LF_30_tests

August 24, 2020

1 LF_30_tests

Fine tune mesh and simulation parameters for the evaporation of pure methane an $8m^3$ storage tank filled at 30% of it's capacity.

August 2020

1.0.1 Setting up the environment

[1]:
```
# System modules
import sys
import os
# 3rd party modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Set CRYODIR environmental variable to access cryo-foam functionality
%env CRYODIR=~/cryo-foam
# Add cryo-foam to path
sys.path.append('/home/felipe/cryo-foam')

# Custom cryo-foam modules
from analytical_solutions import create_small, cryogens_init, Tank
from postProcessing import concat_of_fobjs, plot_fobj
```

```
env: CRYODIR=~/cryo-foam
```

Case 1: `LF_30_vessel_noqvl_norelax`

No relaxation, no vapour to liquid heat ingress.

1.0.2 Analytical solution setup

[2]:
```
# Initialize tank
# Define tank properties
d_i = 1.604/4
d_o = 1.630/4
V = 8/64
LF = 0.30

TANK=Tank(d_i, d_o, V, LF)
TANK.set_HeatTransProps(U_L=0.019, U_V=0.019, Q_roof=0, T_air=288.15)

# Set cryogen
TANK.cryogen = cryogens_init.methane_init()
# Remove bottom heat ingress
TANK.Q_b_fixed = 0

# Evaporation rate in kg/h
evap_an = 3600 * TANK.b_l_dot
print("Analytical evaporation rate =  %.3f kg/h" % evap_an)
```

```
Analytical evaporation rate =  0.009 kg/h
```

Case input

[3]:
```
case_path = 'LF_30_vessel_noqvl_norelax'
```

1.0.3 Define functions

[4]:
```
TANK.V
```

[4]:
```
0.125
```

[5]:
```
def prepare_dmdt(case_path, tank, pool_length = 0.03):
    """ Create a dataframe concatenating separate postprocessing idmdt wdmdt␣
 ↪files"""

    # Calculate data frame with multiphase evaporation rate in kg/s
    fobj_name = 'idmdt_average'
    # Concatenate idmdt results for each simulation interval
    df = concat_of_fobjs(case_path, fobj_name)
    # Repeat the process for wall evaporation rate
    fobj_name = 'wdmdt_average'
    df_w = concat_of_fobjs(case_path, fobj_name)

    mesh_volume = tank.V*(1-tank.LF + pool_length)
```

```
    print("mesh volume = %.2f m^3" % mesh_volume)

    # Calculate evaporation rate
    dmdt_df = df.copy()
    dmdt_df.drop(columns='idmdt_average', inplace=True)
    # Multiply averages by mesh volume to reconstruct dmdt_tank
    dmdt_df['dmdt'] = 3600 * mesh_volume * (df['idmdt_average'].values +␣
↪df_w['wdmdt_average'].values)
    # Only subset the mass flow rates higher than zero: skip the condensation␣
↪phase
    dmdt_df = dmdt_df[dmdt_df['dmdt'] > 0]
    return dmdt_df

def plot_dmdt(dmdt_df, case_path, evap_an):
    plt.plot(dmdt_df['Time'], dmdt_df['dmdt'], label='Multiphase CFD')
    plt.hlines(evap_an, dmdt_df['Time'].iloc[0], dmdt_df['Time'].iloc[-1],␣
↪linestyle='--', label='Analytical')
    plt.xlabel('Time / s', size= 12)
    plt.ylabel('Evaporation rate / kgh$^{-1}$', size = 12)
    # Increase tick size
    plt.xticks(size=12)
    plt.yticks(size=12)
    # Add a nice title
    plt.title(case_path, size=13)
    plt.legend()
    plt.show()

def dmdt_avg(df_dmdt):
    """ Calculates average dmdt in kg/h """
    int_evap = np.trapz(df_dmdt['dmdt'], x=df_dmdt['Time'])
    simtime = df_dmdt['Time'].iloc[-1] - df_dmdt['Time'].iloc[0]
    return int_evap/simtime
```

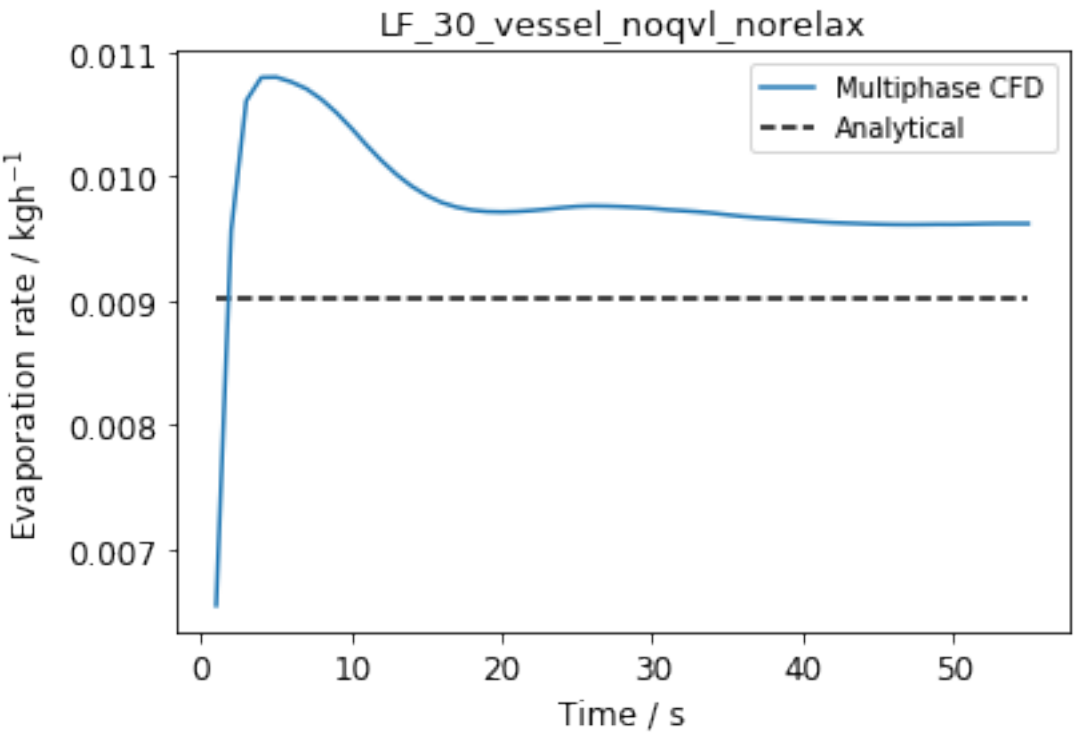
1.0.4 Generate dataframe

[6]:
```
dmdt_df = prepare_dmdt(case_path, TANK)
```

```
mesh volume = 0.09 m^3
```

1.0.5 Plot

[7]:
```
plot_dmdt(dmdt_df,case_path,evap_an)
```



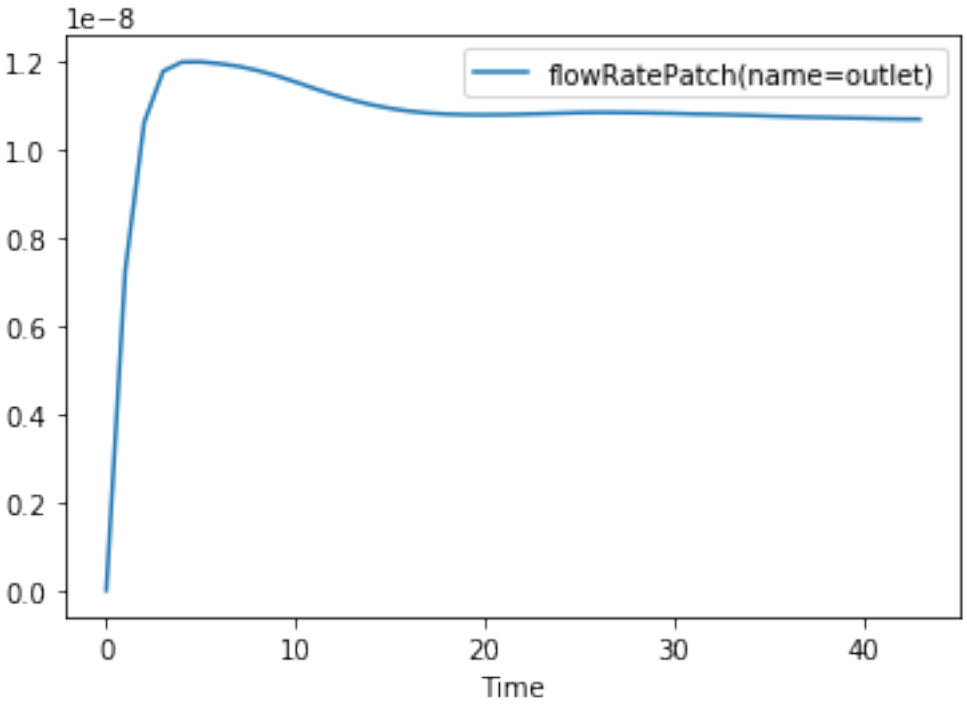
Where the bloody bias is coming from!

Plot bog

[8]:
```
outflow = pd.read_csv(case_path+'/'+case_path+'/flowRatePatch(name=outlet).csv')
```

[9]:
```
outflow.plot(x='Time',y='flowRatePatch(name=outlet)')
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4c60abea58>
```



1.0.6 Test 2: constant heat flux

Test 2.1: fixedMultiphaseHeatFlux This boundary condition produces instabilities in the interfacial temperature and it requires extremely small time steps. It may be one of the reasons why k-epsilon was used, to dissipate heat near the interface.

The simulation was aborted because it wasn't possible to run conveniently

Test 2.2: modify liquid wall boundary condition *A word of caution*: the saturation temperature is hard-coded in the last line of the boundary condition, on `operator==...`

[10]:
```
U_star = 0.3/0.03 * TANK.U_V * TANK.d_o/TANK.d_i
T_air = 288.15
T_L = 111.538
dT_dr = U_star*(T_air-T_L)
print("U_star = %.4f W m^-2 K^-1" % U_star)
print("dT/dr = %.4f K/m" % dT_dr)
```

```
U_star = 0.1931 W m^-2 K^-1
dT/dr = 34.1002 K/m
```

1.0.7 Test 3: no phase change wall function

In this test I aim to answer the dramatic question:

Has the wall boiling function spurious influence on the evaporation rate?

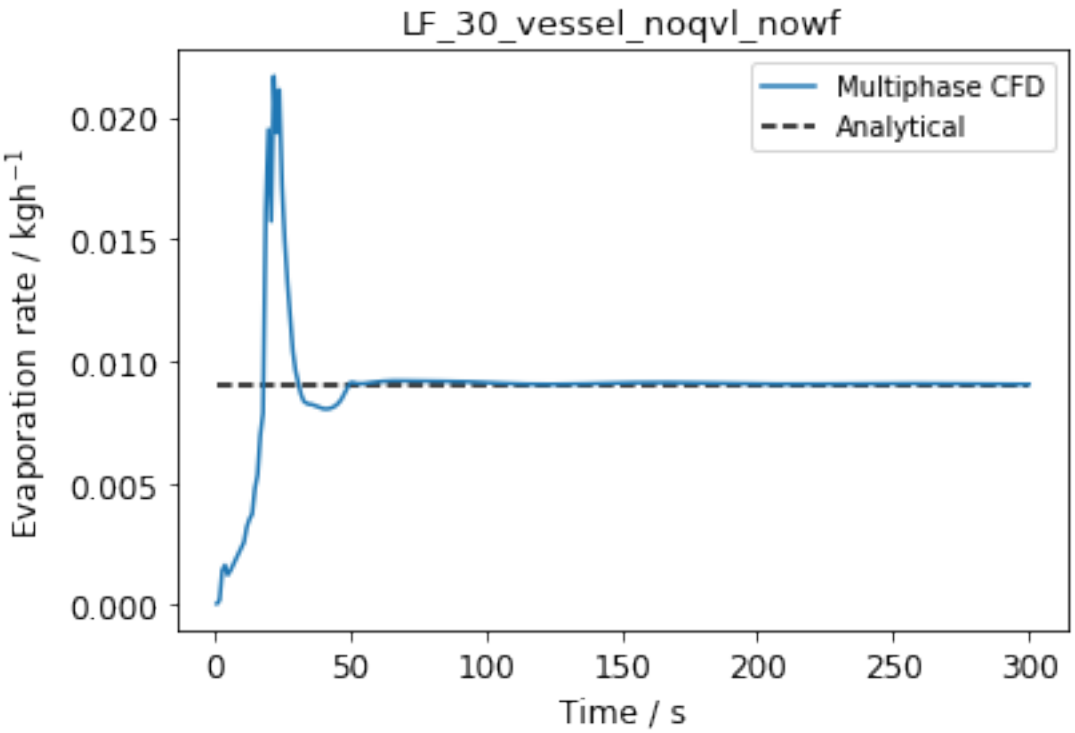
`0/alphat.liquid/wall` was modified to use

```
type            compressible::alphatPhaseChangeJayatillekeWallFunction;
Prt             0.85;
Cmu             0.09;
kappa           0.41;
E               9.8;
value           uniform 0;
```

My aim here is to transfer all the face change to *idmdt* interface evaporation and don't model the wall heat flux. Notably, this approach has stricter time-step requirements compared with Test 1.

[12]:
```
test3_path = 'LF_30_vessel_noqvl_nowf'
dmdt_3 = prepare_dmdt(test3_path,TANK)
plot_dmdt(dmdt_3,test3_path, evap_an)
print("dmdt_average = %.3e kg/h " % dmdt_avg(dmdt_3))
print("dmdt_analytical = %.3e kg/h" % evap_an)
last_err = (dmdt_3.iloc[-1,1] - evap_an)/evap_an * 100
print("latestError = %.3f%%" %last_err)
```

```
mesh volume = 0.09 m^3
```



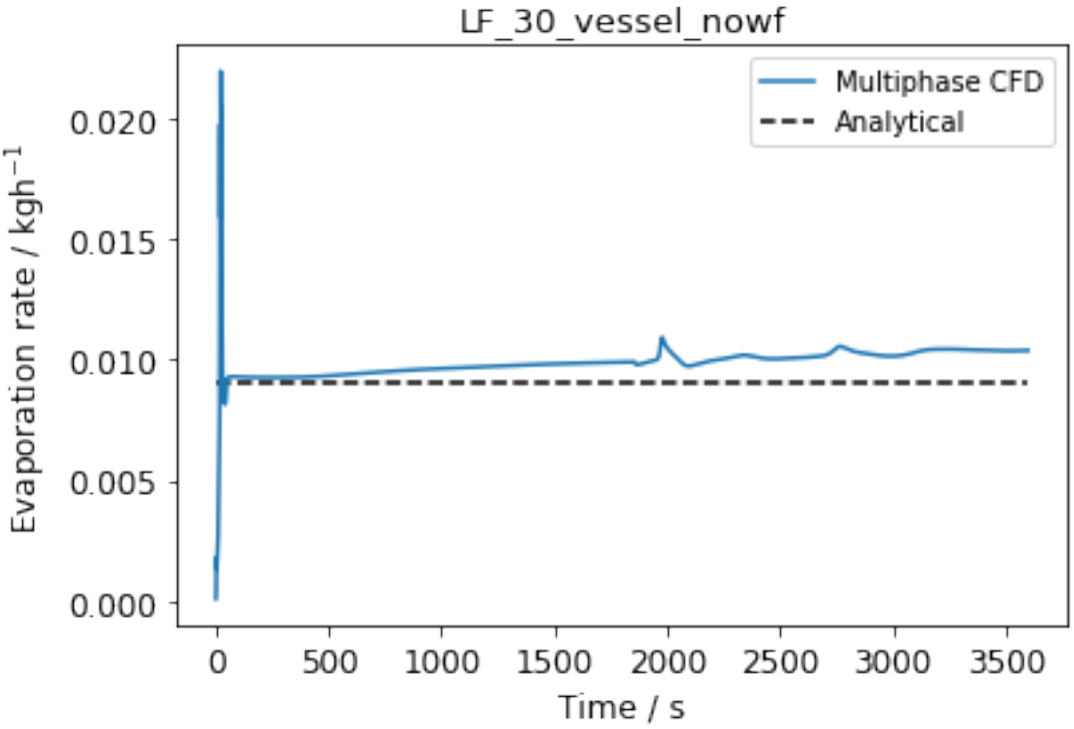
```
dmdt_average = 8.948e-03 kg/h
dmdt_analytical = 9.019e-03 kg/h
latestError = 0.193%
```

2 Test passed

2.0.1 3.2: Vapour heating on

[14]:
```
test32_path = 'LF_30_vessel_nowf'
dmdt_32 = prepare_dmdt(test32_path,TANK)
plot_dmdt(dmdt_32, test32_path, evap_an)
print("dmdt_average = %.3e kg/h " % dmdt_avg(dmdt_32))
print("dmdt_analytical = %.3e kg/h" % evap_an)
last_err = (dmdt_32.iloc[-1,1] - evap_an)/evap_an * 100
print("latestDeviation = %.3f%%" %last_err)
```

```
mesh volume = 0.09 m^3
```




```
dmdt_average = 9.857e-03 kg/h
dmdt_analytical = 9.019e-03 kg/h
```

latestDeviation = 14.987%

The change in seems to have produced an instability. However, from the simulation it is really difficult to observe any change on the distribution of the evaporation rate. Although after = t =1800s the evaporation rate seems more oscillatory, the positive deviation of this value with respect to the analytical solution is expected owing to the contribution of the vapour to liquid heat transfer rate to evaporation rates.

We can proceed to examine the BOG rates and vapour temperatures using cryo-foam and paraview.

2.0.2 BOG rates

[15]:
```
# Load vertical temperatures and boil-off gas rate
vessel_nowf_BOG = pd.read_csv(test32_path+'/'+test32_path+'/BOG.csv')
# Explore the dataframe
vessel_nowf_BOG.head(5)
```

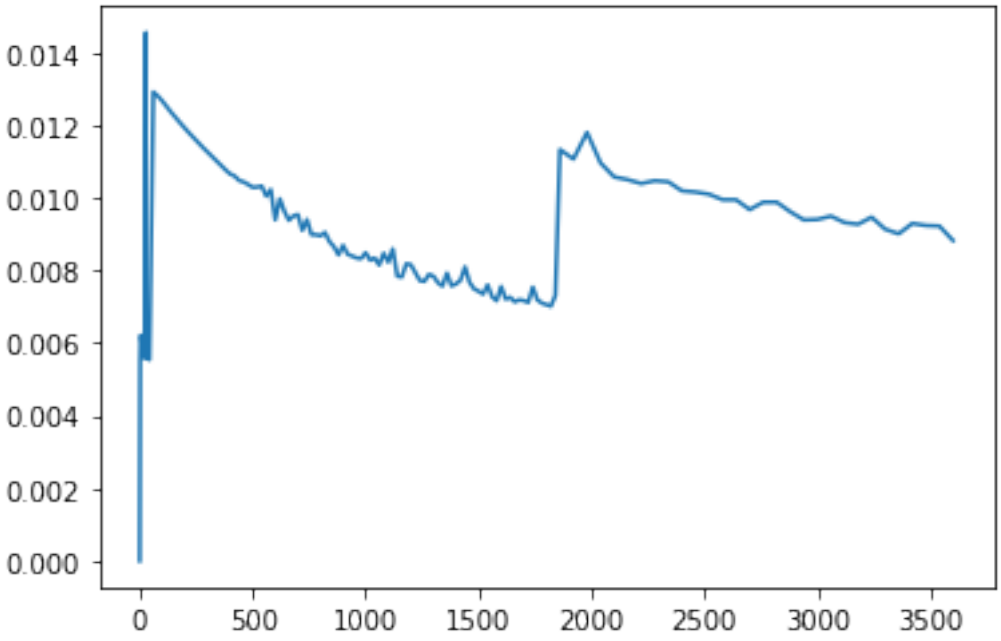
[15]:

	Unnamed: 0	Unnamed: 0.1	Time	outflow	Height	T_BOG	rho	\
0	0	0	0.0	0.000000e+00	0.686869	111.538	1.730005	
1	1	1	1.0	7.258291e-09	0.059400	111.538	1.730005	
2	2	2	2.0	1.061120e-08	0.059400	111.538	1.730005	
3	3	3	3.0	1.177155e-08	0.059400	111.538	1.730005	
4	4	4	4.0	1.198157e-08	0.059400	111.538	1.730005	

	vol_outflow	BOG
0	0.000000e+00	0.000000e+00
1	4.195532e-09	7.258291e-09
2	6.133623e-09	1.061120e-08
3	6.804344e-09	1.177155e-08
4	6.925744e-09	1.198157e-08

[16]:
```
plt.plot(vessel_nowf_BOG['Time'],vessel_nowf_BOG['BOG'] * 360/2.5 * 3600)
```

[16]: [<matplotlib.lines.Line2D at 0x7f4c6089f588>]



vessel_nowf_BOG.plot(x='Time',y='BOG')

Plot the temperature profile at t = 3600s

Define a function to plot an arbitrary temperature profile

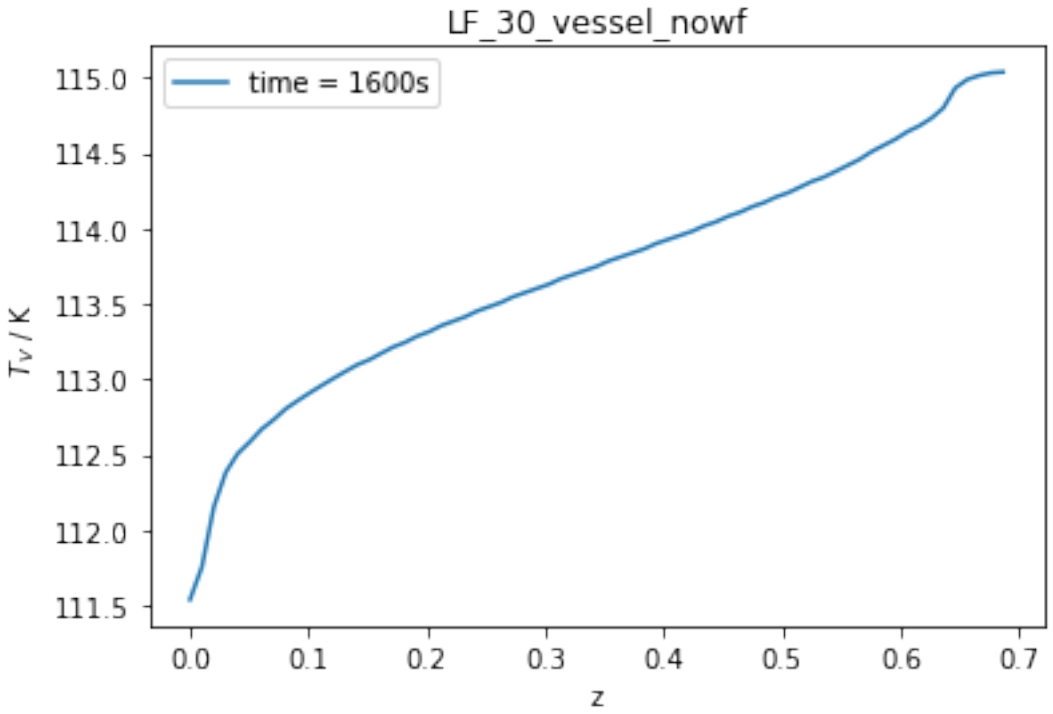
[17]:
```
# Define base directory
base_dir = os.getcwd()
```

Interactively go to a case directory and plot the vapour temperature at a specific time along a vertical line at a radius specified in the `singleGraph` file located in `system/singleGraph`

[18]:
```
def plot_sg(base_dir, case_path, time, sg_name='singleGraph'):
    # Change dir to test case
    os.chdir(case_path)
    # Run OpenFOAM postprocessing utility
    os.system('mpirun -np 8 reactingTwoPhaseEulerFoam -parallel -postProcess␣
↪-func '+ sg_name +' -time '+ str(time) +' > t.log')
    # Come back to the base dir
    os.chdir(base_dir)
    "Plot the results of singleGraph function object for a specific time"
    temp_gas = pd.read_csv(test32_path+'/postProcessing/'+sg_name+ '/' +␣
↪str(time)+'/line_T.gas.xy', header=None, sep='\t')
    temp_gas.columns = ['z', 'T']
    temp_gas.plot(x='z', y='T')
    plt.ylabel('$T_V$ / K')
    plt.title(case_path)
```

```
    plt.legend(["time = %.0fs" % time])
    plt.show()
    return temp_gas
```

[19]:
```
t_gas = plot_sg(base_dir, test32_path, 1600)
```



2.0.3 Read R_T and *height* from the mesh

blockMesh > mesh.log

[20]:
```
# Read tank radius and height from the mesh
# Change dir to test case
os.chdir(case_path)
# Run OpenFOAM postprocessing utility
os.system('blockMesh > mesh.log')
# Come back to the base dir
os.chdir(base_dir)
file = open(test32_path+'/mesh.log')
for i in file:
    try:
        if i.split()[1]=='domain':
            geo_list = i.split()
    except:
```

```
        pass
file.close()
# Define x_start and x_end of the bounding box
# (r0, z0, theta0) = (4,5,6)
x_start = geo_list[4:7]
x_end = geo_list[7:]
# Extract tank radius and vapour height
R_T = float(x_end[0].replace('(','')) - float(x_start[0].replace('(',''))
vapour_height = float(x_end[1])
print("R_T = %.3f m" % R_T)
print("vapour_height = %.3f m" % vapour_height)
```

```
R_T = 0.200 m
vapour_height = 0.693 m
```

[21]:
```
# Open singlegraph dict
sg_dict = open(test32_path+'/system/singleGraph', "r")
lines = sg_dict.readlines()
sg_dict.close()
```

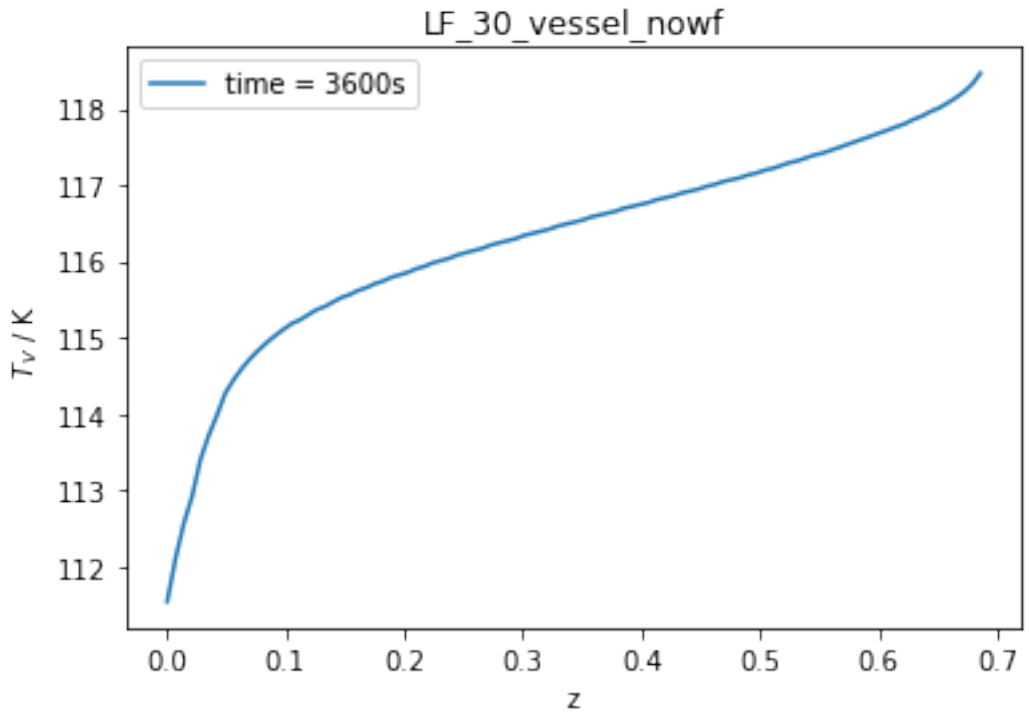
Write automatically

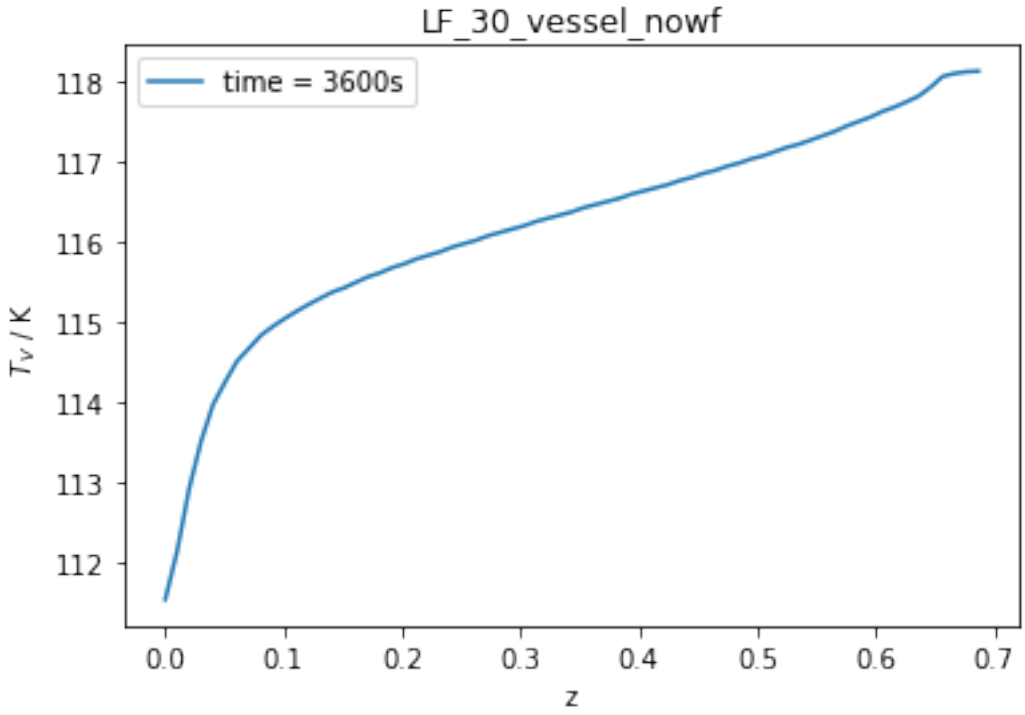
[22]:
```
def w_single_graph(case_path, radius_ratio, filename):
    # Open singlegraph dict
    sg_dict = open(case_path+'/system/singleGraph', "r")
    lines = sg_dict.readlines()
    sg_dict.close()

    # Use radius
    new_start = "    start   (%.4f" %(R_T*radius_ratio) + " " + "0 " + "0 );\n"
    new_end = "    end     (%.4f" %(R_T*radius_ratio) + " " + "%.4f" %␣
↪vapour_height + " 0);\n"
    # Write new_start and new_end in line array

    for idx, i in enumerate(lines):
        try:
            if i.split()[0] == 'start':
                lines[idx] = new_start
            elif i.split()[0] == 'end':
                lines[idx] = new_end
            elif i.split()[0] == 'singleGraph':
                # rename function object
                lines[idx] = filename+"\n"
        except:
            pass

    sg_dict = open(test32_path+'/system/'+filename, "w")
```

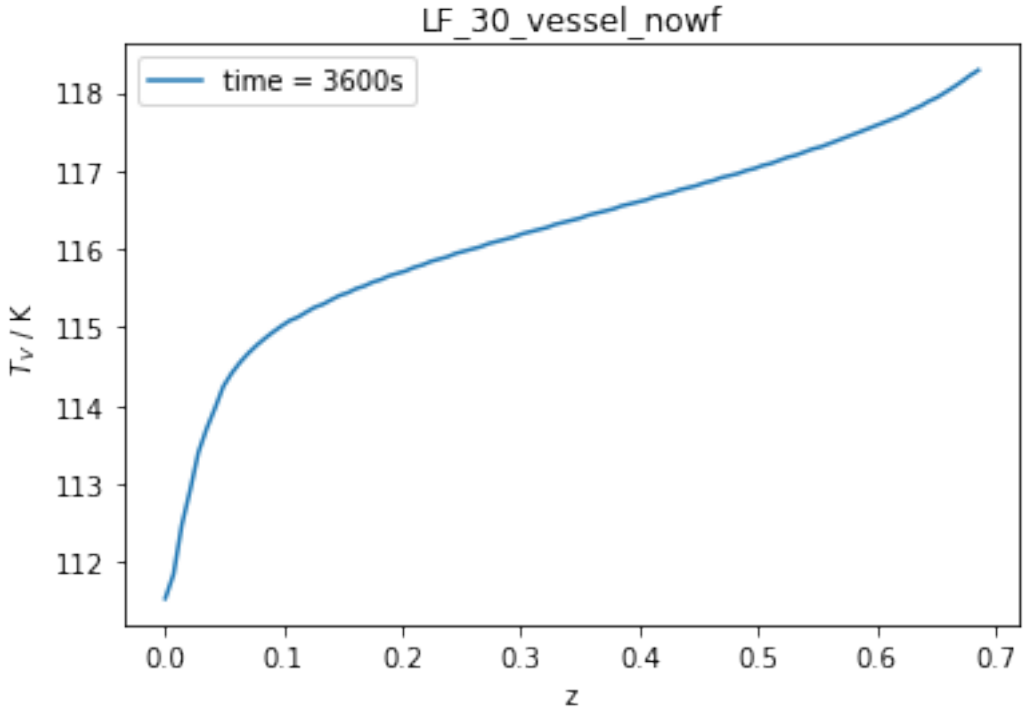
```
    for line in lines:
        sg_dict.write(line)
    sg_dict.close()
```

[23]:
```
w_single_graph(test32_path, 0.5, "singleGraph_half")
w_single_graph(test32_path, 0.99, "singleGraph_wall")
```

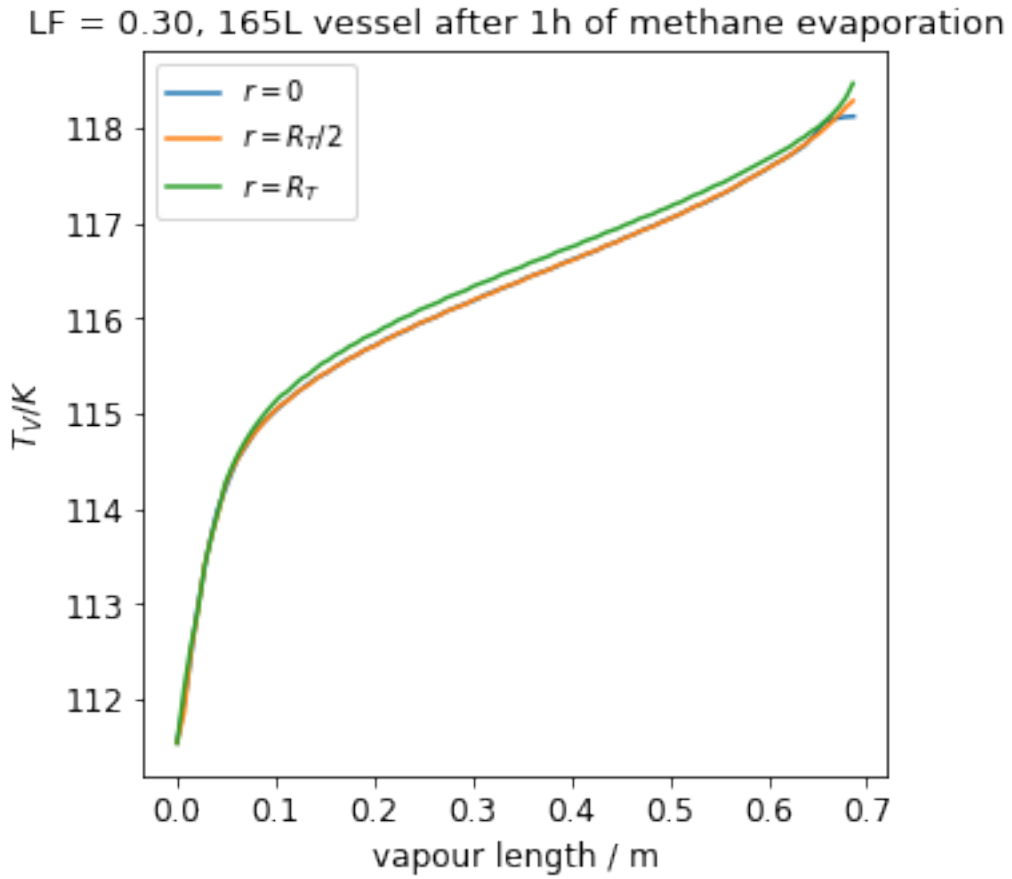
[24]:
```
temp_wall = plot_sg(base_dir, test32_path, 3600, sg_name='singleGraph_wall')
temp_valve = plot_sg(base_dir, test32_path, 3600, sg_name='singleGraph')
temp_half = plot_sg(base_dir, test32_path, 3600, sg_name='singleGraph_half')
```




LF_30_vessel_nowf
time = 3600s
118
117
116
115
114
113
112
T_v / K
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
z


LF_30_vessel_nowf
time = 3600s
118
117
116
115
114
113
112
T_v / K
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
z

[25]:
```
# Compare temperature profiles
z = temp_valve['z'].values
plt.figure(figsize=[5,5])
plt.plot(temp_valve['z'], temp_valve['T'].values, label = '$r=0$')
plt.plot(temp_half['z'], temp_half['T'].values, label = '$r=R_T/2$')
plt.plot(temp_wall['z'], temp_wall['T'].values, label = '$r=R_T$')
plt.xlabel('vapour length / m', size=12)
plt.ylabel('$T_V / K$', size=12)
plt.legend()
plt.title("LF = 0.30, 165L vessel after 1h of methane evaporation", size=13)
plt.xticks(size=12)
plt.yticks(size=12)
plt.show()
```



This visualization is extremely helpful to prove the predominant direction of heat transfer in the vapour phase of a cryogenic liquid stored in a tank under isobaric conditions.

2.0.4 Analysis

17-08-2020

At t = 1840s, the maxCo was increased to 0.75 to accelerate the simulation. This may produce instabilities, but given that this case is hypothetical, is a cost worth paying. If the simulation diverges, it is always easy to start from t=1840s.

2.0.5 Known potential sources of error when using a wall boiling function

- Inaccuracies on U_{corr} owing to discretization errors in the numerical integration of the vapour volume (+- 0.2%)
- Wedge to cylinder discretization (+- 0.05%)
- Integration errors on the calculation of *idmdt* and *wdmdt* (<1%)
- In this case, the mesh is rather coarse in the r direction.

The maximum accuracy of a bubble is determined by the mesh size. As the bubbles near the wall can be extremely small, a potential solution can be refining the mesh in the radial direction near the wall.

Wall boiling function problem For low heat fluxes, the boundary condition related to wall evaporation causes spurious results deviating from the analytical evaporation rate. This may be a consequence of a wrong choice of parameters. Turning off the boundary condition allows the program to achieve the correct evaporation rate. However, it comes with the price of increasing the unstability of the simulation.

2.0.6 Full scale tank

2.0.7 Analytical solution setup

[32]:
```
# Initialize tank
# Define tank properties
d_i = 1.604
d_o = 1.630
V = 8
LF = 0.30

FULL_TANK=Tank(d_i, d_o, V, LF)
FULL_TANK.set_HeatTransProps(U_L=0.019, U_V=0.019, Q_roof=0, T_air=288.15)

# Set cryogen
FULL_TANK.cryogen = cryogens_init.methane_init()
# Remove bottom heat ingress
FULL_TANK.Q_b_fixed = 0

# Evaporation rate in kg/h
```

```
evap_full = 3600 * FULL_TANK.b_l_dot
print("Analytical evaporation rate =  %.3f kg/h" % evap_full)
```

```
Analytical evaporation rate =  0.144 kg/h
```

[33]:
```
ftank_path = '/home/felipe/OpenFOAM/felipe-v1906/run/multiphase/
 ↪reactingTwoPhaseEulerFoam/base_cases/LF_30_nowf'
```

[34]:
```
dmdt_full = prepare_dmdt(ftank_path, FULL_TANK)
plot_dmdt(dmdt_full, ftank_path, evap_full)
```

```
mesh volume = 5.84 m^3
```