mathcal{L}_{n+1} + r \cdot \mathcal{M}_{n+1}\right] d\Omega + (1-\theta)\left[\int_\Omega w \cdot \mathcal{L}_n + r \cdot \mathcal{M}_n\right] d\Omega = 0\end{aligned} \tag{4}$$

Listing 2: FeniCS Syntax for Unsteady Monolithic Solve

```
# Define weak form of Nav. Stokes at t=n+1
NS_1 =\
rho*fe.inner(v,fe.grad(u)*u)*fe.dx\
+mu*fe.inner(fe.grad(v), fe.grad(u))*fe.dx\
-p*fe.div(v)*fe.dx\
-q*fe.div(u)*fe.dx\
-rho*(1.0-alpha*t0)*fe.inner(v,g)*fe.dx\

# Define weak form of Nav. Stokes at t=n
NS_0 =\
rho*fe.inner(v,fe.grad(u0)*u0)*fe.dx\
+mu*fe.inner(fe.grad(v), fe.grad(u0))*fe.dx\
-p*fe.div(v)*fe.dx\
-q*fe.div(u0)*fe.dx\
-rho*(1.0-alpha*t0)*fe.inner(v,g)*fe.dx\

# Define weak form of thermal equation at t=n+1
TH_1 =\
fe.inner(z,fe.inner(u0,fe.grad(t)))*fe.dx\
-kappa*fe.inner(fe.grad(t), fe.grad(z))*fe.dx

# Define weak form of thermal equation at t=n+1
TH_0 =\
fe.inner(z,fe.inner(u0,fe.grad(t0)))*fe.dx\
-kappa*fe.inner(fe.grad(t0), fe.grad(z))*fe.dx

# combine weak forms of NS and TH at t_n and t_n+1
T_0 = NS_0-TH_0
T_1 = NS_1-TH_1

# Theta-Galerkin Form
F = (1.0/dtime) * (fe.inner((u-u0),v)- fe.inner((t-t0),z))*fe.dx+ ↩
    (1.0-theta)*T_0 + theta*T_1

#Set up solution space, jacobian, nonlinear problem
dW = fe.TrialFunction(W)
dFdW = fe.derivative(F, w, dW)
problem = fe.NonlinearVariationalProblem(F, w, bcSet, J = dFdW)
solver  = fe.NonlinearVariationalSolver(problem)

while time<endtime:
    solver.solve()
    (u,p,t) = w.split()
    w0.assign(w)
    time+=dt
```

APPENDIX C: Unsteady Formulation - Operator Split Solve

Step 1:

We choose to solve the full conservation of thermal energy equation in Step 1, as opposed to the two-step advecitve/diffusive split suggested by [5].

$$\int_\Omega r \cdot \frac{\partial T}{\partial t} d\Omega + \mathcal{L}_1 = 0 \tag{5}$$

where

$$\mathcal{L}_1 = \int_\Omega r \cdot (u \cdot \nabla T) d\Omega + \int_\Omega (\kappa \nabla r : \nabla T) d\Omega \tag{6}$$

We integrate forward in time using a Crank-Nicholson scheme (we also experimented with Backward Euler time integration - effects were marginal because Step 2 took $> 97\%$ of the solve time):

$$\int_\Omega r \cdot \frac{T^{(n+1)} - T^n}{\Delta t} d\Omega + \frac{1}{2}(\mathcal{L}_1^{(n+1)} + \mathcal{L}_1^n) = 0 \tag{7}$$

Step 2a:

We implement a Chorin-Temam projection to split the operators of the Navier Stokes Equations. In the first step, we collect the terms which are pressure-independent:

$$\rho_0 \int_\Omega w \cdot \left(\frac{\partial u}{\partial t} \right) d\Omega + \mathcal{L}_2 = 0 \tag{8}$$

where

$$\mathcal{L}_2 = \rho_0 \int_\Omega w \cdot (u^{(1)} \cdot \nabla u^{(2)}) d\Omega + \int_\Omega \mu \nabla w : \nabla u^{(2)} - \rho_0 \int_\Omega w \cdot (1 - \alpha(T - T_0)) g d\Omega \tag{9}$$

We integrate forward in time using a fully implicit formulation ($u^{(1)} = u^{(2)} = u^{(n+1)*}$)

$$\rho_0 \int_\Omega w \cdot \left(\frac{u^{(n+1)*} - u^n}{\Delta t} \right) d\Omega + \mathcal{L}_2^{(n+1)*}) = 0 \tag{10}$$

This equation can now be solved for $u^{(n+1)^*}$

Step 2b:

In this step we compute the pressure correction - that is, we solve for the pressure field requred to leave the final velocity field divergence free. We start with the pressure terms of the Navier Stokes Equations:

$$\rho_0 \frac{\partial u}{\partial t} + \nabla p = 0 \tag{11}$$

We can expand the time derivative on velocity, and apply the Incremental Pressure Correction Scheme by using the pressure from the previous timestep $p = p^{(n+1)} - p^n$

$$\rho_0 \left(\frac{u^{(n+1)} - u^{(n+1)^*}}{\Delta t} \right) + \nabla (p^{n+1} - p^n) = 0 \tag{12}$$

Since we know the velocity field at the end of the timestep $u^{(n+1)}$ to be divergence free, we can take the divergence of this entire equation, leaving a linear equation in pressure $p^{(n+1)}$

$$\nabla^2 p^{(n+1)} = \nabla^2 p^n - \frac{\rho_0}{\Delta t} u^{*(n+1)} \tag{13}$$

Applying the method of weighted residuals gives us the weak form:

$$\int_\Omega \nabla p^{(n+1)} : \nabla q d\Omega = \int_\Omega \nabla p^n : \nabla q d\Omega - \frac{\rho_0}{\Delta t} \int_\Omega u^{*(n+1)} \cdot q d\Omega \tag{14}$$

This equation can now be solved for $p^{(n+1)^*}$

Step 2c:

We finally compute the end-of-step velocity field by again collecting the pressure-dependent terms in the Navier-Stokes Equations. We can directly inherit Equation 12 below, (this time without needing IPCS) - we just use the end-of-step pressure field solved for in Step 2b.

$$\rho_0 \left(\frac{u^{(n+1)} - u^{(n+1)^*}}{\Delta t} \right) + \nabla (p^{n+1}) = 0 \tag{15}$$

Compute final velocity field based on pressure correction (with the zero Neumann BC):

$$\int_\Omega w \cdot \left(\frac{u^{(n+1)} - u^{*(n+1)}}{\Delta t} \right) - \int_\Omega w \nabla p^{(n+1)} d\Omega = 0 \tag{16}$$

rearranging to create the left and right hand side explicitly:

$$\int_\Omega w \cdot u^{(n+1)} d\Omega = \int_\Omega w \cdot u^{(n+1)^*} d\Omega - \Delta t \int_\Omega w \nabla p^{(n+1)} d\Omega \tag{17}$$

This equation can now be solved for $u^{(n+1)}$

Listing 3: FeniCS Syntax for Unsteady Operator-Split Solve

```
#Step 1: TH Equation Full Solve
TH  = (-1/dt)*fe.inner(t-t0,r)*fe.dx+fe.inner(r,fe.inner(u0,fe.grad(t))
    )*fe.dx-kappa*fe.inner(fe.grad(t), fe.grad(r))*fe.dx

#Step 2a: NS Convective-Diffusive Solve
F1 = (1/dt)*fe.inner(u - u0, v)*fe.dx\
+fe.inner(fe.grad(u0)*u, v)*fe.dx\
+nu*fe.inner(fe.grad(u), fe.grad(v))*fe.dx\
-fe.inner((1.0-beta)*t1*g, v)*fe.dx\
#+stab

#Step 2b: NS Pressure Term with IPCS
a2 = fe.inner(fe.grad(p), fe.grad(q))*fe.dx
b2 = -(1/dt)*fe.div(u1)*q*fe.dx + fe.inner(fe.grad(p0), fe.grad(q))*fe
    .dx

#Step 2c: NS Final Velcoity Solve
a3 = fe.inner(u, v)*fe.dx
b3 = fe.inner(u1, v)*fe.dx -dt*fe.inner(fe.grad(p1), v)*fe.dx

while time<endtime:
    fe.solve(fe.lhs(TH)==fe.rhs(TH),t1,bc_t)
    fe.solve(fe.lhs(F1)==fe.rhs(F1),u1,bc_u)
    fe.solve(a2==b2,p1,bc_p)
    fe.solve(a3==b3,u1,bc_u)
    u0.assign(u1)
    p0.assign(p1)
    t0.assign(t1)
    time+=dt
```