Chalmers University of Technology

Implementing Heat Transfer in pyCALC-LES

Course Project for LES and DES using in-house Python code

Author:
Anand Joseph Michael

Supervisor:
Lars Davidson

January 18, 2024

Contents

1 Introduction

The pyCALC-LES code is a versatile turbulence modelling code capable of carrying out DNS, RANS, LES and DES simulations[1]. Multiple turbulence models have been implemented taking into consideration the transport equations for k, ϵ and ω. The code is capable of carrying out simulations with curvilinear grids in the x-y plane. The grid configuration needs to be Cartesian in the z-direction but grid number and spacing can be varied. The base code of the program consists of a base file named `pyCALC-LES.py`. A file named `setup.py` chooses the turbulence model and its corresponding initial settings. Another file named `modify.py` is used to modify parameters as the program is running. The grids required for each simulation is usually generated using a file named `generate-channel-grid.py`. The original pyCALC-LES code provided as part of the course does not account for heat transfer. As part of this project, I have modified the code base file `pyCALC-LES.py` to account for heat transfer and resultant buoyancy effects. The first section of this report details the modifications made to the code base and the setup and modify files. This is followed by an example case consisting of vertical channel flow where buoyancy effects have been both included and excluded.

2 Modifications to pyCALC-LES

The main challenge in the project was to include the heat transfer equations as part of the CFD code. The heat transfer equations read as follows [2]:

$$\frac{\partial T}{\partial t} + \frac{\partial v_i T}{\partial x_i} = \alpha \frac{\partial^2 T}{\partial x_i \partial x_i} \tag{1}$$

Here α refers to the thermal diffusivity and is a property of the fluid under consideration. The Prandtl number(Pr) can be used to calculate the thermal diffusivity from the viscosity of the fluid using the relation $\alpha = \nu / Pr$.

While modelling turbulence often time-averaged equations are used in the turbulence model. When the temperature equation is time- averaged, it reads as follows[2].

$$\frac{\partial \overline{T}}{\partial t} + \frac{\partial \overline{v_i} \overline{T}}{\partial x_i} = \alpha \frac{\partial^2 \overline{T}}{\partial x_i \partial x_i} - \frac{\partial \overline{v_i' T'}}{\partial x_i} \tag{2}$$

Here, the unknown time-average of the fluctuating component can be approximated using the Boussinesq assumption to get

$$\overline{v_i'T'} = -\alpha_t \frac{\partial \overline{T}}{\partial x_i} \tag{3}$$

The turbulent thermal diffusivity α_t can be determined from the turbulent prandtl number using the relation $\alpha_t = \nu_t/Pr_t$ where ν_t is the turbulent viscosity.

The temperature gradients in the fluid flow cause changes in density which can lead to buoyancy effects in the flow depending on the direction of the gravity and other body forces acting on the body. The buoyancy effect when added to the averaged momentum equations read as follows[2]:

$$\frac{\partial \overline{v_i}}{\partial t} + \frac{\partial \overline{v_i}\overline{v_j}}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \overline{p}}{\partial x_i} + \nu \frac{\partial^2 \overline{v_i}}{\partial x_j \partial x_j} - \frac{\partial \overline{v_i'v_j'}}{\partial x_i} - \beta g_i(\overline{T} - T_{ref}) \tag{4}$$

To include the heat transfer equation 2, a set of changes were made to the `pyCALC-LES.py` file and it was renamed to `pyCALC-LES-heat.py`. The script file usually used to run the combine these files and execute the program was also changed to reflect this change in the name of the base code file. The `setup.py` and the `modify.py` files were also changed to account for the modified code while also including the buoyancy term into the momentum equation 4. The suggestions provided in the workshop section 24.5 in the pyCALC-LES manual were used as a starting point for this purpose[1]. The major changes made in each section have been highlighted below. The changes made in the code have also been commented as `#temp` for easy identification.

2.1 global

The global file contains the declaration of all the global variables used in the program. The file was modified to include few new variables that will be used for the temperature part of the code. The variables storing the type of temperature boundary conditions as well as their values was declared as follows:

```
t_bc_east, t_bc_east_type, t_bc_north, t_bc_north_type, t_bc_south,
t_bc_south_type, t_bc_west, t_bc_west_type, t_bc_low,t_bc_high,
t_bc_low_type, t_bc_high_type
```

The `scheme_t` variable was declared to specify the scheme used for the discretisation of the temperature equation while the `solver_t` variable was declared to specify the solver to be used. Finally, two variables `prand_visc` and `prand_temp_turb` were declared to record the fluids Pr number as well as the turbulent Prandtl number Pr_t that is used to determine α_t.

2.2 pyCALC-LES-heat.py

When it comes to the main code file, the easiest way to follow the code would be to start with the section commented as `#the execution of the code starts here#`. The code starts with loading the grid and then initializing all the variables. Here, the code has been modified to include the default boundary conditions for the temperature.

```
t_bc_west=np.zeros((nj,nk))
t_bc_east=np.zeros((nj,nk))
t_bc_south=np.zeros((ni,nk))
t_bc_north=np.zeros((ni,nk))
t_bc_low=np.zeros((ni,nj))
t_bc_high=np.zeros((ni,nj))

t_bc_west_type='d'
t_bc_east_type='d'
t_bc_south_type='d'
t_bc_north_type='d'
t_bc_low_type='d'
t_bc_high_type='d'
```

This is followed by the initialization of the field variables for the first time where a line is added to initialize the temperature field as well.

```
    t3d=np.ones((ni,nj,nk))*1e-20 #temp
```

Following this, the variables storing the turbulent mean values are initialized. Here, variables to store $\overline{T}$,$\overline{u_i'T'}$ and $\overline{T'^2}$ are initialized

```
    t3d_mean=np.ones((ni,nj,nk))*1e-20 #temp
    ut3d_mean=np.ones((ni,nj,nk))*1e-20 #temp
    vt3d_mean=np.ones((ni,nj,nk))*1e-20 #temp
    wt3d_mean=np.ones((ni,nj,nk))*1e-20 #temp
    tt3d_mean=np.ones((ni,nj,nk))*1e-20 #temp
```

The `setup_case()` function that specifies the particular simulation case conditions is run after this and changes the boundary conditions and initialises most of the global variables.

The next modification to the code is made in the call to the `modify_init()` function which is used to modify the initial flow field variables. The function now has an extra argument named `t3d` and also returns an extra variable also called `t3d`.

Similarly, the functions `read_restart_data()`, `modify_inlet()`, `time_stats()`, `save_vtk()`, `save_time_aver_data()`, `save_data()`, `update()` as wells as the parts of the code calling them were modified to include the corresponding temperature terms in addition to the existing velocity terms.

The residual for the temperature calculations namely `residual_t` is added as part of the other set of residuals being initialised and it is set to zero.

The next modification is the addition of the code snippet to calculate the temperature field from the heat transfer equations. The major changes are in the `coeff()` and the `bc()` functions that are used to calculate the coefficients and source terms used in the calculation of the temperature field.

The `coeff()` function was changed to include an extra argument in the form of `prand_temp` which was used to divide the viscosity to obtain the thermal diffusivity α to be used to calculate the coefficients in the temperature equations. Any place where viscosity(ν) was previously used was modified to use this newly calculated α which in the code has been called `visc`.

```
visc=viscos/prand_temp #temp

visw[0:-1,:,:]=fx*vis_turb+(1-fx)*np.roll(vis_turb,1,axis=0)+visc
viss[:,0:-1,:]=fy*vis_turb+(1-fy)*np.roll(vis_turb,1,axis=1)+visc
visl[:,:,0:-1]=fz*vis_turb+(1-fz)*np.roll(vis_turb,1,axis=2)+visc

if cyclic_x:
   visw[0,:,:]=0.5*(vis_turb[0,:,:]+vis_turb[-1,:,:])+visc
   diffw[0,:,:]=visw[0,:,:]*areaw[0,:,:]**2/(0.5*(vol[0,:,:]+vol[-1,:,:]))
if cyclic_z:
   visl[:,:,0]=0.5*(vis_turb[:,:,0]+vis_turb[:,:,-1])+visc
   diffl[:,:,0]=visl[:,:,0]*areal[:,:,0]**2/(0.5*(vol[:,:,0]+vol[:,:,-1]))
```

Also passing `prand_temp_turb` as the argument for the `prand_1` and `prand_2` in the `coeff()` function automatically takes care of the calculation of the turbulent thermal diffusivity α_t. Thus the final call to the `coeff()` function while calculating the temperature field becomes

```
aw3d,ae3d,as3d,an3d,al3d,ah3d,apo3d,su3d,sp3d=coeff(convw,convs,convl,\
vis3d,prand_temp_turb,prand_temp_turb,prand_visc,f1_sst,scheme_t) #temp
```

When `coeff()` is called to calculate the coefficients for the other flow fields the argument for `prand_temp` is passed as 1 so that the viscosity is used unmodified.

Similarly, the function `bc()` was also modified to include an extra argument `prand_temp` which is used to modify the viscosity to get the thermal diffusivity. Passing 1 to this argument retains the viscosity as an unmodified quantity. All calls to the functions `coeff()` and `bc()` are modified to reflect this modification in its argument structure. Appropriate values for `prand_temp`(`prand_visc` for temperature equation and 1 for all others) are used.

A new function called `calct()` was added which was then used to add or modify the source terms to the temperature equations. This function was written like the `calcu()` function including an inbuilt call to the `modify_t()` function which was defined in the `modify.py` file.

The remaining part of the code segment used to calculate the temperature field is the same as that used to calculate all other field variables with the only changes being that the field variable is `t3d`, the scheme part of the function calls uses `scheme_t` and the convergence limit is set to `convergence_limit_t` which is defined in the `setup.py` file. The solver used for the temperature field is mentioned as `solver_t` while the max iterations for the solver is passed as `nsweep_t`

The residual value calculated at the end of the iteration is normalised using the variable `resnorm_t` which is defined in the `setup.py` file.

The last part of the `pyCALC-LES_heat.py` file consists of calls to functions to save the flow fields and the corresponding averaged quantities based on conditions specified in the `setup.py` file. These functions and their calls, as mentioned previously, have been modified to include the temperature field and its relevant fluctuating components.

2.3 setup.py

The following variables have been added to the `setup.py` file as new quantities to be specified for each simulation case.

1. `scheme_t` and `solver_t` is used to specify the scheme used to discretise the heat transfer equations and the solver used to solve the equations respectively. The same set of schemes and solvers as used for the velocity equations are available in this case as well. The `nsweep_t` term specifies the maximum iterations in the solver.

2. `prand_temp_turb` and `prand_visc` which is used to define the turbulent and viscous prandtl number to be used for the temperature equations.

3. `beta`,`g` and `t_ref` which are used to calculate the bouyant term in the momentum equations.

4. The `convergence_limit_t` specifies the convergence criteria for the temperature field calculations.

5. `resnorm_t` which is the quantity used to normalise the residual of the temperature field. It is calculated as the temperature flux entering in the flow direction.

   ```
   resnorm_t=tin*uin*zmax*y2d[1,-1]
   ```

   The `tin` used in this equation is specified as the reference temperature field for the simulation.

6. The final addition in `setup.py` is the inclusion of boundary conditions and values for the temperature field that is different from the default case mentioned in the base code.

2.4 modify.py

The functions used to modify the flow field variables as well as the coefficients used to calculate them are specified in this file. Since these functions change depending on the simulation case being run they are separately mentioned. The functions that have been modified are listed below with brief description of the changes made.

1. `modify_init()` function was changed to include `t3d` as a return variable. In case the simulation requires changes yo the initialised temperature field, it can be added here.

2. `modify_inlet()` function is also changed to include `t_bc_west` since the flow usually enters from the boundary where $x = 0$. Alternate inlet values can be mentioned here in case the inlet is not from the x-direction. If a synthetic temperature fluctuation needs to be added to the simulation then it can be inserted here.

3. `modify_u()`,`modify_v` and `modify_w()` functions are changed to include buoyancy terms in their source term depending on the direction of the body force acting on the fluid based on equation 4. For example, in the vertical channel flow case tested here, the body force acts in the negative x-direction resulting in an addition to the u-velocity source term as follows

```
        su3d=su3d+vol+g*beta*(t3d-t_ref)*vol
```

4. `modify_t()` function is added to the file to make any changes to the source terms. The function can include the addition of inlet boundary conditions to the source terms in the following manner.

   ```
   visc=viscos/prand_visc
   su3d[0,:,:]= su3d[0,:,:]+np.maximum(convw[0,:,:],0)*t_bc_west
   sp3d[0,:,:]= sp3d[0,:,:]-np.maximum(convw[0,:,:],0)
   vist=(vis3d[0,:,:]-viscos)/prand_temp_turb
   su3d[0,:,:]=su3d[0,:,:]+vist*aw_bound*t_bc_west
   sp3d[0,:,:]=sp3d[0,:,:]-vist*aw_bound
   ```

   It can also include heat sources inside the domain for example

   ```
   ss=2 #volume source
   su3d[5:10,10:20,:]= su3d[5:10,10:20,:]+ss*vol[5:10,10:20,:]
   ```

3 Channel Flow including Heat Transfer at $Re_\tau = 150$

The code described in the previous section was used to simulate a flow through a channel at Reynolds number $Re_\tau = 150$. The simulation case was based on the setup described in paper [3]. It consists of a vertical channel with a temperature difference of one across the vertical walls as shown in figure 1. The gravitational force acts in the downward direction along the negative x-direction. In the reference paper, DNS simulations were carried out at two Grashof(Gr) numbers to study the effects of buoyancy on the flow in a vertical channel. The initial case consisted of a $Gr = 0$ which corresponds to purely forced convection with no buoyancy effects while the second case in the paper corresponds to a $Gr = 7.68 \cdot 10^6$ which is referred to in some parts of this report as the mixed convection case as it includes buoyancy effects. The averaged flow field quantities obtained from the DNS simulations have been displayed in figure 2.

For this project, RANS simulations were initially carried out for both scenarios followed by some DES simulations to see if the accuracy could be improved while using less number of cells than used in the DNS simulations mentioned in the paper.

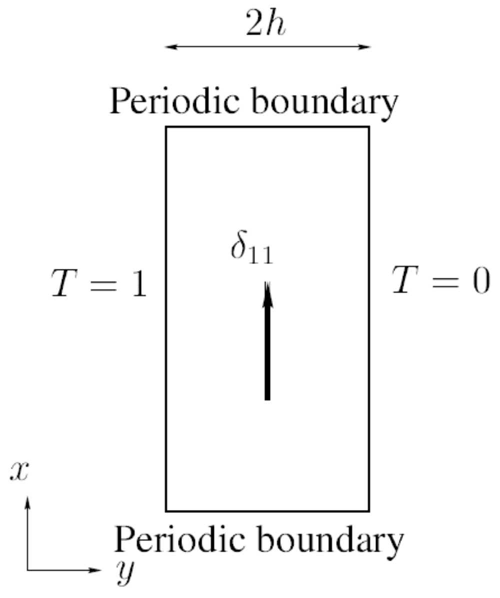
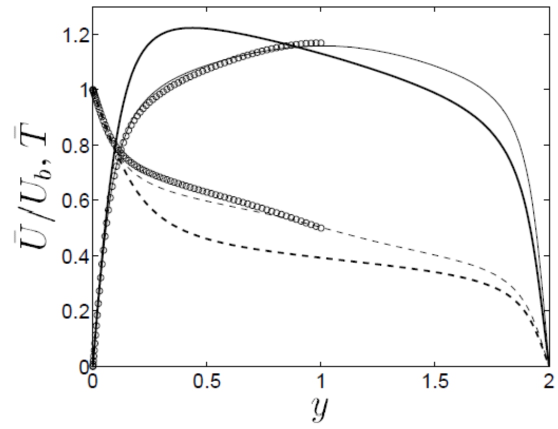


Figure 1: Setup for Vertical Channel Flow

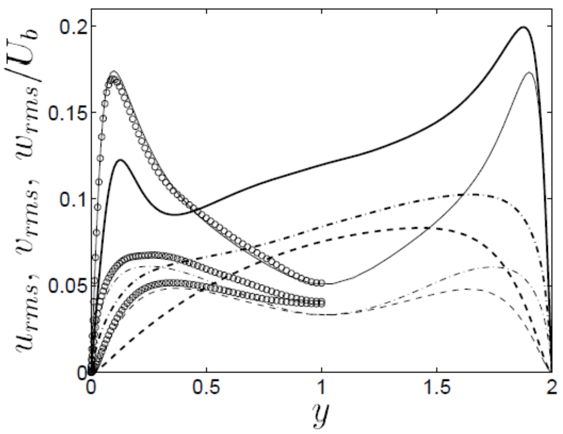
For both sets of simulations, the vertical wall at y=0 was set to a temperature of 1 while at the other side at y=2 it was set to 0. Since $Re_\tau = \frac{u_\tau h}{\nu}$, it was set to 150 by setting viscosity $\nu = 1/150$. The u_τ and the half-width of the channel h were both set to 1 to allow for this manner of controlling the Reynolds number. The Prandtl number were set to $Pr = 0.7$ considering air as the fluid in the simulations. The Pr_t value usually used varies between 0.7-0.9 and here a middle value was randomly selected setting $Pr_t = 0.85$. The Grshaof number $Gr = \frac{g\beta\Delta T H^3}{\nu^2}$ was set to 0 for the first case by setting g to 0. For the second case, $g = 7.5$ and $\beta = 1$ for a $Gr = 1.68{\cdot}10^5$. In both cases, the $\Delta T = T_{hot} - T_{cold} = 1$. The reference temperature was taken as the average of the hot surface and the cold surface and set to $T_{ref} = 0.5$. Trying to set the Gr number to higher values by modifying g resulted in flow fields with negative velocities in parts of the flow channel. So for the current project, a lower Gr number than that presented in paper [3] was used and a qualitative comparison has been provided for the mixed convection scenario. Possible sources of error are mentioned towards the end of the report.

The file `modify.py` was changed to include a constant pressure gradient that would drive the flow in the form of a source term in the u velocity equations(x-momentum equation). Apart from the buoyancy effect on the u-velocity has also been added as a source term. The change made is as follows:

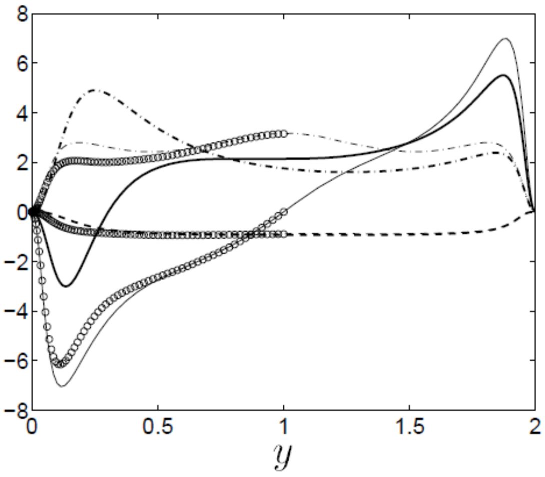
```
    su3d=su3d+vol+g*beta*(t3d-t_ref)*vol
```



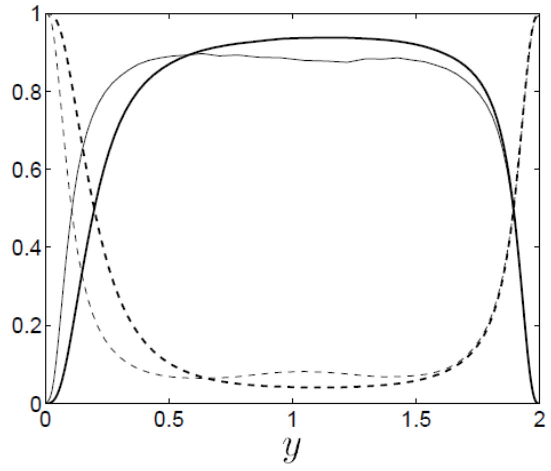
(a) Solid lines: $\overline{U}/U_b$, Dashed lines: T



(b) Solid lines: $\overline{u^2}$, dashed lines: $\overline{v^2}$, dash-dotted lines: $\overline{w^2}$



(c) Solid lines: $\overline{ut}/|q_w|$, dashed lines: $-\overline{vt}/|q_w|$, dash-dotted lines: $500\overline{t^2}$



(d) Solid lines: turbulent heat flux $\overline{vt}/q_w$, dashed lines: viscous heat flux $-1/(Re_\tau Prq_w)\partial\overline{T}/\partial y$, dash-dotted lines: $\overline{w^2}$

Figure 2: Lines represent results from DNS simulations carried out in paper [3] while the circles show DNS results from paper[4]. Thin lines :Gr=0, Thick lines: $Gr = 7.68 \cdot 10^6$

The pressure gradient is set to 1 and multiplied with the volume represented by `vol`. This will result in a Re_τ of $1/\nu$ in the absence of buoyancy, which in this case is 150. The buoyancy term is similarly added as `g*beta*(t3d-t_ref)*vol`.

3.1 RANS simulations

The $k-\omega$ model was used to run RANS simulations for the vertical channel. The absolute length of the simulation domain in the x, y and z directions were 1, 2 and 1.6 respectively. The grid sizing was set to 3x80x3 with periodic boundary conditions in both the x and z directions.

The simulation was run for a total of 2000 timesteps with the last 100 timesteps being used to obtain the averaged field values. The turbulent kinetic energy(k) and temperature(T) field values at different heights of the channel were recorded at each time interval to check for convergence of the solution to a steady value. In the forced convection case($Gr = 0$) the solution was observed to converge in around 250 timesteps but the simulation was allowed to run till 2000 timesteps. In the case of mixed convection involving buoyancy effects it was observed that restarting the simulation using the values obtained from the forced convection case resulted in smoother velocity and temperature profiles in the converged case. If the simulation was initialised with a zero velocity and temperature field while including buoyancy effects(especially at higher g values) then the velocity and temperature profiles across the channel showed sharper inflexions. Alternatively, if the Gr number was gradually increased from zero the profiles were smoother. The RANS results shown in figure 3 are from such a gradual increase in Gr.

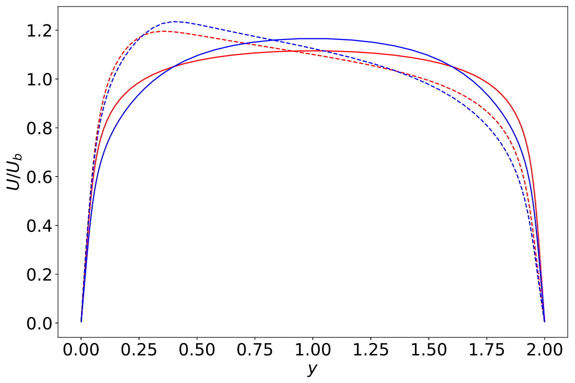
3.2 DES simulations

The $k-\omega$ DES model was used to run the same cases as previously. In this case the simulation domain was extended in the x direction to π. The grid was remade to have 32x80x32 cells. Similar to the previous case periodic boundary setting was used in the x and z direction.

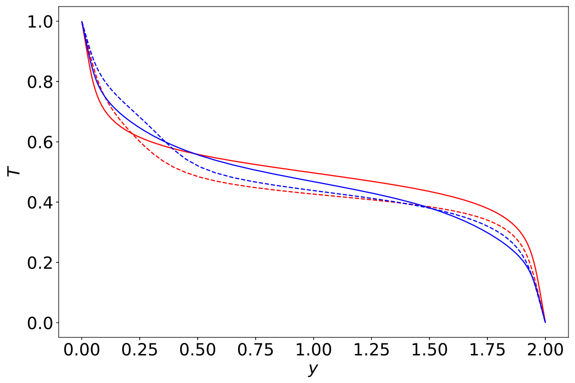
The simulation was run for a total of 20000 timesteps in the case of the purely forced convection case. The results for time averaging were recorded from timestep 17000. In the case of mixed convection, similar to the RANS cases, the forced convection end field variables were used to start a simulation and run for 5000 timesteps. Each of the DES simulations took considerably longer time to complete both from the increase in grid resolution and the number of timesteps. Previous, trials with the code had shown that initialising the simulation with zero for the field variables for the mixed convection case took much longer than for the purely forced convection case. Apart from this, as mentioned in the RANS section, the simulation results gave sharper inflexions in the averaged velocity and temperature fields. So for the set of results shown here, the mixed convection simulations were initialised with the results of the forced convection case.

3.3 Simulation Results

The averaged velocity and temperature profiles have been shown from the RANS as well as DES simulations for both the forced convection case($Gr = 0$) and the mixed convection case($Gr = 1.68 \cdot 10^5$) in figure 3. An exact comparison cannot be done with the DNS results for the mixed convection case since the Gr number is different. However, a qualitative comparison has been done here to show that the results show similar trends.



(a) $\overline{U}/U_b$



(b) $\overline{T}$

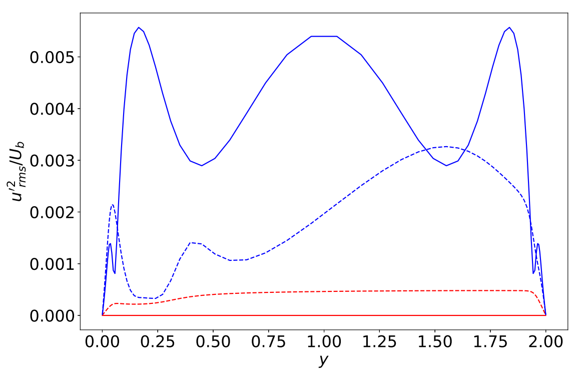
Figure 3: Averaged velocity and temperature profile from RANS and DES simulations. —:RANS($Gr = 0$),- - -: RANS($Gr = 1.68 \cdot 10^5$,—:DES($Gr = 0$),- - -: DES($Gr = 1.68 \cdot 10^5$)

The average velocity can be seen to be symmetric as expected in the forced convection case. The values are also quite close to those obtained from the DNS simulations with the peak of the curve between 1 and 1.2. But when it comes to the mixed convection case, it can be seen that the velocity is higher near the hot wall and lower near the cold wall as expected from the DNS results. The buoyancy term acts to assist the flow near the hot wall while it hinders it near the cold one. The DES simulation produced a more curved velocity profile than the RANS simulations but the general trends are similar in both cases. The peak in the DES mixed convection case has shifted further away from the wall compared to its RANS counterpart.

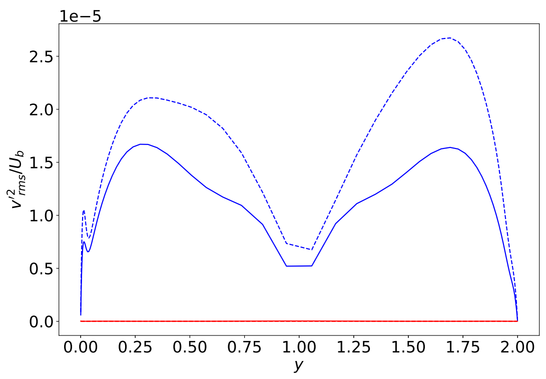
The temperature profile in the purely forced scenario can be seen to be quite similar with the first curve in the graph close to 0.6 and the second close to 0.4. The temperature profile can be seen to shift down in the middle section with the introduction of the buoyancy effect into the simulations as expected from the DNS simulations. However, the mixed convection temperature profile is not completely below the forced convection case either in RANS or DES as predicted by the DNS simulations. The point where the temperature goes below in the mixed convection case seems to match the peak in the velocity profiles in these cases. This makes sense as the high-speed fluid flow near the hot wall will have less time to transfer heat to the flow's inner regions, leaving them at lower temperatures.

The averaged values of the Reynolds Stress and fluctuating terms involving the temperature (turbulent heat fluxes) can be seen in figure 4. When it comes to the Reynolds stresses, there does not seem to be much agreement with the results shown from the DNS simulations. The RANS simulations record only very small values for the Reynolds Stresses in all directions. The DES simulations record a more significant amount of Reynolds Stresses. The $\overline{u'^2}$ and $\overline{v'^2}$ terms from the DES simulations have a similar shape to the DNS results but the magnitude of the terms are much smaller than that in the DNS simulations. The shift in these two Reynolds stresses with the higher peak towards the colder wall reflects the general trend predicted by DNS. The $\overline{w'^2}$ term does not seem to match either in shape or magnitude to the DNS results. This could possibly be due to the low number of cells as well as domain size in the z-direction.

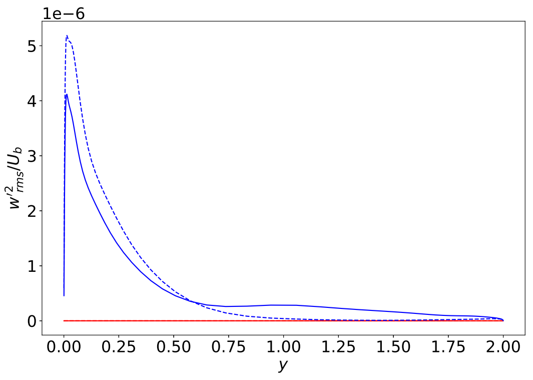
When it comes to the fluctuating terms involving the temperature (turbulent heat fluxes), the terms again are much smaller in magnitude than that seen in the DNS simulations. The RANS simulations again predict negligible turbulent heat fluxes while the DES simulations predict more noticeable quantities. In the DES forced and mixed convection case, the $\overline{u't'}$ term goes negative near the hot wall and positive near the cold wall but the graph is not symmetrical as expected from DNS. The $\overline{v't'}$ term does not show any similarity in the RANS or DES simulations. The only similarity between the $500\overline{t'^2}$ graph from DES and DNS is that it has a higher peak near the hot wall than at the cold wall in the mixed convection case.



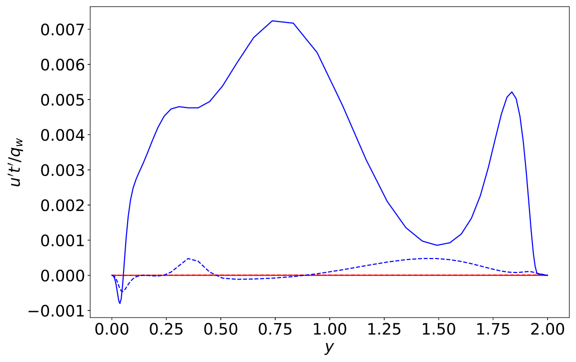
(a) $\overline{uu}_{rms}/U_b$



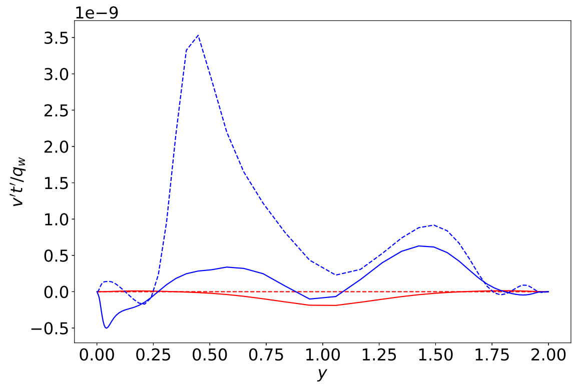
(b) $\overline{vv}_{rms}/U_b$



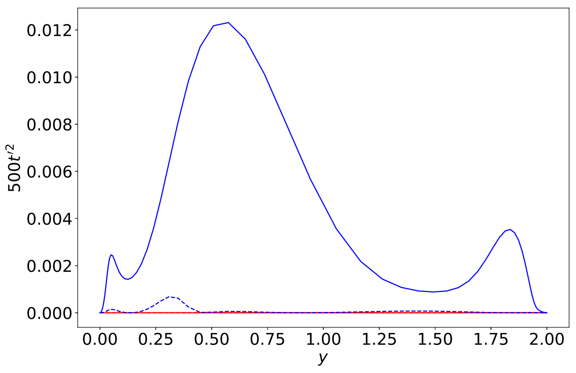
(c) $\overline{ww}_{rms}/U_b$



(d) $\overline{ut}/q_w$



(e) $\overline{vt}/q_w$



(f) $500\overline{tt}$

Figure 4: Averaged product of fluctuating quantities from RANS and DES simulations. —:RANS($Gr = 0$),- - -: RANS($Gr = 1.68 \cdot 10^5$,—:DES($Gr = 0$),- - -: DES($Gr = 1.68 \cdot 10^5$)

3.4 Discussions and Conclusions

The simulations seem to capture the overall trends when it comes to the average fluid field quantities such as the x-velocity and the temperature. The results from the forced convection case are quite close while the mixed convection case shows the same general trends. The DES simulation seems to be slightly better since it produces a more curved velocity profile.

When it comes to the fluctuating turbulent quantities, RANS does not produce any good results. DES seems to give slightly better results, especially in the case of $\overline{u'^2}$ and $\overline{v'^2}$. The shape of the graphs in this case is similar but the magnitudes are still off. Perhaps the modelling method used to predict these fluctuating quantities can be improved with other turbulence models such as wale. Due to a lack of time, it was not possible to try out the wale model or do direct DNS for these cases. It is also possible that a finer refinement of the grid and a longer domain might improve the results since the original case in the paper [3] used a more longer and finely meshed grid for both the cases.

The current system used for this project did not have GPU configured for pyCALC-LES and hence the DES simulations each took a couple of hours. The RANS simulations on the other hand took only a couple of minutes at the most. If the user is only interested in the trends of the averaged quantities then a RANS simulation should suffice as it produces results that are quite close to a DES simulation with fractional computational cost.

The Gr number used for the mixed convection case was lower than the one used in the paper [3] used for reference in this project. Higher Gr numbers were tried out by increasing the value of g but this resulted in parts of the average velocity profile going negative. For the purpose of the project a Gr number was chosen that produced similar average temperature and velocity profiles as that seen in paper [3]. An exact reason for this mismatch in Gr could not be identified but there could possibly be an error in the manner in which the number was calculated for this project. It is also possible that this could be an error in the implementation of the buoyant force, since the forced convection trends seem to match quite well with the DNS results. It is also possible that the Prandtl numbers used in the simulations may have influenced the temperature profile. Any suggestions, regarding this part of the code are welcome.

It was also interesting to note that the intial conditions especially, the field variable values influenced the final converged solution so strongly in the mixed convection case. To reduce large fluctuations in the simulation and allow for quick convergence the mixed convection cases were started from the end point of the forced convection cases. It was also noted that, instead of starting the DES simulations from zero, the flow profile generated from the RANS simulation could have been used to start the simulation thereby saving time for convergence to a solution.

References

[1] Lars Davidson. pyCALC-LES: A python code for DNS, LES and Hybrid LES-RANS, 2023.

[2] Lars Davidson. Fluid mechanics, turbulent flow and turbulence modeling, 2023.

[3] L. Davidson, D. Cuturic, and S.-H. Peng. DNS in a plane vertical channel with and without buoyancy, 2003.

[4] Nobuhide Kasagi and Oaki Iida. Progress in direct numerical simulation of turbulent heat transfer. In *Proceedings of the 5th ASME/JSME Joint Thermal Engineering Conference*, pages 15–19. American Society of Mechanical Engineers San Diego, 1999.