Python Code that Generates Analytical Solutions of the Linearized R13 Equations over an Annular Domain

August 15, 2020

Abstract

A Python code capable of generating analytical solutions of the linearized R13 equations over an annular domain is presented here. The generic solution for the field variables temperature θ, pressure p, velocity $\underline{u}$, heat flux $\underline{q}$, and stress tensor $\underline{\underline{\sigma}}$ over the mentioned domain are already available. However, these expressions contain integration constants that are to be determined based on the boundary conditions and source terms corresponding to a specific problem under consideration. The code allows for the user to specify the boundary conditions such as the temperature θ^w, pressure p^w, tangential velocity u_t^w, normal velocity u_n^w and inflow parameters $\left(\epsilon_p^w, \epsilon_v^w, \epsilon_s\right)$ along with the formulation parameters such as the inner and outer radius (R_0, R_1), Knudsen number Kn, accommodation coefficient $\tilde{\chi}$ and source terms and then proceeds to solve the boundary equations to obtain the values of the integration constants for the specified problem. Furthermore, this code then exports the generated solution in three different formats. The exported scripts are in the forms of a Python script, a C++ header file and a .cpp file compatible with the DOLFIN python module (popularly used to solve PDEs). These exported scripts can then be used for validation purposes or to perform convergence studies over a range of solvers.

Keywords : Linearized R13 Equations, Analytical Solution, Python, Sympy, C++, DOLFIN

Contents

1 Introduction **3**
1.1 Linearized R13 Equations . . . 3
1.2 Wall Boundary Conditions . . . 3
1.3 Annular Domain . . . 4

2 Code Description **5**
2.1 SymPy Module . . . 5
2.2 Code Structure . . . 6
2.3 Code Execution . . . 6
2.4 Using Exported Solution . . . 7
2.4.1 Python Format . . . 8
2.4.2 C++ Format . . . 8
2.4.3 DOLFIN Format . . . 9
2.5 Limitations . . . 10

A Analytical Expressions of the field variables **11**
A.1 Temperature . . . 11
A.2 Pressure . . . 11
A.3 Velocity . . . 11
A.4 Heat Flux . . . 12
A.5 Stress . . . 12

1 Introduction

1.1 Linearized R13 Equations

The R13 equations [3] is an equation set that predicts rarefied gas flows, with great accuracy, even at high Knudsen numbers ($Kn = 0.5$) which are well beyond the reach of the classical Navier-Stokes and Fourier laws (NSF). These equations are derived using a combination of moment methods in kinetic gas theory and a Chapman-Enskog-like series expansion approach. In addition to the three field variables (temperature θ, pressure p and velocity $\underline{u}$) which are solved for in classical gas dynamics, the R13 equations also solve for the field variables heat flux $\underline{q}$ and stress tensor $\underline{\underline{\sigma}}$.

A linearized steady-state variant of the full non-linear R13 equation set is considered throughout this report. This linearization is carried out by considering the unknown field variables as a first-order perturbation of an equilibrium ground state [7]. Physically this makes these equations quite appropriate for cases that involve slow flows (low Mach number). Additionally, the steady-state assumption leads to the exclusion of the temporal derivative terms. The resulting linear equation set consists of three balance laws (mass, energy and momentum), two evolution equations for the non-equilibrium quantities (heat flux $\underline{q}$ and stress tensor $\underline{\underline{\sigma}}$) and three closure equations for the highest-order moments $\underline{\underline{\underline{m}}}$, $\underline{\underline{R}}$ and R. The mass source, heat source and body force vector are represented by f_m, f_h and $\underline{f_b}$ respectively, in the equations below.

$$\nabla \cdot \underline{u} = f_m \tag{1}$$

$$\nabla \cdot \underline{q} + \nabla \cdot \underline{u} = f_h \tag{2}$$

$$\nabla p + \nabla \cdot \underline{\underline{\sigma}} = \underline{f_b} \tag{3}$$

$$\frac{5}{2}\nabla\theta + \nabla \cdot \underline{\underline{\sigma}} + \frac{1}{2}\nabla \cdot \underline{\underline{R}} + \frac{1}{6}\nabla R = -\frac{1}{Kn}\frac{2}{3}\underline{q} \tag{4}$$

$$\frac{4}{5}\left(\nabla \underline{q}\right)_{STF} + 2\left(\nabla \underline{u}\right)_{STF} + \nabla \cdot \underline{\underline{\underline{m}}} = -\frac{1}{Kn}\underline{\underline{\sigma}} \tag{5}$$

Closure for the highest-order moments $\underline{\underline{\underline{m}}}$, $\underline{\underline{R}}$ and R are provided by the following equations.

$$\underline{\underline{\underline{m}}} = -2Kn\left(\nabla \underline{\underline{\sigma}}\right)_{STF} \tag{6}$$

$$\underline{\underline{R}} = -\frac{24}{5}Kn\left(\nabla \underline{q}\right)_{STF} \tag{7}$$

$$R = -12Kn\left(\nabla \cdot \underline{q}\right) \tag{8}$$

where the deviatoric part of the symmetric tensor is denoted by $(\cdot)_{STF}$.

The above equations are all in the dimensionless form and the Knudsen number Kn describes the rarefaction situation of the gas being considered. A detailed derivation of the analytical solution, including details about the ansatz used, for the cases of slow rarefied gas flows past a cylinder and a sphere can be found in [5, 9].

1.2 Wall Boundary Conditions

In [6], boundary conditions for the R13 equations were given in accordance with the Maxwell accommodation model. These equations involve the accommodation coefficient χ which represents the fraction of the particles hitting the boundary that are absorbed by the wall. After appropriate linearization and simplification for a two-dimensional space considering a boundary-aligned coordinate system (n, t), these equations, which are non-linear in their fullest extent, look as shown below

$$\epsilon_p^w \left(u_n - u_n^w\right) = \epsilon_v^w \tilde{\chi} \left(\left(p - p^w\right) + \epsilon_s \sigma_{nn}\right) \tag{9}$$

$$\sigma_{nt} = \tilde{\chi} \left(\left(u_t - u_t^w\right) + \frac{1}{5} q_t + m_{nnt}\right) \tag{10}$$

$$R_{nt} = \tilde{\chi} \left(-\left(u_t - u_t^w\right) + \frac{11}{5} q_t - m_{nnt}\right) \tag{11}$$

$$q_n = \tilde{\chi} \left(2\left(\theta - \theta^w\right) + \frac{1}{2}\sigma_{nn} + \frac{2}{5}R_{nn} + \frac{2}{15}R\right) \tag{12}$$

$$m_{nnn} = \tilde{\chi} \left(-\frac{2}{5}\left(\theta - \theta^w\right) + \frac{7}{5}\sigma_{nn} - \frac{2}{25}R_{nn} - \frac{2}{75}R\right) \tag{13}$$

$$\left(\frac{1}{2}m_{nnn} + m_{ntt}\right) = \tilde{\chi} \left(\frac{1}{2}\sigma_{nn} + \sigma_{tt}\right) \tag{14}$$

where θ^w, u_n^w, u_t^w and p^wrepresent the temperature, normal velocity, tangential velocity and pressure at the wall. The parameters ϵ_p^w, ϵ_v^w and ϵ_s are used for the inflow modelling and the modified Maxwell accommodation factor is given by $\tilde{\chi} = \sqrt{\frac{2}{\pi\theta_0}}\frac{\chi}{2-\chi}$ [8].

It is important to observe how Eq. 9 is structured to understand the inflow model being used here. The parameters involved are the normal velocity prescription ϵ_v^w, the pressure prescription ϵ_p^w and the stress prescription ϵ_s. When $\epsilon_p^w >> \epsilon_v^w$ there is greater emphasis on the normal velocity boundary condition $u_n = u_n^w$ and when $\epsilon_v^w >> \epsilon_p^w$ there is greater emphasis on the pressure boundary condition $p^w = p + \epsilon_s \sigma_{nn}$. Clearly the stress prescription ϵ_s represents the emphasis of the stress term within the pressure boundary condition. As an example, a standard zero normal velocity boundary condition can be enabled by setting $\epsilon_v^w = 0$ and $u_n^w = 0$.

1.3 Annular Domain

The domain considered here is the annular region between two infinitely long coaxial cylinders of radii R_0 and R_1. An assumption of symmetry and homogeneity along the length of the cylinders allows for the extraction of a 2D model problem while retaining the 3D nature of the field variables such as the stress tensor $\underline{\underline{\sigma}}$ (refer to [4, 8] for a more detailed explanation). This model problem is represented by a 2D domain described by the area between two concentric circles of radii R_0 and R_1 as

$$\Omega = \left\{(x, y) \in \mathbb{R}^2 \mid R_0 \leq \sqrt{x^2 + y^2} \leq R_1\right\} \tag{15}$$

Sharp corners are avoided in this domain and the fact that the origin of the coaxial circles has been excluded from the domain allows for the prescription of normal fluxes without any issues. A representation of the domain is shown in Fig. 1. The values for each parameter on both the boundaries (Γ_0 and Γ_1 in Fig. 1) can be of the form $a + b\cos(\theta)$ or $a + b\sin(\theta)$ where $(a, b) \in \mathbb{R}$. This allows for a variety of cases to be specified a few of which are specified below.

- A rotating cylindrical shell could be modelled by setting the tangential velocity at the boundary to a constant value and simultaneously setting the normal velocity to zero. $(u_t = v_{rot}, u_n = 0)$
- The classical problem of the flow past a circular cylinder can be modelled by setting the tangential velocity, normal velocity and pressure at the inner boundary to zero $(u_t = 0, u_n = 0, p = 0)$ and assigning periodic functions for the the same at the outer boundary $(u_t = -v_0\sin(\theta), u_n = v_0\cos(\theta), p = -p_0\cos(\theta))$.

For annular domain problem considered here, analytical solutions do not exist for any arbitrary functions for the source terms f_m, f_h and $\underline{f_b}$. Only specific forms of the source functions allow for non-homogeneous analytical solutions. Therefore, general forms of source functions corresponding to each of f_m, f_h and $\underline{f_b}$, which allow for analytical solutions in this case, have been determined empirically and are shown in Eq 16-18

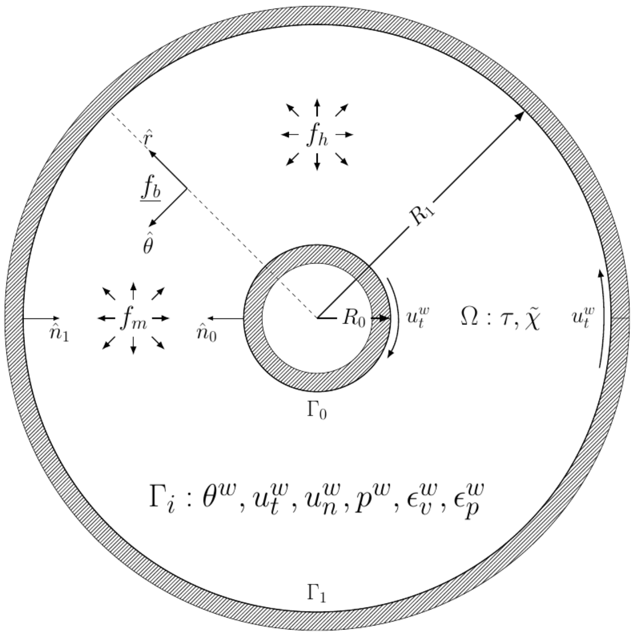


Figure 1: Annular Domain

f_m	f_h	$\underline{f_b}$
$\frac{2r^2}{\tau^3}$	$\frac{3}{\tau}$	$\begin{pmatrix} \left[\frac{4}{\tau} + \left(\frac{2r^2}{\tau^3}\right)\cos(\theta)\right] \\ \left[\frac{1}{\tau}\left(1 - \frac{5r^2}{27\tau^2}\right) + \frac{3r}{\tau^2}\right] \\ 0 \end{pmatrix}$
$\frac{1}{\tau}\left(1 + \frac{3r\cos(\theta)}{\tau}\right)$	$\frac{r}{2\tau^2}\cos(\theta) + \frac{r^2}{\tau^3}$	$\frac{1}{\tau}\begin{pmatrix} \left[1 + 2\left(\frac{r}{\tau} + \frac{9\tau}{5r}\right)\cos(\theta) + \frac{3r^2}{\tau^2}\right] \\ \left[\frac{2r}{\tau} + \left(\frac{3r^2}{\tau^2} + \frac{4\tau}{r}\right)\sin(\theta)\right] \\ 0 \end{pmatrix}$

Table 1: Examples of Source Function

$$f_m = \frac{1}{\tau}\left(M_o + \frac{M_1 r}{\tau}\cos(\theta) + \frac{M_2 r^2}{\tau^2}\right) \tag{16}$$

$$f_h = \frac{1}{\tau}\left(Q_o + \frac{Q_1 r}{\tau}\cos(\theta) + \frac{Q_2 r^2}{\tau^2}\right) \tag{17}$$

$$\underline{f_b} = \frac{1}{\tau}\begin{pmatrix} \left[F_0 + \left(F_1\left(\frac{r}{\tau} + \frac{9\tau}{5r}\right) + F_{22}\frac{r^2}{\tau^2}\right)\cos(\theta) + F_2\frac{r^2}{\tau^2}\right] \\ \left[G_0\left(1 - \frac{5r^2}{27\tau^2}\right) + G_1\frac{r}{\tau} + \left(G_2\frac{r^2}{\tau^2} + G_{11}\frac{\tau}{r}\right)\sin(\theta)\right] \\ 0 \end{pmatrix} \tag{18}$$

where $M_0, M_1, M_2, Q_0, Q_1, Q_2, F_0, F_1, F_2, F_{22}, G_0, G_1, G_2, G_{11} \in \mathbb{R}$. The values of these constants can be adjusted to create a range source functions which can in turn be used to specify a variety of problems. Simple functions for the source terms can be enabled by setting most of these constants to zero. However, it is important that the structure of these functions is unaltered. Examples of some acceptable function for the source terms are listed in Table 1

2 Code Description

2.1 SymPy Module

SymPy is a lightweight open-source Python library used for symbolic mathematics [2]. SymPy is written entirely in Python which allows for simple installation and use. This library offers a variety of capabilities ranging form basic symbolic arithmetic to calculus, linear algebra, discrete mathematics and quantum physics. Here we use this library to solve the linear system of equations arising from boundary conditions to obtain the values of the integration constants.

2.2 Code Structure

The boundary condition equations corresponding to the annular domain under consideration along with the generic analytical expressions for the 5 field variables $\left(\theta, p, \underline{u}, \underline{q}, \underline{\underline{\sigma}}\right)$ (see Appendix A) have been hard coded into the script presented. Consequentially, the only calculation that needs to be done involves solving the boundary condition equations, according to the boundary values provided by the user as an input, to obtain the required values of the integration constants. Once the integration constant values are determined, these values are plugged into the generic expressions for the field variables to get the final expressions for each field variable corresponding to the specific problem considered. Furthermore, this code exports the final expression of the field variables in three different formats which can then be used in tandem with a solver for the linearized R13 equations. This could serve the purpose of simply validating the solver or for performing convergence studies.

The script requires a YAML file, to be provided as input, that holds the boundary specifications and formulation parameters for the problem under consideration. The script then reads the YAML file and produces the solution corresponding to the specified problem. As mentioned above, the solution is then exported into the current working directory as a python script (.py), a C++ header file (.h) and a C++ script (.cpp) which conforms to the syntax required by DOLFIN [1] (python library popularly used to solve PDEs).

2.3 Code Execution

System requirements :

1. A system with Python installed
   (a) Execute the following commands to check
   ```
   python3 --version
   ```
2. The SymPy and SciPy libraries need to be installed
   (a) Execute the following commands to check
   ```
   pip3 list
   ```
   i. If pip3 is not installed then install using (on Ubuntu)
   ```
   sudo apt-get update
   sudo apt install python3-pip
   ```
   (b) If either of the libraries are not listed then install using
   ```
   pip3 install sympy
   ```
   ```
   pip3 install scipy
   ```

Code Execution :

1. Open/Create the YAML file which will contain the boundary condition values (Fig. 2).
2. Adjust/Set the values of the formulation, source and boundary parameters to describe the problem.
   (a) Comments are provided in the example YAML file (input.yml) describing each parameter.
   (b) Note that the source terms need to conform to the format specified in Section 1.3. Additionally the format for each source term is provided in the example YAML file (input.yml) for ease of use.
   (c) Consider the direction of the normal at both boundaries and assign the sign to the velocity terms at the boundaries appropriately.

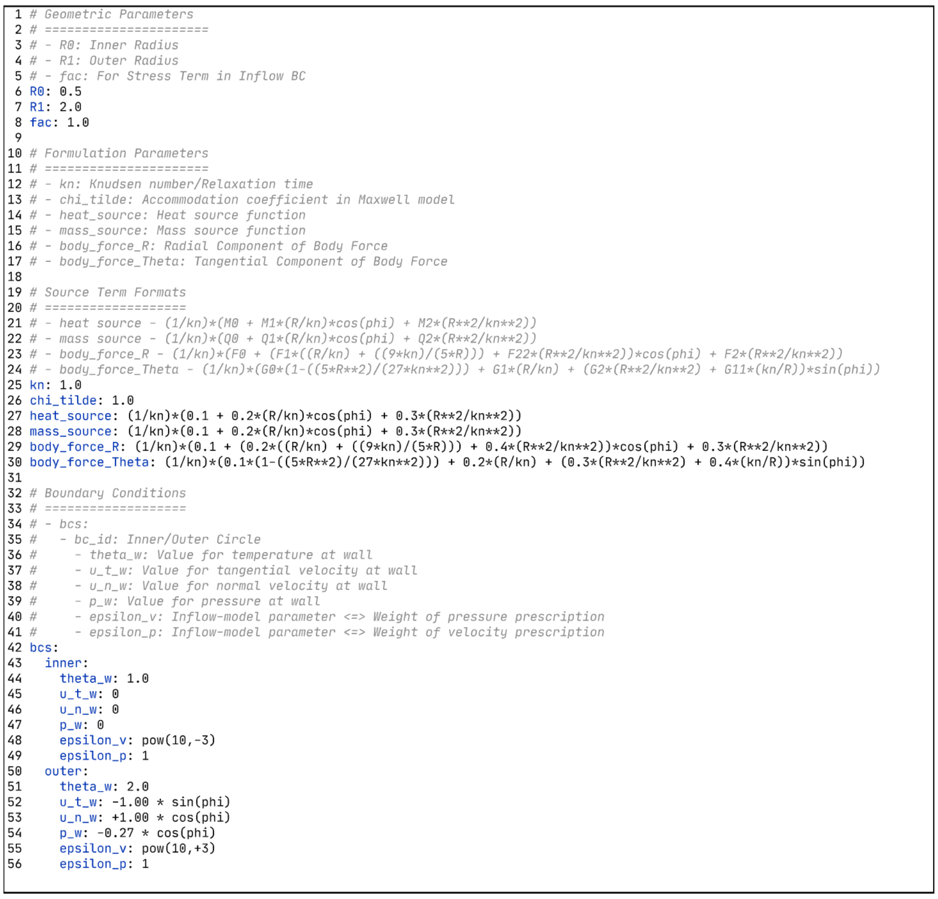
3. Ensure that the YAML file is in the same directory as the main script (exact_sol_gen_R13.py).
4. Open a command line interface.
5. Move into the directory containing the script and the YAML file.
6. Execute

   ```
   python3 exact_sol_gen_R13.py <input_file_name>.yml
   ```

   For example :

   ```
   python3 exact_sol_gen_R13.py input.yml
   ```

7. Wait until a message saying “Exact Solution Generated in current working directory” appears.
8. Find the exported solutions corresponding to the specified problem in the current working directory.



```yaml
# Geometric Parameters
# ======================
# - R0: Inner Radius
# - R1: Outer Radius
# - fac: For Stress Term in Inflow BC
R0: 0.5
R1: 2.0
fac: 1.0

# Formulation Parameters
# ======================
# - kn: Knudsen number/Relaxation time
# - chi_tilde: Accommodation coefficient in Maxwell model
# - heat_source: Heat source function
# - mass_source: Mass source function
# - body_force_R: Radial Component of Body Force
# - body_force_Theta: Tangential Component of Body Force

# Source Term Formats
# ===================
# - heat source - (1/kn)*(M0 + M1*(R/kn)*cos(phi) + M2*(R**2/kn**2))
# - mass source - (1/kn)*(Q0 + Q1*(R/kn)*cos(phi) + Q2*(R**2/kn**2))
# - body_force_R - (1/kn)*(F0 + (F1*((R/kn) + ((9*kn)/(5*R))) + F22*(R**2/kn**2))*cos(phi) + F2*(R**2/kn**2))
# - body_force_Theta - (1/kn)*(G0*(1-((5*R**2)/(27*kn**2))) + G1*(R/kn) + (G2*(R**2/kn**2) + G11*(kn/R))*sin(phi))
kn: 1.0
chi_tilde: 1.0
heat_source: (1/kn)*(0.1 + 0.2*(R/kn)*cos(phi) + 0.3*(R**2/kn**2))
mass_source: (1/kn)*(0.1 + 0.2*(R/kn)*cos(phi) + 0.3*(R**2/kn**2))
body_force_R: (1/kn)*(0.1 + (0.2*((R/kn) + ((9*kn)/(5*R))) + 0.4*(R**2/kn**2))*cos(phi) + 0.3*(R**2/kn**2))
body_force_Theta: (1/kn)*(0.1*(1-((5*R**2)/(27*kn**2))) + 0.2*(R/kn) + (0.3*(R**2/kn**2) + 0.4*(kn/R))*sin(phi))

# Boundary Conditions
# ===================
# - bcs:
#   - bc_id: Inner/Outer Circle
#     - theta_w: Value for temperature at wall
#     - u_t_w: Value for tangential velocity at wall
#     - u_n_w: Value for normal velocity at wall
#     - p_w: Value for pressure at wall
#     - epsilon_v: Inflow-model parameter <=> Weight of pressure prescription
#     - epsilon_p: Inflow-model parameter <=> Weight of velocity prescription
bcs:
  inner:
    theta_w: 1.0
    u_t_w: 0
    u_n_w: 0
    p_w: 0
    epsilon_v: pow(10,-3)
    epsilon_p: 1
  outer:
    theta_w: 2.0
    u_t_w: -1.00 * sin(phi)
    u_n_w: +1.00 * cos(phi)
    p_w: -0.27 * cos(phi)
    epsilon_v: pow(10,+3)
    epsilon_p: 1
```

Figure 2: Input YAML File

2.4 Using Exported Solution

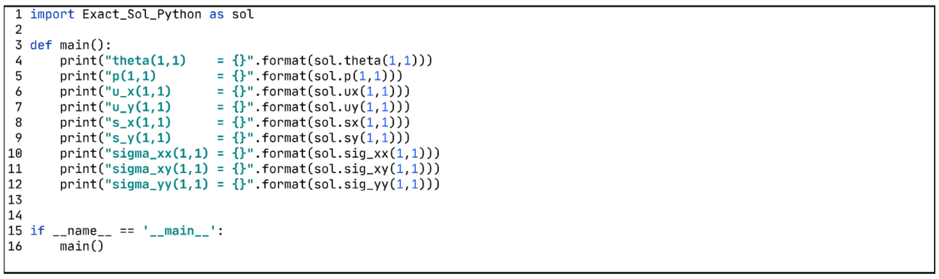
The output files generated are :

1. Exact_Sol_Python.py
2. Exact_Sol_C++.h
3. Exact_Sol_Dolfin.cpp

Examples depicting how these files can be imported and used in other programs are elaborated below.

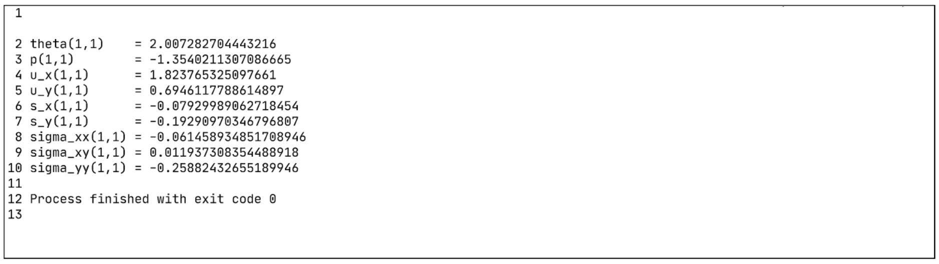
2.4.1 Python Format

Here a simple python script that imports the Exact_Sol_Python.py file and evaluates each field variable at the point $(1, 1)$ is shown (Fig. 3). The output produced by this python script is shown below (Fig. 4)



```python
import Exact_Sol_Python as sol

def main():
    print("theta(1,1)    = {}".format(sol.theta(1,1)))
    print("p(1,1)        = {}".format(sol.p(1,1)))
    print("u_x(1,1)      = {}".format(sol.ux(1,1)))
    print("u_y(1,1)      = {}".format(sol.uy(1,1)))
    print("s_x(1,1)      = {}".format(sol.sx(1,1)))
    print("s_y(1,1)      = {}".format(sol.sy(1,1)))
    print("sigma_xx(1,1) = {}".format(sol.sig_xx(1,1)))
    print("sigma_xy(1,1) = {}".format(sol.sig_xy(1,1)))
    print("sigma_yy(1,1) = {}".format(sol.sig_yy(1,1)))


if __name__ == '__main__':
    main()
```

Figure 3: Import Exact_Sol_Python.py

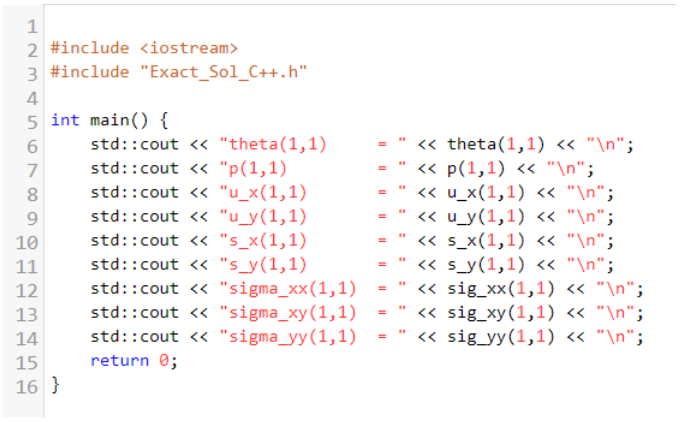


```

theta(1,1)    = 2.007282704443216
p(1,1)        = -1.3540211307086665
u_x(1,1)      = 1.823765325097661
u_y(1,1)      = 0.6946117788614897
s_x(1,1)      = -0.07929989062718454
s_y(1,1)      = -0.19290970346796807
sigma_xx(1,1) = -0.061458934851708946
sigma_xy(1,1) = 0.011937308354488918
sigma_yy(1,1) = -0.25882432655189946

Process finished with exit code 0

```

Figure 4: Output - Python script

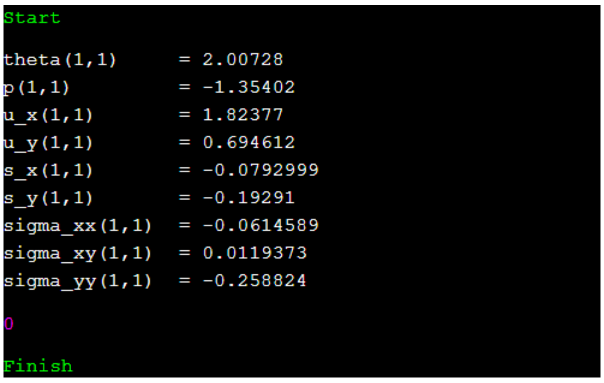
2.4.2 C++ Format

In a similar fashion to the previous section, the simple C++ code shown here (Fig. 5) imports the header file "Exact_Sol+C++.h" and evaluates the generated solution corresponding to each field variable at the point $(1, 1)$. The output generated is also provided (Fig. 6). Note that since the evaluation of the modified bessel functions is required within the header file, it is essential to ensure that the C++ Boost library is installed and available for use.



```cpp

#include <iostream>
#include "Exact_Sol_C++.h"

int main() {
    std::cout << "theta(1,1)     = " << theta(1,1) << "\n";
    std::cout << "p(1,1)         = " << p(1,1) << "\n";
    std::cout << "u_x(1,1)       = " << u_x(1,1) << "\n";
    std::cout << "u_y(1,1)       = " << u_y(1,1) << "\n";
    std::cout << "s_x(1,1)       = " << s_x(1,1) << "\n";
    std::cout << "s_y(1,1)       = " << s_y(1,1) << "\n";
    std::cout << "sigma_xx(1,1)  = " << sig_xx(1,1) << "\n";
    std::cout << "sigma_xy(1,1)  = " << sig_xy(1,1) << "\n";
    std::cout << "sigma_yy(1,1)  = " << sig_yy(1,1) << "\n";
    return 0;
}
```

Figure 5: Import Exact_Sol_C++.h

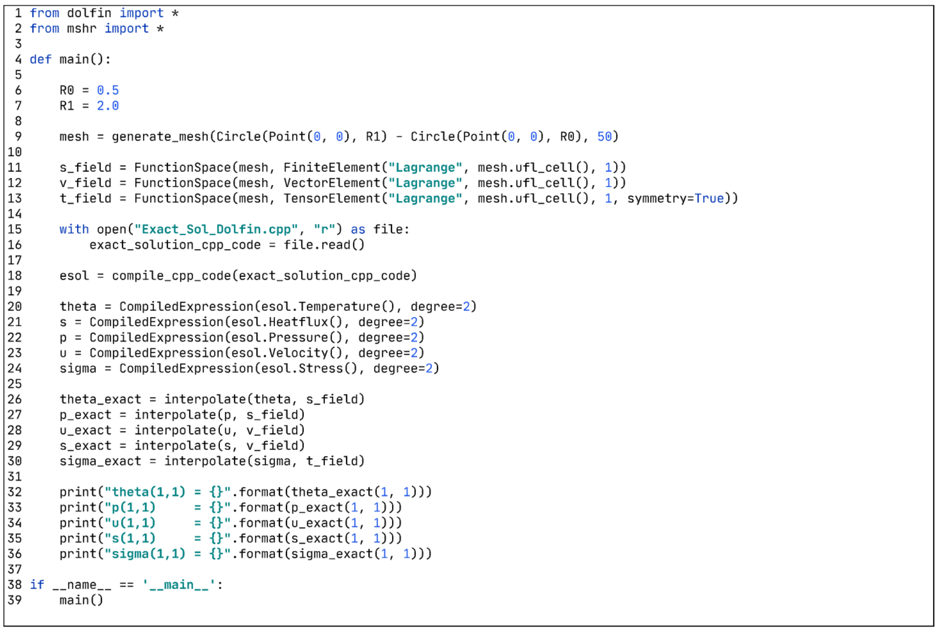


```
Start

theta(1,1)      = 2.00728
p(1,1)          = -1.35402
u_x(1,1)        = 1.82377
u_y(1,1)        = 0.694612
s_x(1,1)        = -0.0792999
s_y(1,1)        = -0.19291
sigma_xx(1,1)   = -0.0614589
sigma_xy(1,1)   = 0.0119373
sigma_yy(1,1)   = -0.258824

0

Finish
```

Figure 6: Output - C++ script

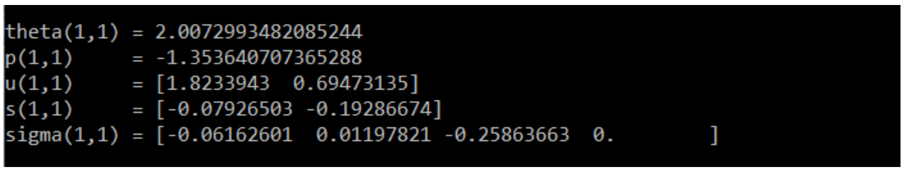
2.4.3 DOLFIN Format

Along the same lines as the two previous sections, here this sample python program reads the Exact_Sol_Dolfin.cpp file using the DOLFIN python module then interpolates it over a mesh for the annular domain and finally prints the values of each variable at the point $(1,1)$ (Fig. 7). The resulting output (Fig. 8) is quite similar to the outputs from the previous two sections.



```python
from dolfin import *
from mshr import *

def main():

    R0 = 0.5
    R1 = 2.0

    mesh = generate_mesh(Circle(Point(0, 0), R1) - Circle(Point(0, 0), R0), 50)

    s_field = FunctionSpace(mesh, FiniteElement("Lagrange", mesh.ufl_cell(), 1))
    v_field = FunctionSpace(mesh, VectorElement("Lagrange", mesh.ufl_cell(), 1))
    t_field = FunctionSpace(mesh, TensorElement("Lagrange", mesh.ufl_cell(), 1, symmetry=True))

    with open("Exact_Sol_Dolfin.cpp", "r") as file:
        exact_solution_cpp_code = file.read()

    esol = compile_cpp_code(exact_solution_cpp_code)

    theta = CompiledExpression(esol.Temperature(), degree=2)
    s = CompiledExpression(esol.Heatflux(), degree=2)
    p = CompiledExpression(esol.Pressure(), degree=2)
    u = CompiledExpression(esol.Velocity(), degree=2)
    sigma = CompiledExpression(esol.Stress(), degree=2)

    theta_exact = interpolate(theta, s_field)
    p_exact = interpolate(p, s_field)
    u_exact = interpolate(u, v_field)
    s_exact = interpolate(s, v_field)
    sigma_exact = interpolate(sigma, t_field)

    print("theta(1,1) = {}".format(theta_exact(1, 1)))
    print("p(1,1)     = {}".format(p_exact(1, 1)))
    print("u(1,1)     = {}".format(u_exact(1, 1)))
    print("s(1,1)     = {}".format(s_exact(1, 1)))
    print("sigma(1,1) = {}".format(sigma_exact(1, 1)))

if __name__ == '__main__':
    main()
```

Figure 7: Import Exact_Sol_Dolfin.cpp



```
theta(1,1) = 2.0072993482085244
p(1,1)     = -1.353640707365288
u(1,1)     = [1.8233943  0.69473135]
s(1,1)     = [-0.07926503 -0.19286674]
sigma(1,1) = [-0.06162601  0.01197821 -0.25863663  0.        ]
```

Figure 8: Output - DOLFIN

2.5 Limitations

Some of the limitations associated with the range of cases that this code can accommodate are mentioned below

1. As mentioned earlier, although there might be other source functions which allow for non-homogeneous analytical solutions, this code is only able to deal with source functions of the forms specified in Section 1.3.
2. The boundary values have to be of the form $a + b\cos(\theta)$ or $a + b\sin(\theta)$ where $(a, b) \in \mathbb{R}$ as mentioned in Section 1.3.
3. The code is unable to generate solutions when the Knudsen number (Kn) is less than about 0.005. The system becomes ill-conditioned.

A Analytical Expressions of the field variables

Note that in the expressions shown below *log* denotes the natural logarithm and $I_n(\cdot)$ and $K_n(\cdot)$ represent the modified Bessel functions of first and second kind respectively. The subscript n denotes the order of the Bessel Function. A list of all the integration constants is shown below

$$\left(c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, C_1^{I,1}, C_1^{K,1}, C_1^{I,2}, C_1^{K,2}, C_2^{I,1}, C_2^{K,1}, C_2^{I,2}, C_2^{K,2}, C_3^{I,1}, C_3^{K,1}, C_3^{I,2}, C_3^{K,2}\right)$$

A.1 Temperature

$$\begin{aligned}\theta(r,\phi) = {} & \cos(\phi)\Big[-\frac{16}{15}K_1\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)C_1^{K,2} - \frac{16}{15}I_1\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)C_1^{I,2} + \frac{r^3(M_1-Q_1)}{30\tau^3} + \frac{c_{11}r}{\tau} + \frac{c_{10}\tau}{r}\Big] \\ & - \frac{8}{15}K_0\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)C_1^{K,1} - \frac{8}{15}I_0\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)C_1^{I,1} + \frac{r^4(M_2-Q_2)}{60\tau^4} - \frac{4}{15}c_3\log\left(\frac{r}{\tau}\right) \\ & + \frac{r^2(5M_0-84M_2-5Q_0+124Q_2)}{75\tau^2} + c_5\end{aligned} \tag{19}$$

A.2 Pressure

$$\begin{aligned}p(r,\phi) = {} & \cos(\phi)\Bigg[-\frac{8}{3}I_1\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)C_1^{I,2} - \frac{8}{3}K_1\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)C_1^{K,2} + \frac{r^3(3F_{22}+G_2)}{8\tau^3} + \frac{2F_1r^2}{3\tau^2} - G_{11} \\ & + \frac{r(3M_1+2Q_1)}{5\tau} - \frac{2c_9r}{\tau} + \frac{c_8\tau}{r}\Bigg] - \frac{4}{3}I_0\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)C_1^{I,1} - \frac{4}{3}K_0\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)C_1^{K,1} \\ & + \frac{F_2r^3}{3\tau^3} + \frac{F_0r}{\tau} + \frac{4r^2(3M_2+2Q_2)}{15\tau^2} + c_4\end{aligned} \tag{20}$$

A.3 Velocity

$$\begin{aligned}u_r(r,\phi) = {} & \cos(\phi)\Bigg[\left(I_2\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right) - I_0\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right)\right)\frac{2C_3^{I,2}}{5} + \left(K_2\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right) - K_0\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right)\right)\frac{2C_3^{K,2}}{5} \\ & + \frac{r^2(-25F_{22}-75G_2+42M_1-2Q_1)}{120\tau^2} + \frac{r^4(F_{22}+3G_2)}{192\tau^4} + \frac{F_1r^3}{45\tau^3} - \frac{52F_1r}{45\tau} + \frac{48F_1\tau}{25r} - \frac{c_9r^2}{4\tau^2} \\ & + \frac{c_7\tau^2}{4r^2} - \frac{3c_{10}\tau^2}{2r^2} + c_8\left(\frac{5\tau^2}{3r^2} - \frac{1}{2}\log\left(\frac{r}{\tau}\right) + \frac{1}{2}\right) + c_{12}\Bigg] + \frac{M_2r^3}{4\tau^3} + \frac{M_0r}{2\tau} + \frac{c_2\tau}{2r} - \frac{2c_3\tau}{5r}\end{aligned} \tag{21}$$

$$\begin{aligned}u_\phi(r,\phi) = {} & \sin(\phi)\Bigg[\left(I_0\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right) + I_2\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right)\right)\frac{2C_3^{I,2}}{5} + \left(K_0\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right) + K_2\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right)\right)\frac{2C_3^{K,2}}{5} \\ & - \frac{r^2(-25F_{22}-75G_2+2M_1-2Q_1)}{40\tau^2} - \frac{5r^4(F_{22}+3G_2)}{192\tau^4} - \frac{4F_1r^3}{45\tau^3} + \frac{104F_1r}{45\tau} + \frac{3c_9r^2}{4\tau^2} \\ & - \frac{3c_{10}\tau^2}{2r^2} + \frac{c_7\tau^2}{4r^2} - c_8\left(-\frac{5\tau^2}{3r^2} - \frac{1}{2}\log\left(\frac{r}{\tau}\right)\right) - c_{12}\Bigg] - \frac{2}{5}I_1\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right)C_3^{I,1} \\ & - \frac{2}{5}K_1\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right)C_3^{K,1} + \frac{G_0r^4}{81\tau^4} - \frac{G_1r^3}{8\tau^3} - \frac{52G_0r^2}{81\tau^2} + \frac{16G_0}{15} + \frac{c_6r}{\tau} + \frac{c_1\tau}{2r}\end{aligned} \tag{22}$$

A.4 Heat Flux

$$\begin{aligned}q_r(r,\phi) = \cos(\phi)\Bigg[&\left(I_0\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right) - I_2\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right)\right)C_3^{I,2} + \left(K_0\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right) - K_2\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right)\right)C_3^{K,2}\\ &+ \frac{3r^2(F_{22}+3G_2-2M_1+2Q_1)}{16\tau^2} + \frac{3}{10}(9F_{22}+27G_2-15M_1+20Q_1) + \frac{F_1 r}{2\tau} + \frac{15c_{10}\tau^2}{4r^2}\\ &- \frac{3c_8\tau^2}{2r^2} - 3c_9 - \frac{15c_{11}}{4}\Bigg] + \frac{r^3(Q_2-M_2)}{4\tau^3} + \frac{r(Q_0-M_0)}{2\tau} + \frac{c_3\tau}{r}\end{aligned} \tag{23}$$

$$\begin{aligned}q_\phi(r,\phi) = -\sin(\phi)\Bigg[&\left(I_0\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right) + I_2\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right)\right)C_3^{I,2} + \left(K_0\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right) + K_2\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right)\right)C_3^{K,2}\\ &+ \frac{r^2(9F_{22}+27G_2-2M_1+2Q_1)}{16\tau^2} + \frac{3}{10}(9F_{22}+27G_2-15M_1+20Q_1) + \frac{F_1 r}{\tau} + \frac{3c_8\tau^2}{2r^2}\\ &- \frac{15c_{10}\tau^2}{4r^2} - 3c_9 - \frac{15c_{11}}{4}\Bigg] + I_1\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right)C_3^{I,1} + K_1\left(\frac{\sqrt{\frac{5}{9}}r}{\tau}\right)C_3^{K,1} + \frac{5G_0 r^2}{18\tau^2} - \frac{3G_1 r}{2\tau}\end{aligned} \tag{24}$$

A.5 Stress

$$\begin{aligned}\sigma_{rr}(r,\phi) = \cos(\phi)\Bigg[&\left(I_3\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right) - I_1\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)\right)C_2^{I,2} + \left(\frac{5}{3}I_1\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right) + I_3\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)\right)C_1^{I,2}\\ &+ \left(K_3\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right) - K_1\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)\right)C_2^{K,2} + \left(\frac{5}{3}K_1\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right) + K_3\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)\right)C_1^{K,2}\\ &+ \frac{r^3(-F_{22}-3G_2)}{24\tau^3} - \frac{2F_1 r^2}{15\tau^2} + \frac{6F_1}{5} - \frac{2r(3M_1+2Q_1)}{15\tau} + \frac{c_7\tau^3}{r^3} + \frac{c_9 r}{\tau} + \frac{c_8\tau}{r}\Bigg]\\ &+ \left(I_2\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right) - I_0\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)\right)C_2^{I,1} + \left(\frac{1}{3}I_0\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right) + I_2\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)\right)C_1^{I,1}\\ &+ \left(K_2\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right) - K_0\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)\right)C_2^{K,1} + \left(\frac{1}{3}K_0\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right) + K_2\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)\right)C_1^{K,1}\\ &+ \frac{r^2(-3M_2-2Q_2)}{6\tau^2} + \frac{1}{75}(-15M_0-72M_2-10Q_0-48Q_2) + \frac{c_2\tau^2}{r^2}\end{aligned} \tag{25}$$

$$\begin{aligned}\sigma_{r\phi}(r,\phi) = \sin(\phi)\Bigg[&\left(I_3\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right) - I_1\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)\right)C_2^{I,2} + \left(I_3\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right) - I_1\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)\right)C_1^{I,2}\\ &+ \left(K_3\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right) - K_1\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)\right)C_2^{K,2} + \left(K_3\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right) - K_1\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)\right)C_1^{K,2}\\ &+ \frac{r^3(F_{22}+3G_2)}{12\tau^3} + \frac{F_1 r^2}{5\tau^2} - \frac{3F_1}{5} + \frac{r(3M_1+2Q_1)}{15\tau} + \frac{c_7\tau^3}{r^3} - \frac{c_9 r}{\tau}\Bigg] - \frac{G_0 r^3}{27\tau^3}\\ &+ \frac{G_1 r^2}{4\tau^2} + \frac{G_0 r}{3\tau} + \frac{c_1\tau^2}{r^2}\end{aligned} \tag{26}$$

$$
\begin{aligned}
\sigma_{\phi\phi}\left(r,\phi\right) = & -\cos(\phi)\Bigg[\left(3I_1\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)+I_3\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)\right)C_2^{I,2}+\left(\frac{1}{3}I_1\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)+I_3\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)\right)C_1^{I,2} \\
& +\left(3K_1\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)+K_3\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)\right)C_2^{K,2}+\left(\frac{1}{3}K_1\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)+K_3\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)\right)C_1^{K,2} \\
& +\frac{r^3\left(-F_{22}-3G_2\right)}{24\tau^3}-\frac{2F_1r^2}{15\tau^2}+\frac{6F_1}{5}+\frac{c_7\tau^3}{r^3}+\frac{c_9r}{\tau}+\frac{c_8\tau}{r}\Bigg] \\
& -\left(I_0\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)+I_2\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)\right)C_2^{I,1}-\left(I_2\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)-\frac{1}{3}I_0\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)\right)C_1^{I,1} \\
& -\left(K_0\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)+K_2\left(\frac{\sqrt{\frac{3}{2}}r}{\tau}\right)\right)C_2^{K,1}-\left(K_2\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)-\frac{1}{3}K_0\left(\frac{\sqrt{\frac{5}{6}}r}{\tau}\right)\right)C_1^{K,1} \\
& -\frac{r^2\left(-3M_2-2Q_2\right)}{30\tau^2}+\frac{1}{75}\left(-15M_0-72M_2-10Q_0-48Q_2\right)-\frac{c_2\tau^2}{r^2}
\end{aligned}
\tag{27}
$$

References

[1] G. N. Wells A. Logg and J. Hake. Dolfin: a c++/python finite element library. 84, 2012.

[2] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[3] H. Struchtrup and M. Torrilhon. Regularization of grad's 13 moment equations : Derivation and linear analysis. *Physics of Fluids*, 15:2668–2680, 2003.

[4] Lambert Theisen. Simulation of non-equilibrium gas flows using the fenics computing platform. Master's thesis, RWTH Aachen, 2019.

[5] Manuel Torrilhon. Slow gas microflow past a sphere: Analytical solution based on moment equations. *Physics of Fluids*, 22, 2010.

[6] Manuel Torrilhon and Henning Struchtrup. Boundary conditions for regularized 13-moment-equations for micro-channel-flows. *Journal of Computational Physics*, 227:1982–2011, 2007.

[7] A. Westerkamp. *A Continuous Interior Penalty Method for the Linear Regularized 13-Moment Equations Describing Rarefied Gas Flows.* PhD thesis, RWTH Aachen, 2017.

[8] A. Westerkamp and M. Torrilhon. Finite element methods for the linear regularized 13-moment equations describing slow rarefied gas flows. *Journal of Computational Physics*, 389:1–21, 2019.

[9] Armin Westerkamp and Manuel Torrilhon. Slow rarefied gas flow past a cylinder: Analytical solution in comparison to the sphere. *AIP Conference Proceedings*, 1501:207–214, 2012.