OIL DROPLETS RISING THROUGH DENSITY-STRATIFIED FLUID AT LOW REYNOLDS NUMBER

Holly H. Arrowood

A dissertation submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Mathematics in the College of Arts and Sciences.

Chapel Hill
2018

Approved by:

Roberto Camassa

Richard McLaughlin

David Adalsteinsson

Laura Miller

Gregory Forest

ABSTRACT

Holly H. Arrowood: Oil Droplets Rising through Density-Stratified Fluid
at Low Reynolds Number
(Under the direction of Roberto Camassa and Richard McLaughlin)

Sharp density stratifications occur naturally in the ocean, and play a important role in the dynamics of settling marine snow particles and the dispersion of oil from seeps or spills. When passing through a sharp density transition between two miscible fluid layers, rising immiscible drops slow to well below terminal velocity due to the entrainment of bottom-layer fluid. This dissertation presents an experimentally-validated, numerically-assisted, first-principles model of an oil drop rising in a cylindrical tank of sharply-stratified fluid at low Reynolds number.

The mathematical model presented in this dissertation model adapts previous work on a settling sphere in sharply-stratified fluid to the significantly more complex case of a rising fluid sphere. A Green's function formulation is used to model the density-driven flow, while the method of reflections and method of multipoles are adapted to correct for the increased drag on the drop due to the cylindrical boundary. The numerical implementation of this model is also based upon previous work for a solid sphere, but requires attention to a number of numerical considerations not present in the solid sphere case. Finally, an experimental study is used to validate the final model within its regime of validity. Additional experimental observations of surfactant effects are detailed as well.

To Josephine, Raymond, Gladys, and Bruce

ACKNOWLEDGEMENTS

The work detailed in this dissertation would not have been possible without the help and support of my advisors, committee, labmates, friends, and family. Research can be isolating, but it's much easier with the knowledge that one has many wonderful people cheering you on.

I would especially like to thank my advisors, Roberto Camassa and Richard McLaughlin, for their incredible patience with me over the past years of research. Both have a keen sense of when to provide insight and guidance versus when to let a student tussle his or her own way through their growth as an independent researcher, which has been invaluable to my personal development.

My committee members, Laura Miller, David Adalsteinsson, and Gregory Forest have all been vital to my work on this project. A special thanks to Prof. Miller for her amazing advice, in regards to both academic pursuits and life in general, to Prof. Adalsteinsson for his patience and insight on my code and his spectacular work on DataTank, which allowed for the efficient processing and visualization of my experimental data, and to Prof. Forest for his consistent support for the success of all graduate students in our department. It has been a true privilege to have the opportunity to work with such a stellar cast of researchers, and such genuinely kind and supportive people in general.

The experimental work conducted in the process of validating the model in this dissertation required many years of refinement to experimental protocol and setup, in which I was aided by numerous dedicated postdoctoral and undergraduate researchers. Dr. Shilpa Khatri, from whom I inherited this project, deserves special mention for her mentorship at the very beginning of my graduate career. Undergraduate researchers Arthur Wood, Amar Patel, Issam Ahmed, and Mohit Bajaj have each been invaluable in their help with conducting experiments and sharing in the experimental design process. Postdocs Dan Harris and Pierre-Yves Passaggia have both been extremely generous with their experimental and mathematical expertise over the past few years.

My friends and family have also been essential to my well-being throughout this journey. My fellow graduate students were always willing to discuss research ideas or simply kick back and relax

after a hard day's work. My family always fostered my love of learning, curiosity, and work ethic, even when my interests took me well outside their fields of interest.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Haloclines and thermoclines in the ocean result in natural density gradients, which have been observed to influence the behavior of particles and droplets settling (as in the case of marine snow [3]) or rising (as in the case of oil plumes [4]) through them. The abundance of nutrients at these density discontinuities makes them into hotspots of biological activity [5]. In understanding the behavior of particles or droplets passing through these sharp density gradients, it is often useful to study the behavior of a single drop or particle in an idealized and therefore hopefully more tractable scenario. Models have been developed for solid and porous spheres settling through a sharp density gradient[6, 7], and for oil plumes moving through various density gradients[4]. This dissertation extends this work to the case of a single drop of immiscible fluid rising through two density-stratified, miscible fluid layers.

The problem of creeping flow around solid and fluid spheres in unbounded, homogeneous ambient fluid is a classical problem of fluid dynamics. The case of a solid sphere moving at zero Reynolds number in an infinite fluid domain was first addressed by Stokes in 1851 [8], and the analogous problem for a fluid sphere was independently solved in 1911 by Hadamard [9] and Rybczynski [10]. For small but non-zero Reynolds number flow about a sphere settling through unbounded ambient fluid, Oseen [11] provided a small-Reynolds number correction to Stokes' work in 1910, which was later improved by Lamb in 1911 [12] and Proudman and Pearson in 1956 [13]. The analogous case of an immiscible droplet is significantly more complex, however, as the droplet will deform in the case of nonzero Reynolds number. The flow about a drop at low but nonzero Reynolds number was addressed by Acrivos and Taylor in 1963 [14], and extended by Brignell in 1973 [15]. Further analytic results regarding Stokes flow about fluid and solid spheroids were also obtained by Deo and Datta in 2003 [16] and Srivastava et al in 2012 [17].

Stokes flow about a sphere in a cylinder is a problem of great practical importance, with applications from falling sphere viscometers to microfluidics. The first treatment of the problem, in

the form of corrections to the drag on spheres falling close to the boundary walls of a cylinder of fluid at rest is due to Ladenberg in 1907, with later correction by Fáxen in 1923 [18, 19]. Happel and Byrne (1954) later used the Method of Reflections to find the drag due to the cylinder wall on a sphere in moving fluid [20]. Haberman and Sayre (1958) [21] use the Multipole Method (which is also described by Linton [22]) to compute the drag on solid and fluid spheres moving through cylindrical tanks of both stationary and moving ambient fluid. An additional simplified mathematical model is presented by Blanchette [23], though his results are primarily numerical.

Of key importance in light of our application to density discontinuities in the ocean is the study of motion of particles and drops in sharply-stratified fluid. First-principles models of solid and porous spheres settling in sharply-stratified fluid have been developed in recent years [6, 7, 1]. Additional work addresses the case of oil plumes in sharp and linear stratifications [4]. The case of a single droplet is, however, largely unexplored, save some numerical work for moderate Reynolds number [24]. The work presented in this dissertation takes the obvious next step of extending the first-principles models for the solid sphere case to the case of a immiscible liquid drop rising in a cylindrical tank of sharply-stratified fluid. Though similar in many respects to the solid sphere case, this problem presents is significantly more complex in a number of respects, among which one of the chief difficulties is the deformability of a liquid drop, so that the geometry of the problem is no longer fixed.

The deformability of a liquid drop is one of the primary sources of difficulty in the extension of existing mathematical models for solid spheres to this case. The imposition of a prescribed drop shape in addition to the kinematic and dynamic conditions at the drop surface and the condition at the wall of the tank or at infinity leads to an overdetermined system. As discussed by Batchelor [25], and refined by Taylor and Acrivos [14] and Brignell[15], a fluid drop rising in infinite fluid at zero Reynolds number will not deform (See Appendix B.1 for further details). However, for nonzero Reynolds number or for different boundary conditions (as in the case at hand, with a cylindrical container and stratified ambient fluid), the deformation of the drop may have a significant effect. This dissertation is restricted in scope to those cases in which the drop may be assumed to remain approximately spherical.

Following an approach analogous to one previously used in the case of a settling sphere [6, 1], this dissertation presents a numerically-assisted, first-principles model of the idealized case of a

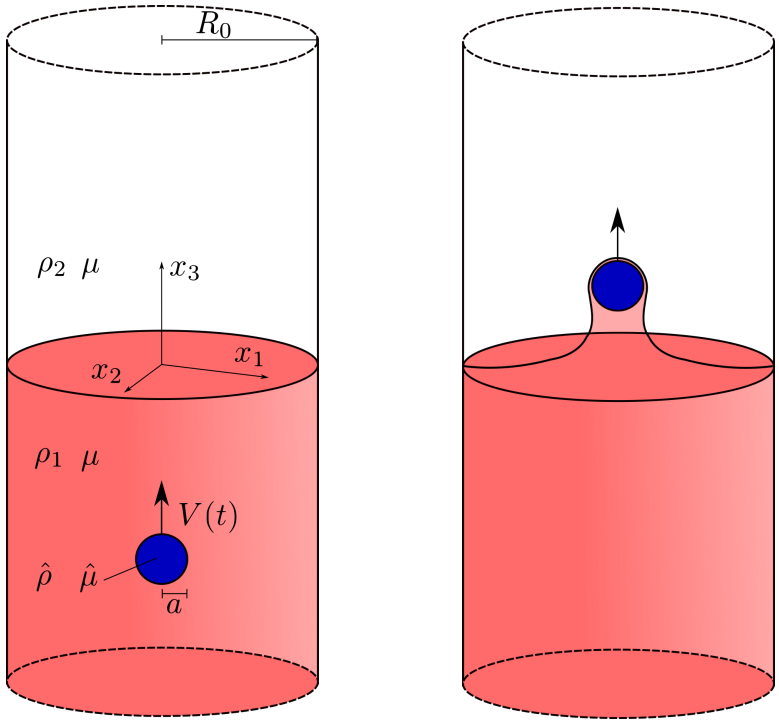

Figure 1.1: A drop of radius a, density $\hat{\rho}$, and viscosity $\hat{\mu}$ rising due to bouyancy along the vertical axis of a cylindrical tank of radius R_0 of sharply-stratified fluid with viscosity μ, bottom-layer density ρ_1, and top-layer density ρ_2. The image to the left depicts the setup at the initial time, with the density distribution undisturbed, and the image to the right is after the drop has risen through and displaced the density distribution. In the lab frame Cartesian coordinates (x_1, x_2, x_3) are used.

single oil drop rising along the vertical axis of a cylindrical tank of sharply-stratified fluid in the low-Reynolds number limit (see Fig. 1.1). To verify this model, an experimental study of the velocity of the drops and the deformation of the density distribution was conducted for a wide range of experimental parameters. A first correction to the shape of the droplet as it deforms due to the stratified ambient density distribution is also given. Ch. 2 of this dissertation presents the derivation of the first-principles mathematical model, the numerical implementation of which is described in Ch. 3. The experimental verification of this model is detailed in Ch.4, together with some additional experimental observations. With the model thus rigorously verified, some nondimensional trends in the model are explored in Ch.5. Finally, a summary of important results and possible future research directions is given in Ch.6.

The primary original theoretical results in this dissertation consist of the reformulation of the Method of Reflections [20] for the case of a drop, the adaptation of the Method of Multipoles [21] to the case of a drop in sharply-stratified fluid, and the evaluation of the cefficients of tangential stress

using the reciprocal theorem. Additional minor but (to the best of the author's knowledge) original numerical techniques for the tracking of an interface described by Lagrangian marker points near a hyperbolic stagnation point are described. Novel experimental results are included as well, including both the model validation and some previously undescribed experimental observations regarding surfactant effects in sharply-stratified fluid.

CHAPTER 2

Mathematical Model

This chapter describes the construction of the theoretical model and the derivation of the final integro-differential equations of motion for an immiscible drop rising through a cylindrical tank of sharply-stratified fluid. Preexisting models of droplets rising in both infinite and cylindrical tanks of homogeneous fluid are described and adapted to this situation, with the bouyancy-driven flow solved using a Green's function formulation due to Oseen [11].

Exploiting the linearity of the Stokes equations, we decompose the flow into that driven by the displacement of the density distribution (the "perturbation flow") and the flow around a drop rising through a cylindrical tank with a static density distribution (the "Stokes flow"), coupled by the dynamic boundary condition at the drop surface. Each of these are treated separately in our analysis, then combined in the final integro-differential equations of motion for the drop, which are integrated numerically at each timestep in our final, numerically-assisted model.

The following work utilizes techniques developed in previous work by Camassa et al. [6] for a solid sphere settling in a cylindrical tank of sharply-stratified fluid, the work of Haberman [21] on a droplet rising through a cylinder, and the work of Happel and Byrne [20] on a sphere in a cylindrical tube. These techniques are summarized and then adapted for our situation in the following sections.

Equations of Motion and Boundary Conditions

Consider a drop of immiscible fluid of density $\hat{\rho}$ and dynamic viscosity $\hat{\mu}$ rising at velocity $\mathbf{V}(t) = (0, 0, V(t))$ through an infinite cylindrical tank of radius R_0 of fluid with density distribution $\rho(\mathbf{x}, t)$ and viscosity μ (note that quantities related to the fluid constituting the drop are denoted with a carat). The velocity field within the drop is denoted $\hat{\mathbf{v}}$. Assume that the volume of the drop is conserved at that of a spherical drop of radius a, with drop surface given by $S(t)$.

Outside the drop, we have the Navier-Stokes equations for incompressible, Newtonian fluid of velocity $\mathbf{v}(t)$ and variable density $\rho(\mathbf{x}, t)$ at a given observation point $\mathbf{x}$

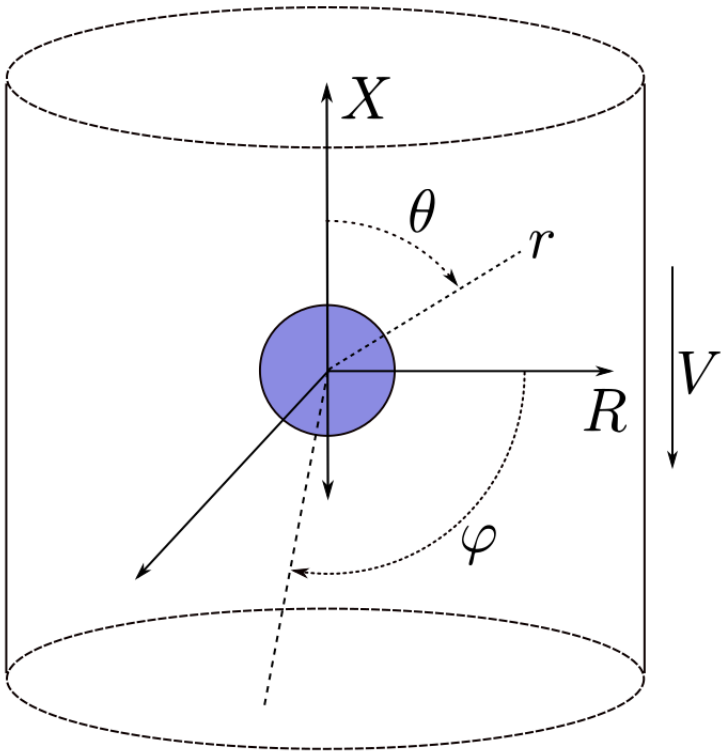

Figure 2.1: Diagram of coordinate systems in the frame of reference moving with the center of mass of the drop. Cylindrical coordinates $\mathbf{x} = (R, \varphi, X)$ and spherical coordinates $\mathbf{x} = (r, \varphi, \theta)$ are used.

$$\rho(\mathbf{x},t)\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v}\right) = \rho(\mathbf{x},t)\mathbf{g} - \nabla p + \mu\nabla^2\mathbf{v} \tag{2.1}$$

$$\nabla\cdot\mathbf{v} = 0, \tag{2.2}$$

where $\mathbf{g} = (0, 0, g)$ is the gravitational acceleration vector with magnitude $g = 981\text{cm/s}^2$.

Similarly, within the drop we have

$$\hat{\rho}\left(\frac{\partial \hat{\mathbf{v}}}{\partial t} + \hat{\mathbf{v}}\cdot\nabla\hat{\mathbf{v}}\right) = \hat{\rho}\mathbf{g} - \nabla\hat{p} + \mu\nabla^2\hat{\mathbf{v}} \tag{2.3}$$

$$\nabla\cdot\hat{\mathbf{v}} = 0. \tag{2.4}$$

Finally, for conservation of mass we have

$$\frac{\partial \rho}{\partial t} + \mathbf{v} \cdot \nabla \rho = 0. \tag{2.5}$$

where the initial, undisturbed density distribution is a sharp two-layer stratification with top density ρ_1 and bottom density ρ_2, so that

$$\rho(\mathbf{x}, 0) = \rho_0(z) = \rho_2 + (\rho_1 - \rho_2)H(z) \tag{2.6}$$

Note that we have ignored the diffusion of the stratified fluid. Our experimental observations of a persistent sharp interface, made visible by dye or by the index of refraction of the fluid, indicate very low diffusivity, and as shown in Lin's thesis ([26] section 2.8), mixing due to shear in the sphere case would be very low; this result should directly apply to our case as well.

Finally, for the equation of motion of the drop, we have

$$m_d \frac{dV}{dt} = m_d g + \int_S \sigma \cdot \mathbf{n} dS, \tag{2.7}$$

where m_d denotes the mass of the drop.

At the cylinder wall, we impose the no-slip condition, so that

$$\mathbf{v} = 0 \text{ for } \sqrt{x_1^2 + x_2^2} = R_0, -\infty < x_3 < \infty, \tag{2.8}$$

and we require that the ambient fluid be undisturbed at ∞, so that

$$\mathbf{v} \to 0 \text{ as } |x_3| \to \infty. \tag{2.9}$$

At the drop surface $S(t)$, we have the usual kinematic and dynamic boundary conditions for the interface between two immiscible fluids. These are continuity of velocity,

$$\mathbf{v} = \hat{\mathbf{v}} \text{ for } \mathbf{x} \in S(t) \tag{2.10}$$

continuity of tangential stress at the drop surface, so that for any $\boldsymbol{\tau}$ orthogonal to the unit normal

vector to the drop surface $\mathbf{n}$, we have

$$\boldsymbol{\tau} \cdot [(\hat{\mu}(\nabla \hat{\mathbf{v}} + (\nabla \hat{\mathbf{v}})^T) - \mu(\nabla \mathbf{v} + (\nabla \mathbf{v})^T)) \cdot \mathbf{n}] = 0 \text{ for } \mathbf{x} \in S(t) \tag{2.11}$$

and finally, assuming uniform surface tension, we require the following of normal stress

$$\mathbf{n} \cdot [(-(\widehat{p} - p)I + 2\hat{\mu}(\nabla \hat{\mathbf{v}} + (\nabla \hat{\mathbf{v}})^T - \mu(\nabla \mathbf{v} + (\nabla \mathbf{v})^T)] \cdot \mathbf{n} = \gamma \left(\frac{1}{R_1} + \frac{1}{R_2} \right) \text{ for } \mathbf{x} \in S(t) \tag{2.12}$$

where R_1 and R_2 are the principal radii of curvature of the drop surface $S(t)$ and γ is surface tension.

Simplifying Assumptions

Dimensional Analysis

We begin by nondimensionalizing the equations of motion, following the same approach as that of Lin [26] and Falcon [1]. For our reference density, we define $\rho_{ref} = \frac{\rho_1 + \rho_2}{2}$, where ρ_1 and ρ_2 are the densities of the bottom and top ambient fluid layers, respectively. Dynamic viscosity is constant throughout the ambient fluid, so we simply define the reference dynamic viscosity to be μ, with associated kinematic viscosity scale $\nu_{ref} = \frac{\mu}{\rho_{ref}}$ (as it would otherwise vary along with the fluid density). Our reference length is a, the radius of a spherical drop of equivalent volume, and the reference velocity is U_{ref}, the terminal velocity of the drop in homogeneous fluid of density ρ_{ref}. Finally, our time scale is T, the deceleration time from terminal velocity of the drop in the bottom layer to the minimum velocity attained, and g is gravity.

Nondimensionalizing the equations of motion with the above characteristic scales, we find that (2.1) becomes

$$ReSt\tilde{\rho}\frac{\partial \tilde{\mathbf{v}}}{\partial \tilde{t}} + Re\tilde{\rho}\tilde{\mathbf{v}} \cdot \tilde{\nabla}\tilde{\mathbf{v}} = \frac{Re}{Fr^2}\tilde{\rho}\mathbf{k} - \tilde{\nabla}\tilde{p} + \tilde{\nabla}^2\tilde{\mathbf{v}}, \tag{2.13}$$

where the Reynolds, Strouhal, and Froude numbers are defined to be $Re = aU\rho/\mu$, $St = A/UT$,and $Fr = U/\sqrt{ga}$, respectively. From the experimental work detailed in Ch. 4, typical values

of our length, time, density, viscosity, and velocity scales are

$$\begin{aligned} \rho_{ref} &\approx 1.25 \text{ g/cm}^3 \\ \mu &\approx 340 \text{ mPa.s} \\ U &\approx .15 \text{ cm/s} \\ T &\approx 10 \text{ s} \\ a &\approx .06 \text{ cm} \end{aligned}$$

Thus the Reynolds number of our experimental setup is $Re \approx 3 \times 10^{-3}$, which is consistent with the Stokes regime.

So, we may instead use the Stokes equations inside the drop and Stokes equations with variable density outside the drop

$$\nabla \hat{p} = \hat{\mu} \nabla^2 \hat{\mathbf{u}} \tag{2.14}$$

$$\nabla \cdot \hat{\mathbf{u}} = 0 \tag{2.15}$$

$$\nabla p - \rho(\mathbf{x}, t)\mathbf{g} = \mu \nabla^2 \mathbf{u} \tag{2.16}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2.17}$$

The linearity of the Stokes equations greatly simplifies the following work.

Applying the same nondimensionalization to (2.7) allows us to scale out the dV/dt term, leaving us with

$$0 = m_d g + \int_S \sigma \cdot \mathbf{n} dS. \tag{2.18}$$

Details of this analysis are reported in Lin's dissertation [26]

Drop Shape

As discussed in Appendix B, we assume that the surface tension of the drop is high enough to keep the drop approximately spherical. That is, the capillary number $Ca = \frac{\mu U}{\sigma} << 1$, where the

capillary number represents the ratio of viscous forces to surface tension forces between the drop and the ambient fluid. This has the additonal benefit of reducing the the normal stress balance (2.12) to the condition that the drop remain perfectly spherical. Full details of this calculation, which is based upon the work of Brignell [15], Taylor, and Acrivos [14], in addition to an estimate of the approximate leading-order correction to the shape of the droplet, may be found in Appendix C.

Separation of Flow into "Stokes" and "Perturbation" Flows

Following an approach similar to that of [6], [26], and [1], we now exploit the linearity of the Stokes equations to separate the flow into flow in a cylinder with a static density distribution, and flow driven by the change in the density distribution.

Consider the second term of the left-hand side of (2.16). We may separate it into static and time-varying components as follows

$$\mathbf{g}\rho(\mathbf{x},t) \equiv \mathbf{g}(\rho(\mathbf{x},t) - \rho_0(z)) + \mathbf{g}\rho_0(z) \tag{2.19}$$

where ρ_0 is as in (2.6).

Now break the velocity $\mathbf{v}$ into flow with static density distribution, which we refer to as "Stokes velocity" $\mathbf{u}$, and flow driven by the changing density distribution, which we refer to as the "perturbation velocity" $\mathbf{w}$, in keeping with the work of [6], so that

$$\begin{aligned} \mathbf{v} &= \mathbf{u} + \mathbf{w} \\ p &= p_u + p_w, \end{aligned}$$

and we have the following equations for $\mathbf{u}$ and $\mathbf{w}$ outside the drop:

$$\begin{aligned} \mu\nabla^2\mathbf{u} &= \nabla p_u + g\mathbf{z}\rho_o(z) && (2.20) \\ \nabla\cdot\mathbf{u} &= 0 \\ \mu\nabla^2\mathbf{w} &= \nabla p_w + g\mathbf{z}(\rho(\mathbf{x},t) - \rho_0(z)) && (2.21) \\ \nabla\cdot\mathbf{w} &= 0. \end{aligned}$$

Inside the drop there is no density variation, so we simply write

$$\hat{\mathbf{v}} = \hat{\mathbf{u}}. \tag{2.22}$$

At the drop surface, with the normal stress condition reduced to the assumption that the drop remains a sphere, the boundary conditions (2.10) and (2.11) become

$$\mathbf{u} = \hat{\mathbf{u}} \text{ at } r = a \tag{2.23}$$

$$\mathbf{w} = 0 \text{ at } r = a \tag{2.24}$$

$$\tag{2.25}$$

and

$$\boldsymbol{\tau} \cdot (\nabla(\mathbf{u}+\mathbf{w}) + (\nabla(\mathbf{u}+\mathbf{w}))^T) \cdot \mathbf{n} = \boldsymbol{\tau} \cdot (\nabla\hat{\mathbf{u}} + (\nabla\hat{\mathbf{u}})^T) \cdot \mathbf{n} \text{ at } r = a \tag{2.26}$$

Finally, for the boundary conditions at the cylinder wall and at infinity, we have

$$\mathbf{u} + \mathbf{w} = 0 \text{ for } \sqrt{x_1^2 + x_2^2} = R_0, -\infty < x_3 < \infty, \tag{2.27}$$

and

$$\mathbf{u} \to 0 \text{ as } |x_3| \to \infty \tag{2.28}$$

$$\mathbf{w} \to 0 \text{ as } |x_3| \to \infty \tag{2.29}$$

Unlike the solid sphere problem, we cannot completely decouple the system for perturbation flow from the system for Stokes flow. Instead, we use it to pose the tangential stress condition (2.26) on $\mathbf{u}$ and $\hat{\mathbf{u}}$. We now solve for $\mathbf{u}$ and $\mathbf{w}$ separately.

Perturbation Velocity

The perturbation velocity used for this model is the same as in the case of the solid sphere studied by Camassa et al [6]. From (2.21) and letting

$$G(\mathbf{x}, t) = (\rho(\mathbf{x}, t) - \rho_0(x_3)) \tag{2.30}$$

we have the governing equations

$$\begin{aligned} \mu\nabla^2\mathbf{w} &= \nabla p_w + gG(\mathbf{x}, t) \qquad (2.31) \\ \nabla \cdot \mathbf{w} &= 0 \end{aligned}$$

with boundary conditions

$$\begin{aligned} \mathbf{w} &= 0 \text{ for } |\mathbf{x}| = a \qquad (2.32) \\ \mathbf{w} &= 0 \text{ for } \sqrt{x_1^2 + x_2^2} = R_0, \qquad (2.33) \\ \mathbf{w} &\to 0 \text{ for } |x_3| \to \infty \qquad (2.34) \\ & \qquad (2.35) \end{aligned}$$

Disregarding the condition on the cylinder wall, we may approximately solve this by convolving the Green's function $W_j(\mathbf{x}, \mathbf{y})$ due to Oseen [11] with the characteristic function $G(\mathbf{y}, t)$.

$$w_k(\mathbf{x}, t) = \int_{\Omega_f} G(\mathbf{y}, t) W_k(\mathbf{x}, \mathbf{y}) d_3 y, \tag{2.36}$$

where Ω_f represents the fluid domain external to the sphere of radius a. The full expressions for the Oseen tensor and discussion thereof may be found in Appendix A. It is important to note that the error this flow incurs on the cylinder wall is $O((\rho_2 - \rho_1)a/R_0)$ [26].

Stokes Velocity

Overall Approach

Consider (2.20). We may rewrite this using

$$\nabla\tilde{p} = \nabla p_u - g\mathbf{z}\rho_0(z) \Rightarrow \tilde{p} = p_u - \int^z g\rho_0(s)ds,$$

so that (2.20) becomes

$$\mu\nabla^2\mathbf{u} = \nabla\tilde{p}$$

The Stokes velocities $\mathbf{u}$ and $\hat{\mathbf{u}}$ satisfy equations of motion (2.20), with kinematic and dynamic conditions at the drop surface

$$\mathbf{u} = \hat{\mathbf{u}} \text{ at } r = a \tag{2.37}$$

and

$$\tau \cdot (\nabla(\mathbf{u}+\mathbf{w}) + (\nabla(\mathbf{u}+\mathbf{w}))^T) \cdot \mathbf{n} = \tau \cdot (\nabla\hat{\mathbf{u}} + (\nabla\hat{\mathbf{u}})^T) \cdot \mathbf{n}. \tag{2.38}$$

with boundary conditions

$$\mathbf{u} \to 0 \text{ as } x_3 \to \infty \tag{2.39}$$

and

$$\mathbf{u} = 0 \text{ at } \sqrt{x_1^2 + x_2^2} = R_0 \tag{2.40}$$

We begin by considering the case of a droplet in infinite ambient fluid using a streamfunction approach similar to that used by Haberman [21], then correct for the boundary condition on the cylinder wall using the Method of Reflections in a manner similar to that used by Happel and Byrne [20]. The adaptations of the Haberman solution and the Method of Reflections to our problem constitute the first original results presented in this dissertation.

Model in Infinite Fluid

The following work is conducted in the frame of reference of the drop, in spherical coordinates (r, φ, θ). (See Fig 2.1) In order to solve this system, we introduce the axisymmetric Stokes

streamfunctions $\hat{\psi}$ and ψ inside and outside the drop, respectively, so that

$$u_r = \frac{1}{r^2 \sin(\theta)} \frac{\partial \psi}{\partial \theta} \tag{2.41}$$

$$u_\theta = \frac{-1}{r \sin(\theta)} \frac{\partial \psi}{\partial r} \tag{2.42}$$

Additionally, tangential and normal stresses may be computed as

$$\tau_{r\theta} = \mu r \frac{\partial}{\partial r}(\frac{u_\theta}{r}) + \frac{\mu}{r} \frac{\partial}{\partial \theta}(u_r) \tag{2.43}$$

$$\tau_{rr} = -p + 2\mu \frac{\partial u_r}{\partial r}. \tag{2.44}$$

For infinite fluid, our boundary condition at the cylinder wall is omitted, and our condition that the ambient fluid be quiescent at infinity (2.39) becomes

$$\mathbf{u} \to -\mathbf{V}(t) \text{ as } r \to \infty \tag{2.45}$$

that is, the velocity field is a uniform stream at infinity from the drop frame of reference.

In spherical coordinates assuming axisymmetric flow, and in terms of the streamfunction, the Stokes equations become

$$[\frac{\partial^2}{\partial r^2} + \frac{1 - \cos^2(\theta)}{r^2} \frac{\partial^2}{\partial (\cos(\theta))^2}]^2 \psi(r, \theta) = 0$$

Following the approach of Haberman [21], the following general solutions for streamfunctions outside and inside the drop may be obtained

$$\psi(r, \theta) = \sum_{n=2}^{\infty} C_n^{-1/2}(\cos(\theta))(A_n r^n + B_n \frac{1}{r^{n-1}} + C_n r^{n+2} + D_n \frac{1}{r^{n-3}}) \tag{2.46}$$

$$\hat{\psi}(r, \theta) = \sum_{n=2}^{\infty} C_n^{-1/2}(\cos(\theta))(E_n r^n + F_n r^{n+2}) \tag{2.47}$$

for $n = 2, 4, 6, ...$, where $C_n^{-1/2}$ denotes the Gegenbauer polynomial of order n and degree $-1/2$.

From these streamfunctions, using (2.41)-(2.43), in addition to the identities

$$\frac{\partial}{\partial\theta}[C_n^{-1/2}(\cos(\theta))] = P_{n-1}(\cos(\theta))\sin(\theta) \tag{2.48}$$

$$\frac{\partial}{\partial\theta}[P_{n-1}(\cos(\theta))] = \frac{-n(n-1)}{\sin(\theta)}C_n^{-1/2}(\cos(\theta)) \tag{2.49}$$

we may obtain the following velocities and stresses inside and outside the drop:

$$\begin{aligned}
\hat{u}_r &= -\sum_{n=2}^{\infty} P_{n-1}(\cos(\theta))(E_n r^{n-2} + F_n r^n) \\
\hat{u}_\theta &= \sum_{n=2}^{\infty} \frac{C_n^{-1/2}(\cos(\theta))}{\sin(\theta)}\big(nE_n r^{n-2} + (n+2)F_n r^n\big) \\
\hat{\tau}_{r\theta} &= \hat{\mu}\sum_{n=2}^{\infty} \frac{C_n^{-1/2}(\cos(\theta))}{\sin(\theta)}\Big\{2n(n-2)E_n r^{n-3} + 2(n^2-1)F_n r^n\Big\} \\
\hat{\tau}_{rr} &= \hat{\mu}\sum_{n=2}^{\infty} P_{n-1}(\cos(\theta))\Big\{-2(n-2)E_n r^{n-3} + \left[\frac{2(2n+1)}{n-1} - 2n\right]F_n r^{n-1}\Big\} \\
u_r &= -\sum_{n=2}^{\infty} P_{n-1}(\cos(\theta))(A_n r^{n-2} + B_n\frac{1}{r^{n+1}} + C_n r^n + D_n\frac{1}{r^{n-1}}) \\
u_\theta &= \sum_{n=2}^{\infty} \frac{C_n^{-1/2}(\cos(\theta))}{\sin(\theta)}(nA_n r^{n-2} - (n-1)B_n\frac{1}{r^{n+1}} + (n+2)C_n r^n - (n-3)D_n\frac{1}{r^{n-1}}) \\
\tau_{r\theta} &= \mu\sum_{n=2}^{\infty} \frac{C_n^{-1/2}(\cos(\theta))}{\sin(\theta)}\Big\{2n(n-2)A_n r^{n-3} + 2(n^2-1)B_n\frac{1}{r^{n+2}} + 2(n^2-1)C_n r^{n-1} \\
&\quad +2n(n-2)D_n\frac{1}{r^n}\Big\} \\
\tau_{rr} &= \mu\sum_{n=2}^{\infty} P_{n-1}(\cos(\theta))\Big\{-2(n-2)A_n r^{n-3} + 2(n+1)B_n\frac{1}{r^{n+2}} + \left[\frac{2n(2n+1)}{n-1} - 2n\right]C_n r^{n-1} + \\
&\quad + \left[\frac{2(2n-3)}{n} + 2(n-1)\right]D_n\frac{1}{r^n}\Big\}
\end{aligned}$$

From the condition that velocity approach $-\mathbf{V(t)}$ as $r \to \infty$, we see that $A_2 = -V(t)$, $A_n = 0$ for $n > 2$, and $C_n = 0$ for all n.

Note that these expressions are determined only by the axisymmetry of the flow, and the fact that it must satisfy the Stokes equations. These are both properties of the perturbation flow $\mathbf{w}$ as well, and so we may assume that $\mathbf{w}$ and the stresses due to $\mathbf{w}$ may be expressed in the same form, so that

$$
\begin{aligned}
w_r &= -\sum_{n=2}^{\infty} P_{n-1}(\cos(\theta)) J_n(r) && (2.50)\\
w_\theta &= \sum_{n=2}^{\infty} \frac{C_n^{-1/2}(\cos(\theta))}{\sin(\theta)} H_n(r) && (2.51)\\
\tau_{r\theta}^{(w)} &= \mu \sum_{n=2}^{\infty} \frac{C_n^{-1/2}(\cos(\theta))}{\sin(\theta)} G_n(r) && (2.52)\\
\tau_{rr}^{(w)} &= \mu \sum_{n=2}^{\infty} P_{n-1}(\cos(\theta)) I_n(r) && (2.53)
\end{aligned}
$$

For brevity, we will write $H_n(a) = H_n$, $I_n(a) = I_n$, and so forth in the following work.

Applying the orthogonality of the Gegenbauer and Legendre polynomials, our boundary conditions at the surface of the drop (2.37) and (2.38) yield the following system of equations to be solved for the coefficients of the drop.

$$
\begin{aligned}
nE_n a^{n-2} + (n+2)F_n a^n &= \delta(n,2)n(-V)a^{n-2} - (n-1)B_n \frac{1}{a^{n+1}} - (n-3)D_n \frac{1}{a^{n-1}}\\
E_n a^{n-2} + F_n a^n &= 0\\
\delta(n,2)(-V)a^{n-2} + B_n \frac{1}{a^{n+1}} + D_n \frac{1}{a^{n-1}} &= 0\\
\hat{\mu}(2n(n-2)E_n a^{n-3} + 2(n^2-1)F_n a^{n-1}) &= \mu(2n(n-2)\delta(n,2)(-V)a^{n-3} + 2(n^2-1)B_n \frac{1}{a^{n+2}}\\
&\quad +2n(n-2)D_n \frac{1}{a^n} + G_n)
\end{aligned}
$$

This yields the solution for $n = 2$

$$
\begin{aligned}
B_2 &= -\frac{a^3(3\hat{\mu}V + a\mu G_2)}{6(\mu+\hat{\mu})}\\
D_2 &= \frac{6a\mu V + 9a\hat{\mu}V + a^2\mu G_2}{6(\mu+\hat{\mu})}\\
E_2 &= \frac{3\mu V - a\mu G_2}{6(\mu+\hat{\mu})}\\
F_2 &= \frac{-3\mu V + a\mu G_2}{6a^2(\mu+\hat{\mu})}
\end{aligned}
$$

and, for $n > 2$

$$\begin{aligned} B_n &= -\frac{a^{n+2}\mu G_n}{2(2n-1)(\mu+\hat{\mu})} \\ D_n &= \frac{a^n \mu G_n}{2(2n-1)(\mu+\hat{\mu})} \\ E_n &= -\frac{a^{3-n}\mu G_n}{2(2n-1)(\mu+\hat{\mu})} \\ F_n &= \frac{a^{1-n}\mu G_n}{2(2n-1)(\mu+\hat{\mu})} \end{aligned}$$

Observe that, in the absence of perturbation flow, this solution reduces to the Hadamard-Rybczynski solution for a drop rising in infinite quiescent fluid at zero Reynolds number [25].

$$\psi(r,\theta) = 1/2\sin^2(\theta)V(-r^2 - \frac{a^3(\hat{\mu})}{2(\mu+\hat{\mu})}\frac{1}{r} + \frac{6a\mu+9a\hat{\mu}}{6(\mu+\hat{\mu})}\frac{1}{r^{-1}}) \text{ for } n = 2,4,6,... \quad (2.54)$$

$$\hat{\psi}(r,\theta) = 1/2\sin^2(\theta)V(\frac{\mu}{2(\mu+\hat{\mu})}r^2 + \frac{-\mu}{2a^2(\mu+\hat{\mu})}r^4). \quad (2.55)$$

Drag Calculation

Guided by Haberman's approach to computing the drag on a solid sphere in a tube [21], we now compute the viscous drag on the droplet. Drag $D = D_r + D_\theta$, where

$$D_\theta = \int \mathcal{T}_{r\theta}\sin(\theta)dS, \;\; D_r = -\int \mathcal{T}_{rr}\cos(\theta)dS$$

where $dS = 2\pi a^2\sin(\theta)d\theta$, and

$$\begin{aligned} \mathcal{T}_{r\theta} &= \tau_{r\theta} + \tau_{r\theta}^{(w)} \\ \mathcal{T}_{rr} &= = \tau_{rr} + \tau_{rr}^{(w)} \end{aligned}$$

Using the facts that

$$C_n^{-1/2}(x) = \frac{1}{2n-1}(P_{n-2}(x) - P_n(x)),$$

$$\int_0^\pi P_m(\cos(\theta))\sin(\theta)d\theta = \begin{cases} 2 & \text{for } m = 2 \\ 0 & \text{otherwise} \end{cases}$$

and

$$\int_0^\pi P_m(\cos(\theta))\sin(2\theta)d\theta = \begin{cases} 0 & \text{for } m \geq 2 \text{ or } m \text{ even} \\ \frac{4}{3} & \text{for } m < 2 \text{ and } m \text{ odd} \end{cases}$$

we see that

$$\begin{aligned} D_\theta &= 2a^2\pi \int_0^\pi (\tau_{r\theta} + \tau_{r\theta}^{(w)})\sin^2(\theta)d\theta \\ &= \frac{4a\pi\mu\hat{\mu}(3V + aG_2)}{3(\mu + \hat{\mu})} \\ D_r &= -2a^2\pi \int_0^\pi (\tau_{rr} + \tau_{rr}^{(w)}\cos(\theta)\sin(\theta)d\theta \\ &= 2a\pi\mu\frac{3V(2\mu + \hat{\mu}) - 2a(\mu + \hat{\mu})I_2 + a\mu G_2}{2(\mu + \hat{\mu})} \end{aligned}$$

and so

$$D = \frac{2a\pi\mu}{3(\mu + \hat{\mu})}(V(6\mu + 9\hat{\mu}) + a(\mu(-2I_2 + G_2) + 2\hat{\mu}(-I_2 + G_2)))$$

Again, note that when there is no perturbation flow, this reduces to the Hadamard result for the drag on a drop in infinite fluid,

$$D_{HR} = \frac{2a\pi\mu}{(\mu + \hat{\mu})}V(2\mu + 3\hat{\mu}) \tag{2.56}$$

Wall Correction

We begin with the case of a drop rising through a cylinder of homogeneous fluid. Following the lead of Happel and Byrne [27], as refined by Camassa et al. [6], we use the method of reflections to find the flow around a drop rising through a cylinder. We also compute the drag force on the drop and the terminal velocity of the drop, followed with a discussion of the extension of this approach to the full problem of a drop rising through a cylinder of stratified fluid.

As before, consider a spherical drop of radius a rising due to buoyancy through a cylindrical tank of radius R_0 of homogeneous fluid. Assume that the drop has density $\hat{\rho}$ and viscosity $\hat{\mu}$, and

the ambient fluid has density ρ and viscosity μ. We will be using Cartesian coordinates (x_1, x_2, z), spherical coordinates (r, θ, φ), and cylindrical coordinates (R, φ, X), as in Fig. 2.1. We need to solve the incompressible Stokes equations within and outside the drop,

$$\begin{aligned} \hat{\mu}\nabla^2\hat{\mathbf{u}} &= \nabla\hat{p} \\ \nabla\cdot\hat{\mathbf{u}} &= 0 \\ \mu\nabla^2\mathbf{u} &= \nabla p \\ \nabla\cdot\mathbf{u} &= 0, \end{aligned}$$

enforcing a no-slip condition on the cylinder walls, and matching velocity and tangential stress on the surface of the drop:

$$\mathbf{u} = 0 \qquad \text{for } \sqrt{x_1^2 + x_2^2} = R_0 \tag{2.57}$$

$$\mathbf{u} \to 0 \qquad \text{for } |\mathbf{x}| \to \infty \tag{2.58}$$

$$\hat{\mathbf{u}} = \mathbf{u} \qquad \text{at } r = a \tag{2.59}$$

$$\boldsymbol{\tau}\cdot[\hat{\mu}\hat{\mathbf{D}} - \mu\mathbf{D}]\cdot\mathbf{n} = 0 \qquad \text{at } r = a \tag{2.60}$$

where $\mathbf{D}$ in (2.60) denotes the deviatoric stress tensor. To see how to apply the method of reflections in this case, we consider the work of Happel and Byrne [27] for the case of a solid sphere moving at velocity V through a cylinder of still Newtonian fluid. In this case, the objective was to solve the incompressible Stokes equations on the fluid domain

$$\begin{aligned} \mu\nabla^2\mathbf{u} &= \nabla p \\ \nabla\cdot\mathbf{u} &= 0 \end{aligned}$$

with a no-slip boundary condition at the surface of the sphere and on the cylinder wall,

$$\begin{aligned} \mathbf{u} = 0 \qquad & \text{at } r = a \text{ and } \sqrt{X^2 + R^2} = R_0 \\ \mathbf{u} \to 0 \qquad & \text{as } |\mathbf{x}| \to \infty \end{aligned}$$

Happel and Byrne use the method of reflections to decompose the flow into a series

$$\mathbf{u} = (\mathbf{u}^{(0)} + \mathbf{u}^{(1)}) + (\mathbf{u}^{(2)} + \mathbf{u}^{(3)}) + ..., \tag{2.61}$$

where, assuming we are in a frame of reference moving with the center of the sphere,

$$\mathbf{u}^{(0)} = -\mathbf{V} \tag{2.62}$$

$$\mathbf{u}^{(1)} = \begin{cases} -\mathbf{u}^{(0)} & \text{at } r = a \\ 0 & \text{as } r \to \infty \end{cases} \tag{2.63}$$

$$\mathbf{u}^{(2)} = \begin{cases} -\mathbf{u}^{(1)} & \text{at } R = R_0 \\ 0 & \text{as } z \to \pm\infty \end{cases} \tag{2.64}$$

$$\mathbf{u}^{(3)} = \begin{cases} -\mathbf{u}^{(2)} & \text{at } r = a \\ 0 & \text{as } r \to \infty \end{cases} \tag{2.65}$$

Consider the first few terms of this series individually. $\mathbf{u}^{(0)}$ is uniform flow; that is, the flow from the frame of reference of a point moving vertically through the cylinder at velocity $\mathbf{V}$. This term remains unchanged for the droplet case.

$\mathbf{u}^{(1)}$ (when added to $\mathbf{u}^{(0)}$) is simply the solution for Stokes flow around a sphere in infinite fluid, since it satisfies only the boundary condition on the drop. This introduces error of order a/R_0 on the cylinder wall.

$\mathbf{u}^{(2)}$ satisfies the boundary conditon on the cylinder wall by canceling out the effects of $\mathbf{u}^{(1)}$ at the cylinder wall, but introduces error of order a/R_0 on the surface of the sphere. This error on the sphere surface is in turn cancelled out by $\mathbf{u}^{(3)}$, and so on for further terms of the series.

We now adapt this approach for a spherical drop. We again seek to expand the flow outside the drop into an asymptotic series $\mathbf{u} = (\mathbf{u}^{(0)} + \mathbf{u}^{(1)}) + (\mathbf{u}^{(2)} + \mathbf{u}^{(3)}) + ...$, where, assuming we are in the

frame of reference moving with the center of the drop,

$$\mathbf{u}^{(0)} = -\mathbf{V} \tag{2.66}$$

$$\mathbf{u}^{(1)} = \begin{cases} \text{satisfies surface conditions} & \text{at } r = a \\ 0 & \text{as } r \to \infty \end{cases} \tag{2.67}$$

$$\mathbf{u}^{(2)} = \begin{cases} -\mathbf{u}^{(1)} & \text{at } R = R_0 \\ 0 & \text{as } z \to \pm\infty \end{cases} \tag{2.68}$$

$$\mathbf{u}^{(3)} = \begin{cases} \text{satisfies surface conditions} & \text{at } r = a \\ 0 & \text{as } r \to \infty \end{cases} \tag{2.69}$$

where by "surface conditions" we refer to the kinematic and dynamic boundary conditions on the drop surface, (2.59) and (2.60). Note that we need to introduce a corresponding series $\hat{\mathbf{u}} = \hat{\mathbf{u}}^{(1)} + \hat{\mathbf{u}}^{(3)} + ..$ for flow inside the drop, matching with the odd terms of the $\mathbf{u}$ series in order to satisfy the conditions at the drop surface. $\hat{\mathbf{u}}^{(1)}$ is simply the Hadamard solution for flow inside a drop, and the following terms should behave similarly.

The main difference from the case of the solid sphere is that $\mathbf{u}^{(0)} + \mathbf{u}^{(1)}$ is now the Hadamard solution for Stokes flow around a drop in infinite ambient fluid. So we have the following for $\mathbf{u}^{(0)}$ and $\mathbf{u}^{(1)}$, once things are put into terms corresponding to those used by Happel and Byrne [20], with cylindrical coordinates (R, φ, X):

$$\mathbf{u}^{(0)} = -\mathbf{V}$$

$$\mathbf{u}^{(1)} = \begin{pmatrix} V\left[\frac{-a^3\kappa}{5(1+\kappa)}\frac{1}{r^3} + \frac{3a^3\kappa}{4(1+\kappa)}\frac{X^2}{r^5} - \frac{a(2+3\kappa)}{4(1+\kappa)}\left(\frac{1}{r} + \frac{X^2}{r^3}\right)\right] \\ -VA\left[\frac{-3a^2\kappa}{4(1+\kappa)}\frac{RX}{r^5} + \frac{2+3\kappa}{4(1+\kappa)}\frac{RX}{r^3}\right] \end{pmatrix}$$

where $r = \sqrt{X^2 + R^2}$ and $\kappa = \frac{\hat{\mu}}{\mu}$ is the ratio of the inner viscosity to the outer viscosity. As in [6], we know that $\mathbf{u}^{(2)}$ will be of the form

$$u_X^{(2)} = \frac{-1}{2\pi}\int_0^\infty \left[\frac{\lambda R}{2}(H+G)I_1(\lambda R) + HI_0(\lambda R)\right]\cos(\lambda X)d\lambda \tag{2.70}$$

$$u_R^{(2)} = \frac{-1}{2\pi}\int_0^\infty \left[\frac{\lambda R}{2}(H+G)I_0(\lambda R) - GI_1(\lambda R)\right]\sin(\lambda X)d\lambda \tag{2.71}$$

where the functions H and G are independent of R and X, depending only on the parameter λ. In order to express the boundary conditions in the same coordinate system as the above, we use the Bessel function integral expression form of $1/r$

$$\frac{1}{r} = \frac{2}{\pi}\int_0^\infty K_0(\lambda R)\cos(\lambda X)d\lambda$$

and repeated partial differentiation with the identity

$$\frac{\partial}{\partial R}K_0(\lambda R) = -\lambda K_1(\lambda R)$$

to express $u_X^{(1)}$ and $u_R^{(1)}$ as

$$u_X^{(1)} = \frac{2}{\pi}\int_0^\infty M\cos(\lambda X)d\lambda$$

$$u_R^{(1)} = \frac{2}{\pi}\int_0^\infty N\sin(\lambda X)d\lambda$$

where

$$M = \frac{-2aV(2+3\kappa)}{4(1+\kappa)}K_0(\lambda R) + \frac{aV(2+3\kappa)}{4(1+\kappa)}\lambda R K_1(\lambda R)$$

$$N = \frac{-Va(2+3\kappa)}{4(1+\kappa)}\lambda R K_0(\lambda R) + \frac{Va^3\kappa}{4(1+\kappa)}\lambda^2 K_1(\lambda R)$$

By applying the boundary equations on $\mathbf{u}^{(2)}$ from (2.66) at the cylinder wall $R = R_0$, we obtain the following system of equations:

$$
\begin{aligned}
4[M]_{R=R_0} &= \frac{\lambda R_0}{2}(H+G)I_1(\lambda R_0) + HI_0(\lambda R_0) \\
4[N]_{R=R_0} &= \frac{-\lambda R_0}{2}(H+G)I_0(\lambda R_0) - GI_1(\lambda R_0)
\end{aligned}
$$

We then solve for H and G, to obtain

$$
\begin{aligned}
H &= ((aV(R_0\lambda I_0(-(4+6\kappa+a^2\kappa\lambda^2)K_0 + R_0(2+3\kappa)\lambda K_1) + I_1((8+12\kappa+(2a^2\kappa + \\
&\quad R_0^2(2+3\kappa))\lambda^2)K_0 - R_0\lambda(4+6\kappa+a^2\kappa\lambda^2)K_1))))/((1+\kappa)(R_0\lambda I_0^2 - 2I_0I_1 - R_0\lambda I_1^2)) \\
G &= \frac{aV\lambda^2(I_0(a^2R_0\kappa\lambda K_0 + (2a^2\kappa - R_0^2(2+3\kappa))K_1) + R_0I_1(-R_0(2+3\kappa)K_0 + a^2\kappa\lambda K_1))}{(1+\kappa)(R_0\lambda I_0^2 - 2I_0I_1 - R_0\lambda I_1^2)}
\end{aligned}
$$

where

$$
\begin{aligned}
I_0 &= I_0(R_0\lambda) \\
I_1 &= I_1(R_0\lambda) \\
K_0 &= K_0(iR_0\lambda) \\
K_1 &= K_1(iR_0\lambda)
\end{aligned}
$$

Note that the sum of these yields a velocity field which satisfies the boundary condition on the cylinder wall, but not the condition on the surface of the drop. Additional corrections $\mathbf{u}^{(3)}$ and $\hat{\mathbf{u}}^{(3)}$ to both the internal and external flows would be required to satisfy the boundary conditions on the drop, though this would introduce error at the cylinder wall.

We may approximate the third reflection as follows. For drops which are small compared to the size of the cylinder, $\mathbf{u}^{(2)}$ is approximately uniform flow near the drop. So we consider an expansion of $\mathbf{u}^{(2)}$ about the origin, and take the first (constant) term to compute the drag correction on the drop due to the cylinder wall.

Since $\mathbf{u}^{(2)}$ is essentially uniform flow in the neighborhood of the drop, we can compute $\mathbf{u}^{(3)}$ in the same manner as $\mathbf{u}^{(1)}$, only with $\mathbf{u}^{(2)}(0,0)$ in place of V. That is, $\mathbf{u}^{(2)}(0,0) + \mathbf{u}^{(3)}$ in streamfunction form is simply the Hadamard solution (2.54) with $\mathbf{u}^{(2)}(0,0)$ in place of $\mathbf{V}$. So the drag exerted

on the sphere by this portion of the flow should have the same form as the expression derived by Hadamard and Rybczynski [9, 10] for a spherical drop in infinite ambient fluid,

$$4\pi a\mu v^{(2)}(0,0)\frac{\mu+\frac{3}{2}\hat{\mu}}{\mu+\hat{\mu}}$$

Setting the sum of the drag forces equal to the buoyancy force, we have

$$\frac{4}{3}\pi a^3(\hat{\rho}-\rho)g = 4\pi a\mu V\frac{\mu+\frac{3}{2}\hat{\mu}}{\mu+\hat{\mu}} + 4\pi a\mu v^{(2)}(0,0)\frac{\mu+\frac{3}{2}\hat{\mu}}{\mu+\hat{\mu}}$$

which is solved to obtain the terminal velocity of the drop in a cylinder of homogeneous fluid:

$$\begin{aligned} V &= (0.321446a^2gR_0^3(\mu+\hat{\mu})^2(\rho-\hat{\rho}))/ \\ &\quad ((\mu(a^3\hat{\mu}(\mu+1.5\hat{\mu}) + aR_0^2(-1.314736\mu^2 - 3.944209\mu\hat{\mu} - 2.958157\hat{\mu}^2) + \\ &\quad R_0^3(-0.964337\mu^2 - 2.410844\mu\hat{\mu} - 1.446506\hat{\mu}^2)))) \end{aligned}$$

This is in good agreement with the terminal velocities observed in experiments with a drop rising in a cylindrical tank of homogeneous fluid, as well as with the result obtained by Haberman [21]. Indeed, we could have also used Haberman's approach to find a wall correction on the drag; this is included in Appendix 2.4.3.

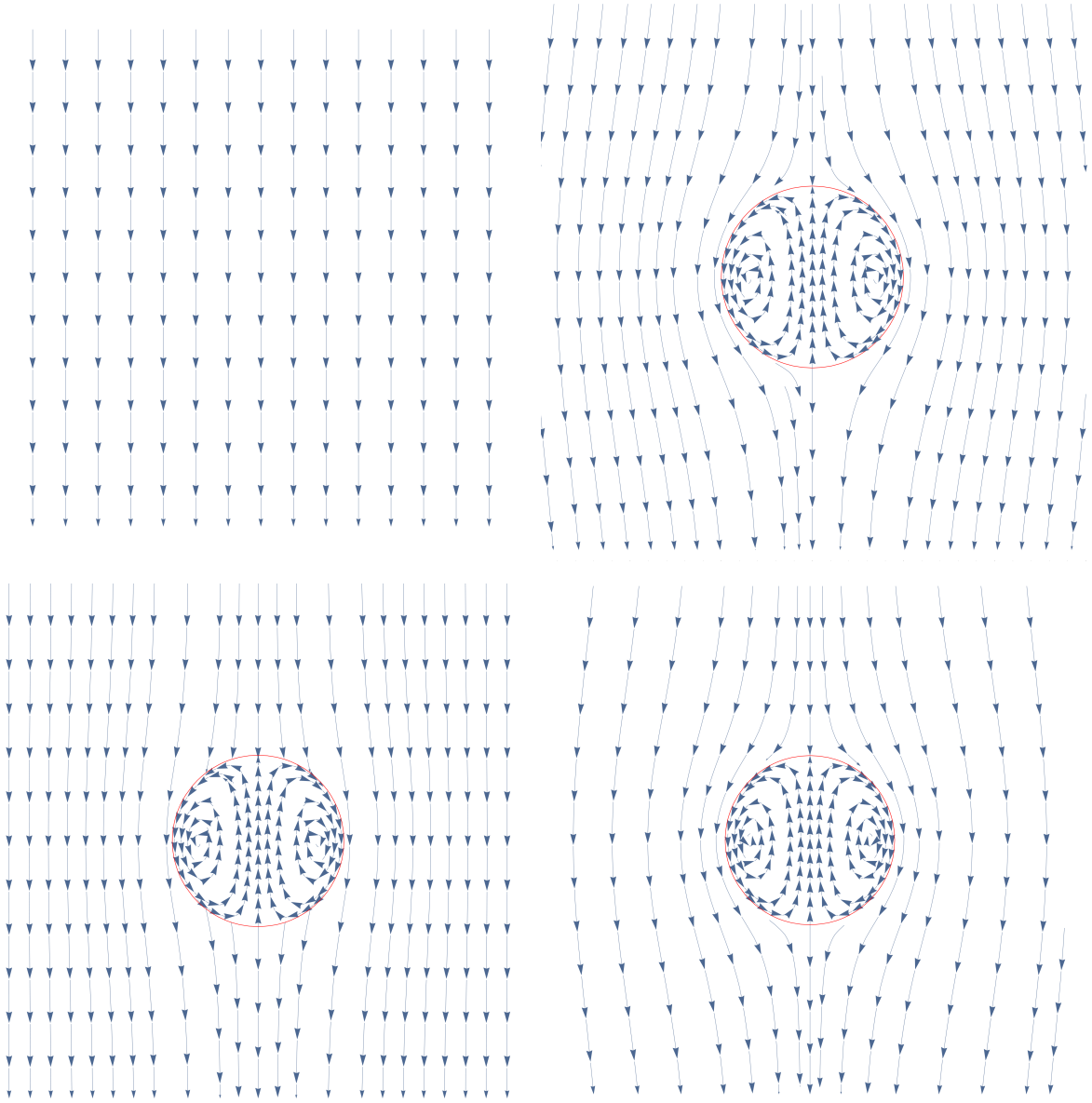

Figure 2.2: Streamlines for Method of Reflections adapted for the droplet boundary conditions. Reading left-to-right from the top left: $\mathbf{u}^{(0)}$, $\mathbf{u}^{(0)}+\mathbf{u}^{(1)}$, $\mathbf{u}^{(0)}+\mathbf{u}^{(1)}+\mathbf{u}^{(2)}$, $\mathbf{u}^{(3)}$, $\mathbf{u}^{(0)}+\mathbf{u}^{(1)}+\mathbf{u}^{(2)}+\mathbf{u}^{(3)}$. Note the alternation between satisfying the boundary conditions at the drop surface and the boundary conditions at the cylinder wall.

Final Equations of Motion

We now combine the results of the preceding sections in order to obtain the final equations of motion for the drop, which will be solved using numerical integration at each timestep as detailed in the following chapter. First, consider (2.7). Combining this with (2.4.3) and wall correction drag

(2.4.4), we find that

$$0 = m_d g - g\int_{\Omega_d} \rho_0 d\Omega - \frac{4a\pi\mu(\mu+\frac{3}{2}\hat{\mu})}{\mu+\hat{\mu}}\left(V + v^{(2)}(0,0)\right) - \frac{2a^2\pi\mu^2 G_2}{3(\mu+\hat{\mu})} - f_w$$

where

$$v^{(2)}(0,0) \ = \frac{(1.363357\mu + 2.045036\hat{\mu})V}{(\mu+\hat{\mu})}\frac{a}{R_0} - \frac{(1.036981V\hat{\mu})}{(\mu+\hat{\mu})}(\frac{a}{R_0})^3 + ... = VK$$

retaining terms to order $(a/R_0)^3$.

Note that the first two terms of this are simply the buoyancy force, the second and third term comprise the drag due to the Stokes velocity, and the final term is the density anomaly force, or the force on the drop due to the perturbation velocity. This final term is the same as in the case of the solid sphere, and is described in detail in Appendix A.

$$\begin{aligned} V(t;\rho) &= \left(-(\frac{4\pi}{3}\hat{\rho} - \int_{\Omega_s}\rho_0 d\Omega_s) - \frac{2a^2\pi\mu}{3(\mu+\hat{\mu})}(\mu(-2I_2+G_2) + 2\hat{\mu}(-I_2+G_2))\right)/ \\ &\quad (\frac{2a\pi\mu}{3(\mu+\hat{\mu})}(6\mu+9\hat{\mu}) + 4a\pi\mu\frac{\mu+3/2\mu}{\mu+\hat{\mu}}(v^{(2)}(0,0))) \\ &= (\mu+\hat{\mu})\left\{(m_d g - g\int_{\Omega_d}\rho_0 d\Omega) - (f_w + \frac{2a^2\pi\mu^2 G_2}{3(\mu+\hat{\mu})})\right\}/\left(4a\pi\mu(\mu+\frac{3}{2}\hat{\mu})(1+K)\right) \end{aligned} \tag{2.72}$$

and

$$\frac{\partial\rho(\mathbf{x},t)}{\partial t} + (\mathbf{u}(\mathbf{x},t;V,\rho) + \mathbf{w}(\mathbf{x},t;\rho))\cdot\nabla\rho = 0 \tag{2.73}$$

These equations comprise our final mathematical model for an oil droplet rising through ambient fluid with a sharp two-layer density distribution. Note that, in (2.72), if we take the limit as $\hat{\mu}\to\infty$, we have the expression for the velocity in the solid sphere case; the limit of $R_0\to\infty$ returns the free-space drop velocity, and in the case of homogeneous ambient density we simply have the velocity of a drop in a cylinder.

This is a coupled pair of integro-differential equations wherein the density field $\rho(\mathbf{x},t)$ determines the domain of integration for $\mathbf{w}$ (and thus also G_2 and I_2), and $\mathbf{u}$ is determined by $V(t)$ together

with G_2 (and therefore $\rho(\mathbf{x}, t)$ as well). The initial density function $\rho(\mathbf{x}, 0)$ completely determines the future advection of the density field by the fluid velocities, and satisfied the boundary conditions specified to within the accuracy of analytic approximations described in the derivation above. The following chapter details the numerical solution of this system.

CHAPTER 3

Numerical Implementation

This chapter details the numerical implementation of the mathematical model given by (2.73) and (2.72). Code and calculation verification are also addressed by means of comparison with analytic benchmarks and a grid convergence study where possible.

The overall approach is very similar to the work of Lin [26] and Falcon [1], and the implementation is, in fact, developed from a code base inherited from these projects. However, this model introduces a number of new matters of consideration. Most notably, it requires the computation of the coefficients G_2 and I_2 of the Gegenbauer and Legendre polynomial expansions of the stresses due to $\mathbf{w}$. Additionally, the hyperbolic stagnation point at the trailing end of the droplet necessitated the development of a new technique for the addition of interpolated points along the density transition interface, informed by the particular conditions of this problem. The Fortran code for this implementation may be found in Appendix E.

Overview

This program simulates the advection of the density distribution of the ambient fluid within a cylindrical tank by the velocity field $\mathbf{v} = \mathbf{u} + \mathbf{w}$. Because our model assumes a sharp two-layer density distribution, this distribution may be completely described by the material surface defining the interface between the two regions of different density. Additionally, because we assume that axisymmetry is preserved throughout our simulation, this material surface may be fully characterized by its intersection with half of a vertical cross section of the cylindrical tank. To track the evolution of the curve defining this intersection, Lagrangian marker points are placed along it. The position of these points in the frame of reference moving with the center of the drop fully characterizes the state of the system at a given timestep.

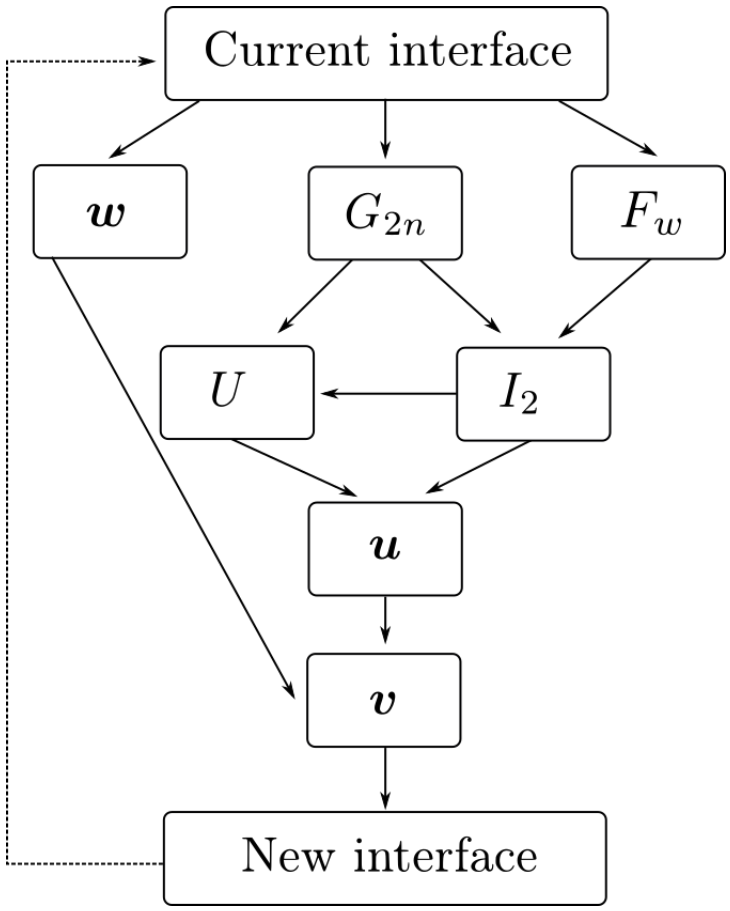

Figure 3.1: Schematic of computation of velocity **v** at each timestep. Once **v** is computed, it is used to advect the interface and determine the new interface for the next timestep.

The program is intialized with the parameters describing the geometric and fluid properties of the system, and the initial ambient density distribution as given by the marker points described in the preceding paragraph. At each timestep, the current density distribution is used to compute **w**, G_2, and I_2, from which the velocity of the drop $V(t)$ is computed using formula (2.72). Then the Stokes velocity **u** is evaluated using G_2 and $V(t)$. Finally, the points tracking the density transition interface are advected using these velocities. New marker points are then added to fill any excessively large gaps in the interface points, and the above steps are repeated for the new interface. Each part of this process is detailed in the following sections.

Perturbation Velocity

The evaluation of the perturbation velocity at each of the tracked interface points is treated in the same way as in [1]. The full details of this implementation may be found in [1]; a synopsis

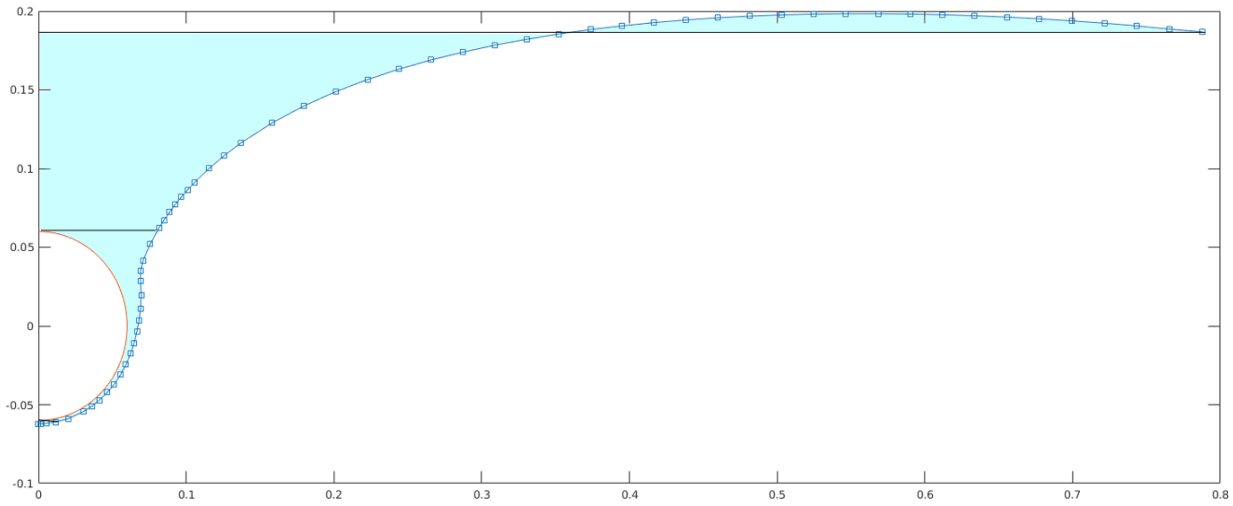

Figure 3.2: The shaded area is the region of integration for $\mathbf{w}$, G_2 and perturbation force integrals. The marker points used to track the density transition are shown, as well as the dissection of the region for numerical integration.

follows.

Recall the definition of $\mathbf{w}$ as a convolution over the external fluid domain (2.30). Because the characteristic function G is zero everywhere except within the region of displaced fluid, we need only integrate over this region. The region of integration is show in Fig. 3.2.

This is by far the most computationally expensive portion of the code. To speed the evaluation of $\mathbf{w}$, the "far field" theory approximation of $\mathbf{w}$ due to Lin [26] is made for regions distant from the drop, and many of the terms of the integrand are analytically integrated once in θ first in order to speed up the calculation, as in Falcon's dissertation [1]. More details may be found in Appendix A.

Gegenbauer expansion of w stresses at drop surface

The computation of the coefficients of the Gegenbauer polynomial expansion of the tangential stress from (2.50) at the surface of the drop constitutes the most significant difference between the implementation of [1] and this code. It was also the most difficult portion of this model to implement. Three different approaches were taken to this; we will refer to these as the "numerical derivative" approach, the "analytic derivative" approach, and the "reciprocal theorem" approach. Only the final "reciprocal theorem" approach yields valid results throughout the full runtime of the code, but the other two approaches were used to verify the output of the final method and hence are described here as well.

Evaluation of G_n

In this section we discuss the more direct approaches to the evaluation of G_2, using first numerical and then analytic derivatives of $\mathbf{w}$. These approaches were ultimately unsuccessful, but provide grounds for comparison with our final method.

In order to compute the representation (2.50) of the $\mathbf{w}$ tangential stress at the drop surface, consider the expression for tangential stress evaluated on the drop surface in spherical coordinates:

$$\tau_{r\theta}^{(w)} = \mu r \frac{\partial}{\partial r}\left(\frac{w_\theta}{r}\right) + \frac{\mu}{r}\frac{\partial}{\partial \theta}(w_r)$$

Because $w_r = 0$ at $r = a$ for all θ, the second term vanishes. Then, as previously discussed, we may assume that w_θ can be expressed in the form

$$w_\theta = \sum_{n=2}^{\infty} \frac{C_n^{-1/2}(\cos(\theta))}{\sin(\theta)} H_n(r)$$

Note also that $H_n(a) = 0$ due to the no-slip boundary condition on $\mathbf{w}$ at $r = a$.

So, we have

$$\begin{aligned}
\tau_{r\theta}^{(w)} &= \mu r \frac{\partial}{\partial r}\left(\frac{1}{r}\sum_{n=2}^{\infty} \frac{C_n^{-1/2}(\cos(\theta))}{\sin(\theta)} H_n(r)\right) \\
&= \mu r \sum_{n=2}^{\infty} \frac{C_n^{-1/2}(\cos(\theta))}{\sin(\theta)} \frac{\partial}{\partial r}\left(\frac{H_n(r)}{r}\right) \\
&= \mu \sum_{n=2}^{\infty} \frac{C_n^{-1/2}(\cos(\theta))}{\sin(\theta)} \left(\frac{-1}{r} H_n(r) + H_n'(r)\right)
\end{aligned}$$

Thus, upon comparison with (2.50), it is clear that

$$G_n(a) = H_n'(a)$$

where, using the orthogonality of the Gegenbauer polynomials, we have

$$H_n(r) = \frac{1}{N(n)} \int_0^\pi w_\theta(r,\theta) C_n^{-1/2}(cos(\theta)) d\theta, \tag{3.1}$$

where the normalization $N(n)$ is given by

$$N(n) = \frac{4\pi\Gamma(n-1)}{n!(n-1/2)\Gamma(-1/2)^2} = \frac{4\pi(n-2)!}{n!(n-1/2)4\pi} = \frac{1}{n(n-1)(n-1/2)}$$

and w_θ may be evaluated from the current expression of $\mathbf{w}$ in Cartesian coordinates as

$$w_\theta = \cos(\theta)w_R - \sin(\theta)w_Z$$

Putting all of this together, we have that, at $r = a$, the coefficients G_n of tangential stress due to the perturbation velocity are

$$G_n = n(n-1)(n-1/2)\int_0^\pi C_n^{-1/2}(\cos(\theta))\frac{\partial}{\partial r}(\cos(\theta)w_R(r,\theta) - \sin(\theta)w_Z(r,\theta))|_{r=a}d\theta \tag{3.2}$$

The r-derivative in the above expression may be evaluated either numerically or analytically. In the case of numerical differentiation, the dramatic decrease in the accuracy of the numerical integration used to compute $\mathbf{w}$ for observation points inside the domain of integration and close to the boundary of the drop causes this method to fail immediately upon the passage of the sphere into the region of integration.

Analytically differentating the Oseen Green's function for w_θ before integrating over the displaced fluid region works out somewhat better, but the error is still roughly $O(1)$ once the drop passes into the region of integration.

With this in mind, we sought a method of evaluating G_2 that does not require the direct evaluation of $\mathbf{w}$ or its derivative at observation points close to the surface of the drop. In order to accomplish this, we used the generalized Lorentz reciprocal therorem, in an approach much like that used by Camassa et al. [6] (see Appendix A for this result), to compute the tangential force on the sphere without having to directly deal with the Oseen Green's function or its derivatives. This result was much more well-behaved numerically, and is in good agreement with the other two approaches within their regions of validity; that is, when the drop is well below the region of integration shown in Fig. 3.2.

The derivation of this approach may be found in the following section.

Reciprocal Theorem Evaluation of G_2

From a calculation very similar to that used to derive 2.4.3, the total vertical component of the drag force on the sphere in terms of G_2 and I_2 is

$$\frac{4}{3}\pi a^2 \mu (G_2 - I_2), \tag{3.3}$$

and we know that the component of the total vertical force on the sphere contributed by the tangential stresses due to $\mathbf{w}$ is

$$D_\theta = \int_{\partial S} \tau_{r\theta}^{(\mathbf{w})} \sin(\theta) dS = \frac{4}{3} a^2 \pi \mu G_2 = \int_{\partial S} \boldsymbol{\tau} \cdot \sigma_{\mathbf{w}} \cdot \mathbf{n} dS \tag{3.4}$$

where $\boldsymbol{\tau}$ is any unit tangent vector to the sphere surface, $\mathbf{n}$ is the outward-facing unit normal vector to the sphere surface, and $\sigma_{\mathbf{w}}$ is the deviatoric stress tensor for $\mathbf{w}$.

Inspired by the approach of Camassa et al. [6], we use the reciprocal theorem in order to avoid needing to directly evaluate any of the stresses due to $\mathbf{w}$.

We begin by constructing a velocity field $\boldsymbol{\tau}$, with associated stress tensor σ_τ, which satisfies the Stokes equations (2.20) and which is a unit tangent vector to the sphere surface when $r = a$. That is, we want $\boldsymbol{\tau}$ to satisfy the Stokes equations (2.20) together with boundary conditions

$$\tau_r = 0 \text{ when } r = a \tag{3.5}$$

$$\tau_\theta = 1 \text{ when } r = a \tag{3.6}$$

$$\boldsymbol{\tau} \to 0 \text{ as } r \to \infty \tag{3.7}$$

We use the same streamfunction approach as for the Stokes velocity, and so have

$$\psi_\tau = \sum_{n=2}^{\infty} C_n^{-1/2}(\cos(\theta))(A_n r^n + B_n \frac{1}{r^{n-1}} + C_n r^{n+2} + D_n \frac{1}{r^{n-3}}) \tag{3.8}$$

Expanding $\tau_\theta(a, \theta) = 1$ in the Gegenbauer polynomial basis in order to impose the boundary condition at $r = a$, we have

$$\tau_\theta(a, \theta) = \sum_{n=2}^{\infty} \frac{C_n^{-1/2}(\cos(\theta))}{\sin(\theta)} T_n = 1 \tag{3.9}$$

where, as previously studied in Section 3.3.1

$$T_n = n(n-1)(n-1/2)\int_0^\pi C_n^{-1/2}(\cos(\theta))d\theta \tag{3.10}$$

Applying the boundary conditions at ∞, we find that $A_n = 0$ and $C_n - 0$, and the boundary conditions at the surface of the sphere yield the following system:

$$\begin{aligned} B_n\frac{1}{a^{n+1}} + D_n\frac{1}{a^{n-1}} &= 0 && (3.11)\\ -(n-1)B_n\frac{1}{a^{n+1}} - (n-3)D_n\frac{1}{a^{n-1}} &= T_n && (3.12)\\ && (3.13) \end{aligned}$$

which has solution

$$\begin{aligned} B_n &= \frac{1}{2}a^{n+1}T_n && (3.14)\\ D_n &= \frac{-1}{2}a^{n-1}T_n && (3.15)\\ && (3.16) \end{aligned}$$

So we have a flow field that solves Stokes' equations, vanishes at ∞, and is the tangent vector on the sphere surface up to arbitrary accuracy,

$$\tau_r = -\sum_{n=2}^{\infty} P_{n-1}(\cos(\theta))\frac{T_n}{2}\big(\frac{a^{n+1}}{r^{n+1}} - \frac{a^{n-1}}{r^{n-1}}\big) \tag{3.17}$$

$$\tau_\theta = \sum_{n=2}^{\infty} \frac{C_n^{-1/2}(\cos(\theta))}{\sin(\theta)}\frac{T_n}{2}\big(-(n-1)\frac{a^{n+1}}{r^{n+1}} + (n-3)\frac{a^{n-1}}{r^{n-1}}\big) \tag{3.18}$$

$$\tag{3.19}$$

where $n = 2, 4, 6, ...$

The first ten or so terms of this expansion should provide a reasonable approximation to the tangent vector to the sphere at $r = a$.

We now use this special solution to Stokes' equations to indirectly compute G_2. Note first that

$$\nabla \cdot \boldsymbol{\tau} = 0 \tag{3.20}$$

$$\nabla \cdot \sigma_\tau = 0 \tag{3.21}$$

$$\tag{3.22}$$

The perturbation velocity, as described in Section 2.3, satisfies the following:

$$\nabla \cdot \mathbf{w} = 0 \tag{3.23}$$

$$\nabla \cdot \sigma_{\mathbf{w}} = G(\mathbf{x}, t) \tag{3.24}$$

$$\mathbf{w} \rightarrow 0 \text{ as } |\mathbf{x}| \rightarrow \infty \tag{3.25}$$

$$\mathbf{w} = 0 \text{ on } \delta S \tag{3.26}$$

$$\tag{3.27}$$

where $\mathbf{g}$ is the gravitational acceleration vector, $G(\mathbf{x}, t)$ is a characteristic function that is equal to the change in density over the initial stable density stratification, and ∂S is the surface of a sphere of radius a.

The total force due to tangential stresses on the surface of the sphere is

$$\int_{\delta S} \boldsymbol{\tau} \cdot \sigma_{\mathbf{w}} \cdot \mathbf{n} dS \tag{3.28}$$

Because we integrate over the surface of the sphere, the contributions of the horizontal portion of the tangential force vector will cancel out. So

$$\int_{\partial S} \boldsymbol{\tau} \cdot \sigma_{\mathbf{w}} \cdot \mathbf{n} dS = \int_{\partial S} \tau_{r\theta} \sin(\theta) dS + \int_{\partial S} \tau_{r\theta} \cos(\theta) dS = \int_{\partial S} \tau_{r\theta} \sin(\theta) dS \tag{3.29}$$

So, in order to compute $\int_{\partial S} \tau_{r\theta} \sin(\theta) dS$, it suffices to compute $\int_{\partial S} \boldsymbol{\tau} \cdot \sigma_{\mathbf{w}} \cdot \mathbf{n} dS$

Now consider

$$\int_{\Omega_f} [\nabla \cdot (\boldsymbol{\tau} \cdot \sigma_{\mathbf{w}}) - \nabla \cdot (\mathbf{w} \cdot \sigma_\tau)] dV = \int_{\Omega_f} \boldsymbol{\tau} \cdot (G(\mathbf{x}, t)\mathbf{g}) dV \tag{3.30}$$

using the incompressibility and the divergence of the stress tensors.

Using the divergence theorem and boundary conditions on $\mathbf{w}$ to rewrite the left-hand side, we have

$$\int_{\partial S} \boldsymbol{\tau} \cdot \sigma \cdot \mathbf{n} dS = \int_{\Omega_f} \boldsymbol{\tau} \cdot (G(\mathbf{x}, t)\mathbf{g}) dV \tag{3.31}$$

Hence combining this with (3.4) we have the desired result for G_2

$$G_2 = \frac{3}{4} a^2 \pi \mu \int_{\Omega_f} \boldsymbol{\tau} \cdot (G(\mathbf{x}, t)\mathbf{g}) dV \tag{3.32}$$

Evaluation of I_2

I_2, which is required only for the evaluation of $V(t)$, may be computed indirectly as follows.

In the work of [26] and [1], the force on the sphere due to the perturbation velocity $\mathbf{w}$, here denoted $D_{\mathbf{w}}$, is calculated, using the reciprocal theorem, by integrating over the volume of displaced ambient fluid. By completing a drag calculation almost identical to the calculation of the force on the drop in section 2.3.2, we find that the force on the sphere due to $\mathbf{w}$ expressed in terms of the coefficients of the expansions in (2.50) is

$$D_{\mathbf{w}} = \frac{4\pi a^2 \mu}{3} (G_2 - I_2) \tag{3.33}$$

Hence, we may compute I_2 as

$$I_2 = G_2 - D_{\mathbf{w}} \frac{3}{4\pi a^2 \mu} \tag{3.34}$$

As in the case of G_2, the normal stress coefficients computed in this manner agree with those computed using the appropriate numerical derivatives of $\mathbf{w}$, but are much less prone to issues due to numerical error.

Stokes Velocity

As in [1], the second reflection $\mathbf{u}^{(2)}$ from (2.70) is separated into time- and spacially-dependent parts, so that $\mathbf{u}^{(2)} = f(\mathbf{x})V(t)$. The spacially-dependent portion is computed on a grid in advance, and cubic spline interpolation is used to evaluate it at intermediate points as needed.

The remaining terms of the Stokes velocity, namely $\mathbf{u}^{(0)}$, $\mathbf{u}^{(1)}$, and (approximate) $\mathbf{u}^{(3)}$ all depend on $V(t)$ and G_n as well and do not involve the expensive integral evaluations necessitated by $\mathbf{u}^{(2)}$. So these are simply computed at each timestep as needed.

Finally, the full Stokes velocity is added to the perturbation velocity at each interface point to obtain the final velocity field with which to advect the interface.

Advection of Density Distribution

Advection

The interface is advected at each timestep using an adaptive Runge-Kutta-Fehlberg (RKF45) method, from a library implemented by Sandia National Laboratory.

Insertion of new interface points

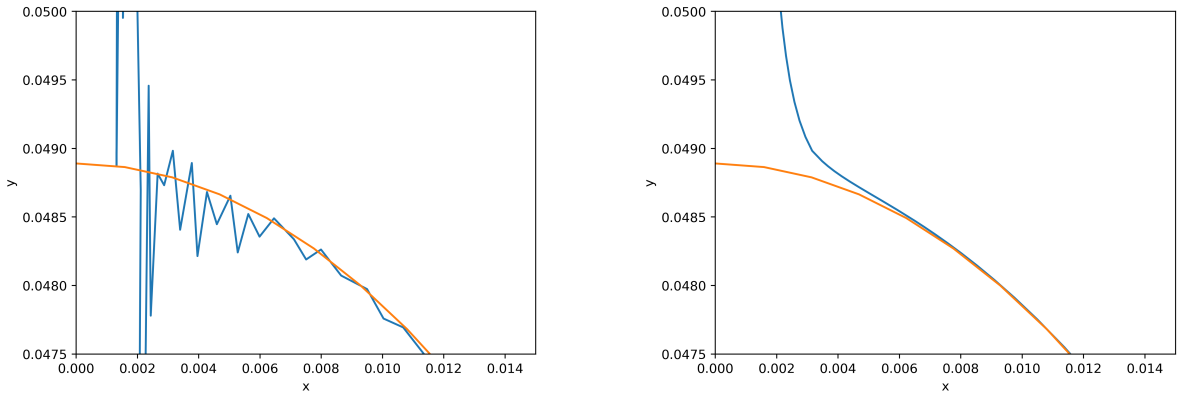

Figure 3.3: The result of using cubic spline interpolation to add new points (left) versus using the "streamline interpolation" described in this section (right). The image portrays a settling rather than rising drop.

As the interface is advected, it stretches around the droplet, and the Lagrangian marker points used to track its position become more sparsely spaced. Thus it becomes necessary to add points to fill the gaps that form as the simulation progresses. For the most part, this is done in the same way as in the case of [1], using cubic spline interpolation. However, when the density interface is very close to the drop surface, a more creative approach must be taken. This approach is, to the best of my knowledge, an original method.

We illustrate the general approach using the case of a drop rising in infinite homogeneous ambient fluid. As in the case of the full problem, we want to advect Lagrangian marker points denoting an interface, and add in more points as the interface is stretched around the droplet. Consider the streamlines for the outer streamfunction of the first reflection flow (2.46), a few of which are plotted in Figure 3.4. We include the level sets that fall inside the drop as well as they are important to our method even though they fall outside the region of interest (the fluid domain exterior to the drop).

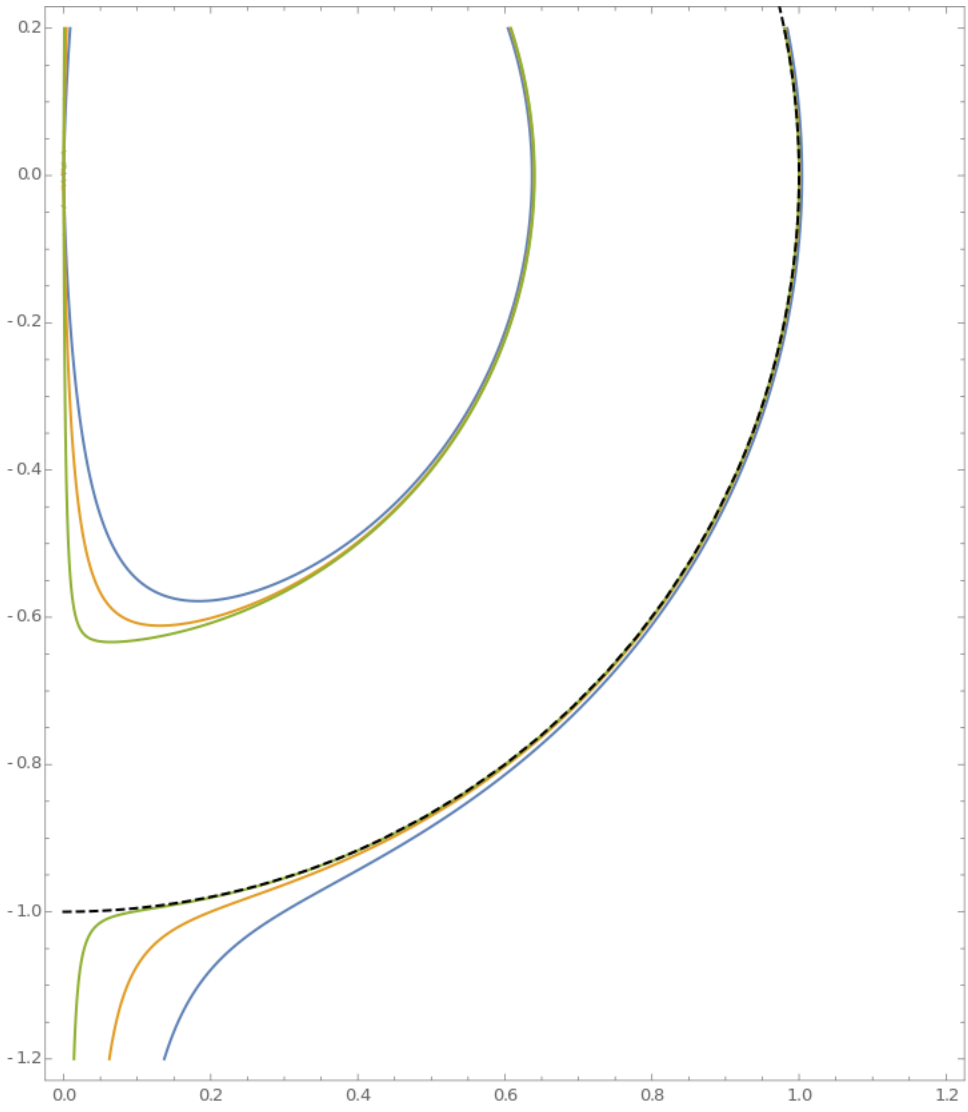

Figure 3.4: Streamlines of the first reflection flow

Note that, since this flow field is time-independent, the level sets of the streamfunction give the trajectories of the fluid particles moving along each one. Note also that trajectories that are very close to each other in the region close to the equator of the drop become much further apart as the fluid is advected around towards the hyperbolic stagnation point at the trailing end of the drop. Hence if we want to add a new point in this sensitive equatorial region, care must be taken that it falls on a trajectory in between those of the points consecutive to it.

For example, suppose that in the image above, we want to insert a point $\mathbf{x}_{new}$ in between a point $\mathbf{x}_i$ on the green trajectory and $\mathbf{x}_{i+1}$ on the yellow trajectory. However, suppose that we are using

cubic interpolation to do so, and because this method is blind to the precise flow geometry under consideration, the new point ends up being placed on the red trajectory. Though the placement of the new point seems reasonable at the time it is made, as the three points are advected towards the trailing end of the drop, the new point ends up being pulled inwards, on a slow trajectory that moves more inwards toward the hyperbolic stagnation point, while the other points move on past the drop on much faster trajectories. This leads to a sharp kink in the interface, which creates a very unrealistic density transition interface.

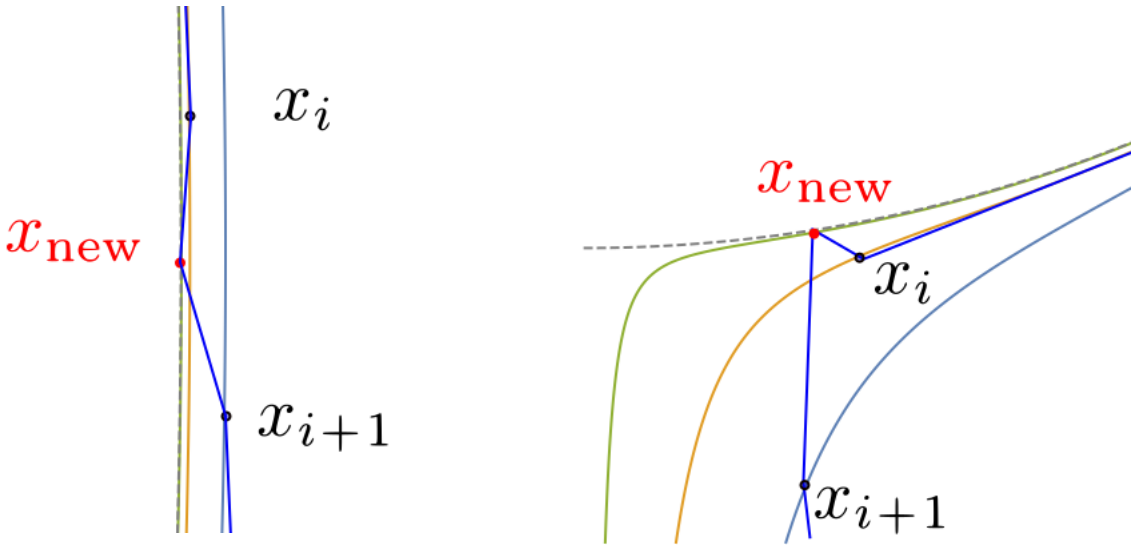

Figure 3.5: Initial position of point placed using cubic spline interpolation, then after advection. Note that the error in placement of the new point is amplified by the behavior of the flow near the hyperbolic stagnation point.

So, rather than blindly using the usual cublic spline interpolation scheme to place the new interface points, we instead use our knowledge of the analytic expression for the streamfunction ψ to ensure that the new point falls on a trajectory in between those of the points next to it in the interface. Working in spherical coordinates (r, θ), so that $\mathbf{x}_1 = (r_1, \theta_1)$, $\mathbf{x}_2 = (r_2, \theta_2)$, and $\mathbf{x}_{new} = (\hat{r}, \hat{\theta})$, we set

$$\hat{\theta} = \frac{\theta_1 + \theta_2}{2}, \tag{3.35}$$

then solve for $\hat{r}$ so that

$$\psi(\hat{r}, \hat{\theta}) = \frac{\psi(r_1, \theta_1) + \psi(r_2, \theta_2)}{2} \tag{3.36}$$

which is very simply reduced to a rootfinding problem, and solved to machine precision with a few iterations of Newton's Method using r_1 or r_2 as our initial guess.

The primary issue with this method is that occasionally, when the density transition interface is very close to the drop surface, there is a second solution to (3.36) just inside the drop. So care must be taken to ensure that the correct root is found. However, the advantages of this approach cannot be overstated. Figure 3.5.2 shows the interface resulting from Falcon's spline interpolation approach, as compared with the interface resulting from the streamline interpolation approach, with the same numerical parameters.

Code Verification

In this section, we discuss the comparison of our code to a number of analytic benchmarks in order to verify that it is a correct implementation of the mathematical model described in Chapter 2.

Reflections Flow

We begin by verifying that in homogeneous ambient fluid, our flow satisfies the boundary conditions on the drop and cylinder wall to the degree specified in our mathematical model. Setting the model parameters so that top- and bottom-layer desities were identical, the reflection flows were added to the implementation one at a time, and it was verified that they satisfied the boundary conditions at the drop surface and at the cylinder wall to the degree of accuracy specified in the model.

The implementation of the perturbation velocity was inherited from the preceding project. Several new considerations arose in the drop case, due to the much closer approach of the interface to the drop surface than to the sphere surface (see Fig. 3.6.1), but these were mostly trivial to resolve.

Evaluation of I_2 and G_2 using the numerical derivative approach was tested by comparison of the total force on the sphere in terms of the coefficients (3.33) to the total force on the sphere evaluated using the reciprocal theorem approach (A.4), and was found to be in good agreement with this expression throughout the regime of convergence of the numerical derivative approach. This was then used to verify the G_2 coefficient computed using the reciprocal theorem approach.

A full convergence study for this code has yet to be undertaken and thus is omitted, but preliminary partial studies have been promising.

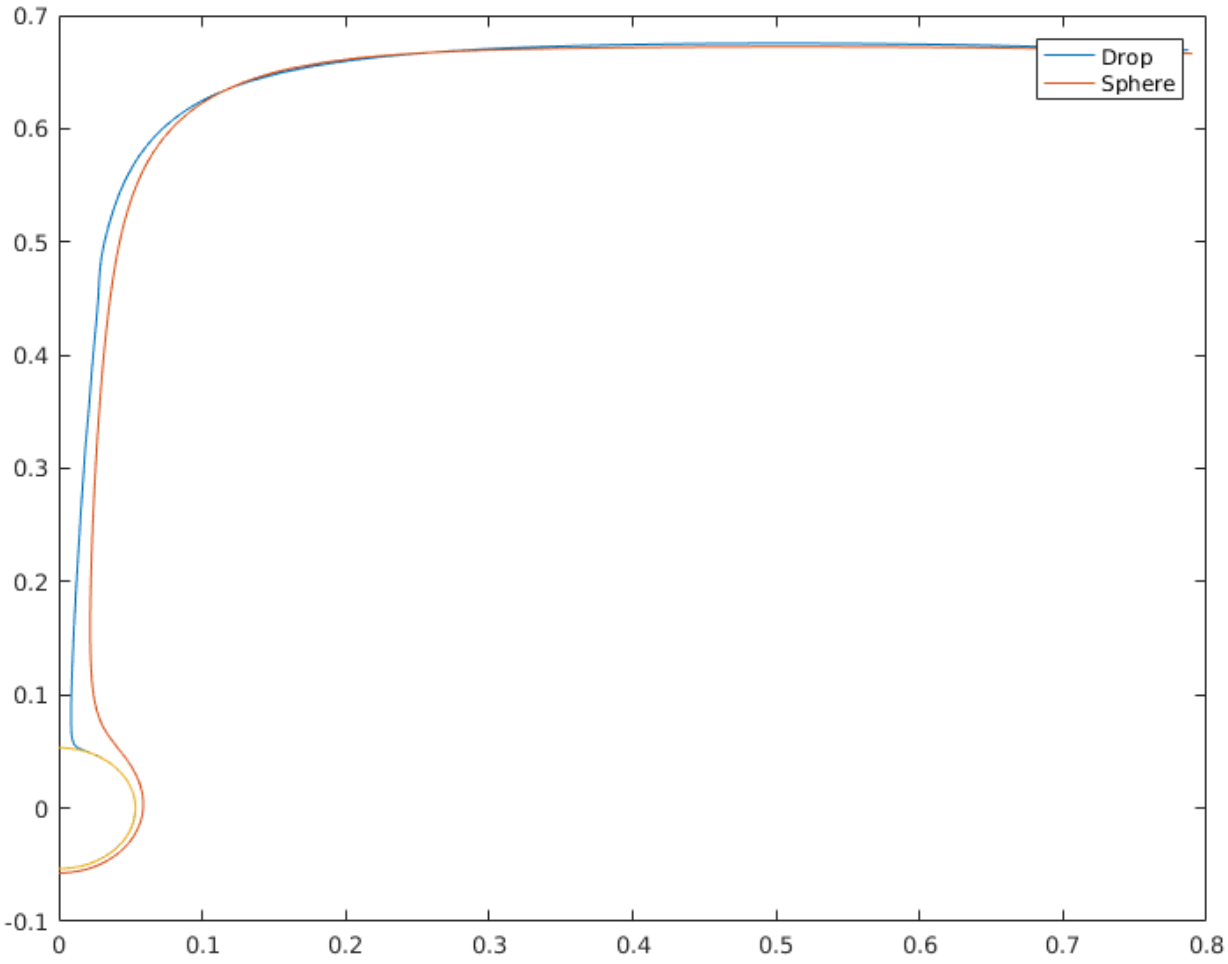

Figure 3.6: Comparison between density transition interfaces resulting from the drop and sphere implementations, with the same external fluid parameters and sphere/drop densities. Note that the drop quickly sheds the entrained fluid almost entirely, whereas the sphere maintains a thin layer of entrained fluid long after passing through the original density transition.

Conclusion

We have successfully implemented the mathematical model presented in Chapter Two. The chief distinctions between the implementations for the fluid and solid sphere cases lay in the evaluation of the coefficients for the Gegenbauer expansion of tangential and normal stresses and the tracking of the density transition interface very close to the surface of the drop. Thus far we have demonstrated the mathematical and numerical consistency of our model. In the following chapter, we present the experimental validation of this model.

CHAPTER 4

Experimental Validation

A series of experiments were conducted in order to validate this model. This chapter details the experimental setup and findings, and provides a quantitative comparison between the model predictions and experimental findings.

Experimental Setup

Experiments are conducted in a cylindrical borosilicate glass tank with radius .7874 cm and height 30.48 cm. The ambient fluid used was high-purity glycerol diluted with deionized water. Glycerol is moderately viscous and approximately Newtonian, allowing us to recreate a low Reynolds number regime at a length scale that is easily controlled in the lab. After dilution with deionized water, the gylcerol mixture was separated into several batches to which varying amounts of potassium iodide were added. Potassium iodide was used because its low diffusivity and small effect on the viscosity of the fluid to which it is added relative to the other salts tested. A stable two-layer stratification was then carefully poured in the tank, and then sharpened using a syringe to carefully draw out any portion where mixing between the two layers had occurred during pouring.

Glycerol was chosen as the ambient fluid because earlier experiments using (significantly cheaper) corn syrup were plagued with surfactant effects due to the accumulation of impurities on the droplet surface, as discussed in the following sections. The high purity of the glycerol and a scrupulous attention to the cleanliness of the setup enabled us to eliminate these effects. Furthermore, glycerol stratified with KI is very stable, with the density transition (which is observable due to the difference in refractive indices of the two fluid layers) remaining visually "sharp" for a timescale much longer than that of the individual experiments. Furthermore, KI has a minimal effect on the viscosity of glycerol.

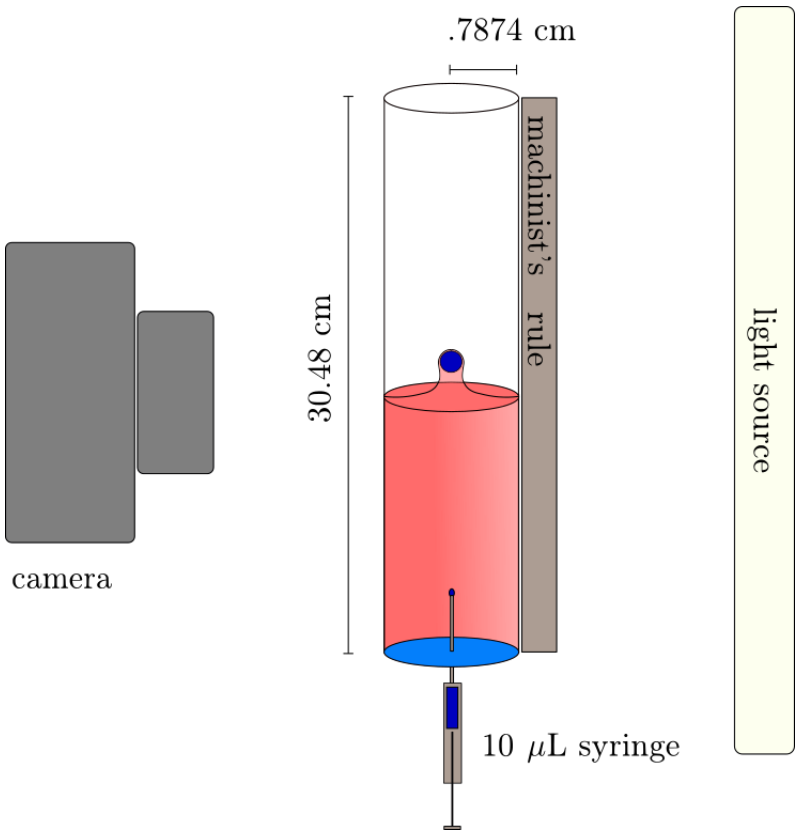

Figure 4.1: The experimental setup. The tank used was a borosilicate glass tube with a soft silicone rubber bottom. The experiment was diffusely backlit using a paper panel and an overhead projector. All experiments were recorded using a Nikon D7000 camera.

In order to conduct the experiments, droplets of dyed silicone oil of lower density than either of the two layers were injected from a microliter syringe through the center of the silicone rubber bottom of the tank. This allowed the release drops with volumes between 1 to 10 μL, with volumes accurate to within .5μL. The drop radius is one of the most significant sources of error in our experimental measurements; being able to compare measured radii with the expected radii based on the approximate drop volume proved useful in attempts to quantify this error.

Earlier in the the development of this experimental apparatus and procedure, corn syrup was used as ambient fluid. However, as previously mentioned, the accumulation of impurities at the trailing end of the drop surface evidently created a surface tension gradient, which caused interesting, time-varying effects as the drop surface accumulated surfactants and interacted with the density transion. These experimental observations are reported in section 4.5, and an experimental study

of the effect of adding increasing amounts of surfactant to the ambient fluid is also discussed in section 4.5. Because surfactant effects are absent only under the most tightly controlled of laboratory settings, further exploration of these effects would be an interesting and useful future avenue of research in this area.

All experiments were recorded using a Nikon D7000 camera. Depending on the length of the experiment, which varied from around ten seconds for some of the glycerol experiments up to half an hour for some of the corn syrup experiments, we recorded them in either high-quality video or with a series of remotely-triggered still images. The analysis of this data is discussed in the following section.

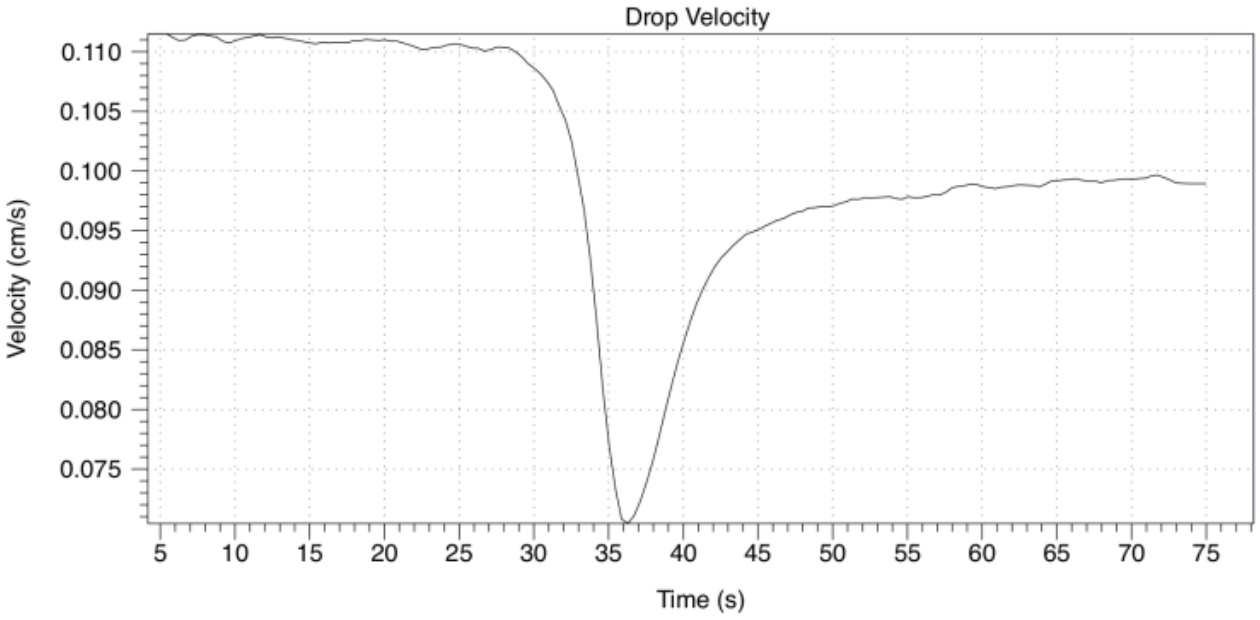

Figure 4.2: Typical experimental data for an approximately 1μL droplet of silicone oil ($\hat{\rho} = 0.87372$ g/cm^3, $\hat{\mu} = 2.1672$ mPa.s) rising through sharply stratified glycerol ($\mu = 274.757$ mPa.s, $\rho_1 = 1.26521$ g/cm^3, $\rho_2 = 1.24017$ g/cm^3)

Data Analysis

The experimental data is imported into DataTank. A background frame not containing the droplet is selected, and the color componend is subtracted from the rest of the frame. This makes the drop the most prominent feature in the frame, and it is selected by setting a boundary threshold. The center of mass and vertical diameter of the region within this boundary around the drop is

Quantity	Typical Value Range	Std. Dev. or Upper Bound on Error
Drop Radius a	.06-.1 cm	$\sigma = .001$ cm
Drop Viscosity $\hat{\mu}$	2.72 mPa.s	.05 mPa.s
Drop Density $\hat{\rho}$	0.8737	.0001 g/cm^3
Ambient Viscosity μ	272.0-278.0	2.0 mPa.s
Ambient Density ρ	1.24-1.26 g/cm^3	.0001 g/cm^3
Tank Radius R_0	.7874 cm	.0001 cm
Tank Length	10 cm	.05 cm

Table 4.1: Typical experimental parameters and error estimates

then tracked through the frames, and thus drop velocity and radius measurements may be made by comparing with a lengthscale in the video.

The resulting velocity data (See Fig. 2) clearly exhibit the dramatic slowing of the drop as it passes through the interface between the layers of differing density, and its eventual return to the terminal velocity in the top layer fluid as the drop sheds the entrained bottom-layer fluid.

Measurement Error Estimates

In collecting our experimental data, we measure drop and ambient viscosities, drop and ambient densities, drop radius, tank radius, and tank length. Some typical values and standard deviations or upper bounds on the error in each of these measurements are summarized in the table below.

The most significant of these by far is the error in drop radius measurement. In order to reduce this, we used a dual-camera setup was used, with a second camera zoomed in as closely as possible with a macro lens, but relative error in this measurement remains around two percent. Altering the drop radius by adding or subtracting two percent has a disproportionately large effect on the velocity predicted by our model, as demonstrated in Fig. 4.5.

Experimental Observations

The chief parameters of interest for comparison with our model are the density difference between the stratified layers $\Delta\rho = \rho_2 - \rho_1$ and the ratio of drop to cylinder radius a/R_0.

For each set of experiments, the base mixture of glycerol and deionized water was divided into three batches, to which varying quantities of potassium iodide were added. This provided use with three possible stratifications of the ambient fluid (and so three different values of $\Delta\rho$). Into each such setup, drops with volumes ranging from approximately 1-5μL were released, allowing us to test a fairly wide range of values for a/R_0. A typical dataset for one such experiment is presented in Fig. 4.3.

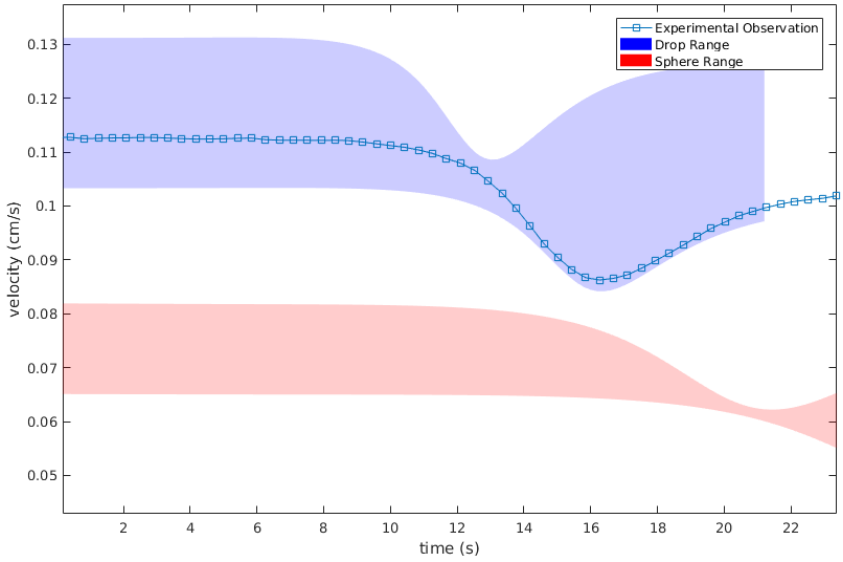

Figure 4.3: Variation in model velocity profile placeholder

Comparison with Model

Because of the high sensitivity of this system to experimental measurement error, we have endeavored to systematically quantify and control for this error as much as possible. The table 4.2.1 gives some typical margins of error for each of the experimental parameters being measured.

With this in mind, in Fig. 4.5 we have plotted the possible spread in velocity as predicted by the model as explained by the 95% confidence interval on the drop radius and ambient fluid viscosity measurements. This amounts to a roughly 10 micron standard deviation in drop radius, the most important source of error. In this figure, the model predictions, accounting for possible experimental error, fully capture the experimentally observed velocity. With this in mind, we are highly confident that the model explains the experimental observations to within the limitations of the theory. Because of the previously observed tendency of small, surfactant-laden droplets to behave more like solid spheres, the corresponding range for the solid sphere model is also shaded (in red). Thus it is clear that our observations can not be explained by current models for solid spheres in stratified fluid.

Experimental Observations of Surfactant Effects

As a drop rises in ambient fluid containing surfactant molecules or particles, these particles are swept to the bottom of the drop and accumulate about the stagnation point there. This creates a

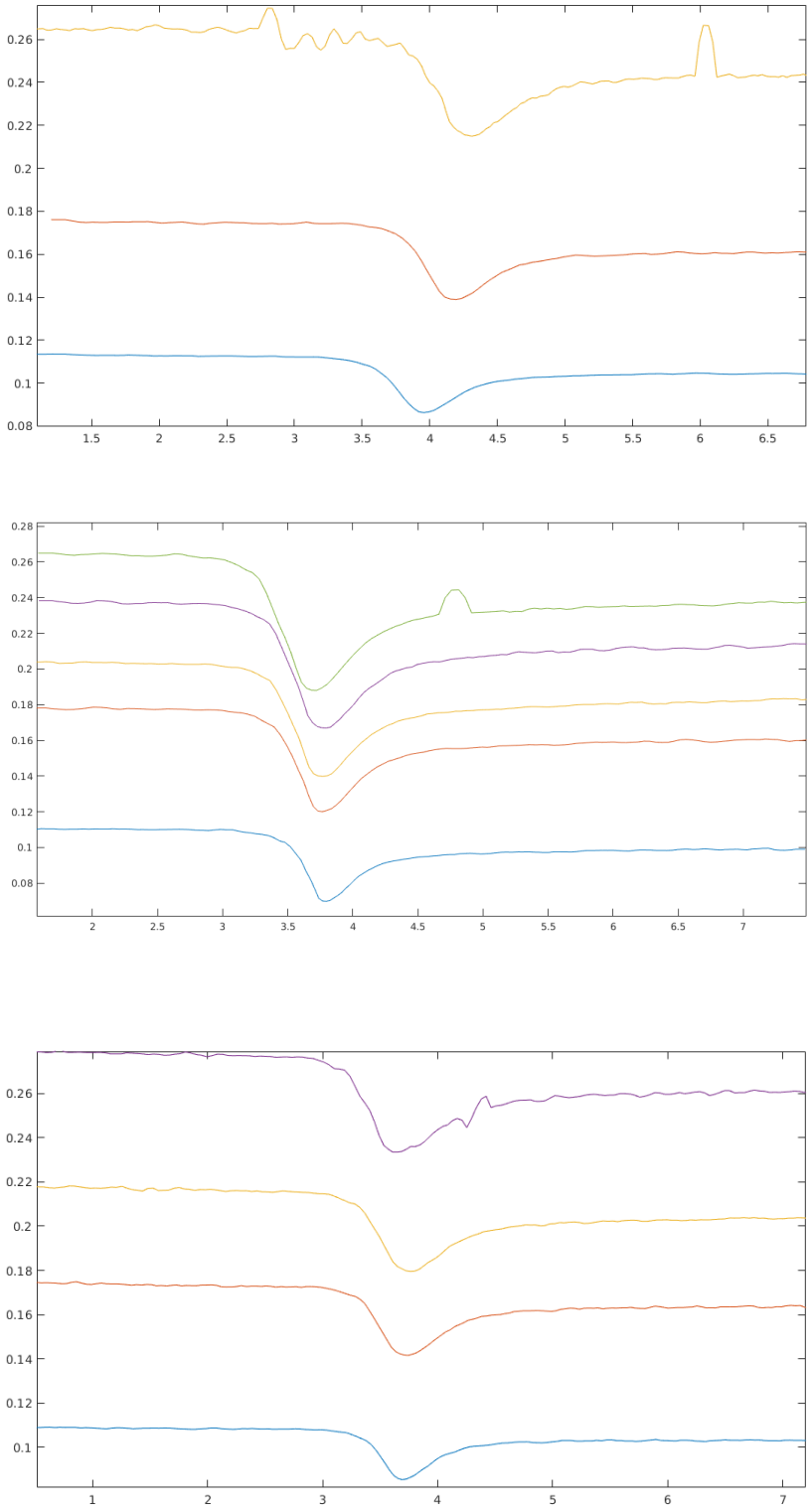

Figure 4.4: A typical experimental dataset. In this case, a 90% glycerol solution was divided into batches of 0%, 2%, and 4% potassium iodide by mass

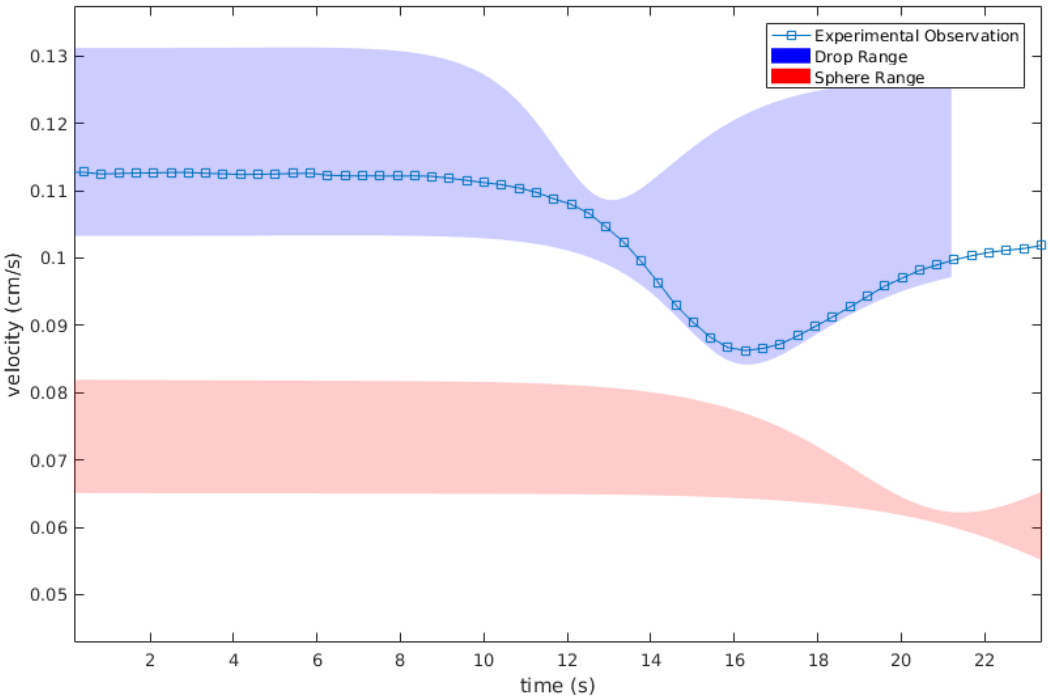

Figure 4.5: Experimental data overlaid with the drop model presented in this dissertation, and the sphere model presented in [1]

gradient of surface tension on the drop surface, with lower surface tension towards the bottom of the drop where the greater surfactant concentration is and higher surface tension in the cleaner regions towards the top of the drop. This surface tension gradient drives a Marangoni flow on the surface of the drop opposing the external fluid flow over the drop surface. The result, as has been observed by numerous authors [2], [25], [28], [24], is that the terminal velocity of a small drop is often much closer to that of a sphere than that of a droplet.

Several authors have proposed models of this surfactant effect in homogeneous ambient fluid, notably including Sadhal and Johnson [2] and Blanchette [24]. Johnson's model proposes using a mixed boundary condition on the drop surface, with the no-slip condition imposed on a spherical cap at the trailing end of the drop to account for the additional drag due to the surfactant accumulation, and a free interface at the top of the drop. In Fig. 4.6, a drop of brominated vegetable oil with dyed oil injected before release to trace the internal flow demonstrates the presence of surfactant effects. Observe the upward shifting of the internal stagnation points, just as predicted by Johnson and Sadhal's model; the streamlines for this model are included in Fig. 4.5 for easy qualitative comparison.

One of the primary challenges in designing this experimental setup lay in eliminating these effects in order to have a valid experiment for comparison with out model.

Surfactant Effects in Homogeneous Fluid

In the process of attempting to eliminate the surfactant effects observed in our experiments, we first ran a set of experiments with varying amounts of surfactant (Dawn) added to the ambient fluid, in hopes of reaching the theoretical "remobilization regime" [24] in which the surface of the drop would be saturated with surfactant, eliminating the gradient driving the Marangoni flow which caused the erratic behavior. In this experiment, increasing the surfactant concentration stabilized the drop behavior (i.e., no more weird accelerations as described in section 4.5.2) and caused it to behave increasingly like a solid sphere. Unfortunately, we were not able to observe the theoretical remobilization regime because adding enough surfactant to saturate the drop surface also reduces the surface tension of the drop enough to cause even very small drops to break up. Additionally,

Surfactant effects in stratified fluid

The stratified case presented the most interesting surfactant effects observed during our experimental investigation of this problem. Across year's worth of experiments, we consistently observed

Figure 4.6: Oil drop rising through stratified corn syrup, with dyed oil injected to trace flow inside the drop. Note the upward shift of the internal stagnation points, as predicted by Sadhal's model.

Surfactant Concentration	Observed Drop V	Theoretical Drop V	Theoretical Sphere V
0% (control)	.08-.119, increasing	0.182618	0.13016
0.2%	0.13257	0.17572	0.12567
1%	0.12954	0.17627	0.12740
1.5%	0.1025	0.15806	0.11563
2.0 %	.2071	0.30078	0.21722

Table 4.2: Drop velocities (cm/s) in homogeneous corn syrup with increasing surfactant concentration

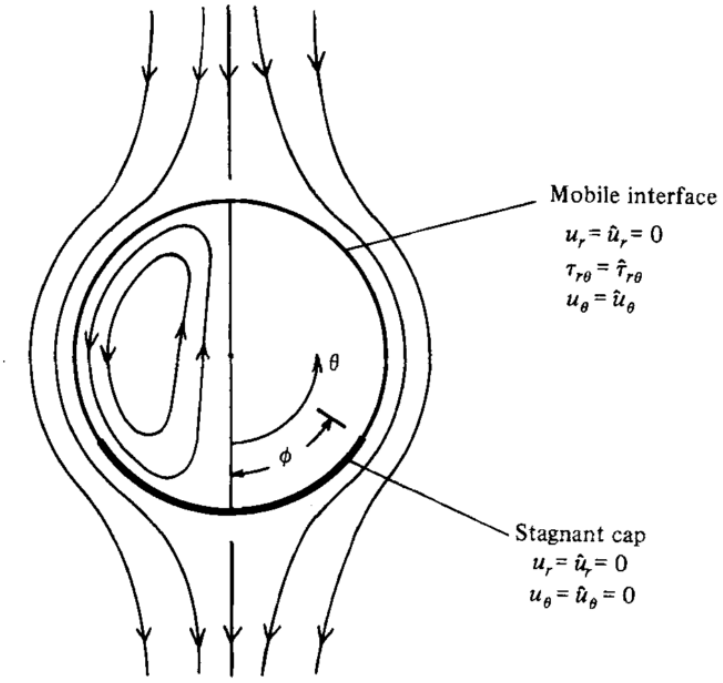

Figure 4.7: Streamlines resulting from a spherical-cap model of surfactant effects proposed by Sadhal and Johnson, borrowed from [2]. Compare to Fig. 4.6

that, when corn syrup was used as the ambient fluid, the drop would briefly accelerate to above its terminal velocity shortly before and after it decelerated when passing through the density transition. When, in order to rule out the effects of optical distortion and possible camera error, the same experiment was repeated with a solid sphere of comparable density and size to the droplet, this acceleration was not observed. See Figure 4.8 for this comparison.

At present, we do not have a good explanation for this effect. For the purposes of the model detailed in this dissertation, we are interested only in eliminating these effects and validating the model with constant surface tension. However, future research in this direction could yield interesting results.

Conclusion

This concludes the experimental validation of this model. The experimental data shows generally good agreement with the model predictions, within the appropriate error bounds on our experimental measurements. We also observed a variety of surfactant effects in both homogeneous and stratified fluid, which could be interesting and useful additions to our model in the future.

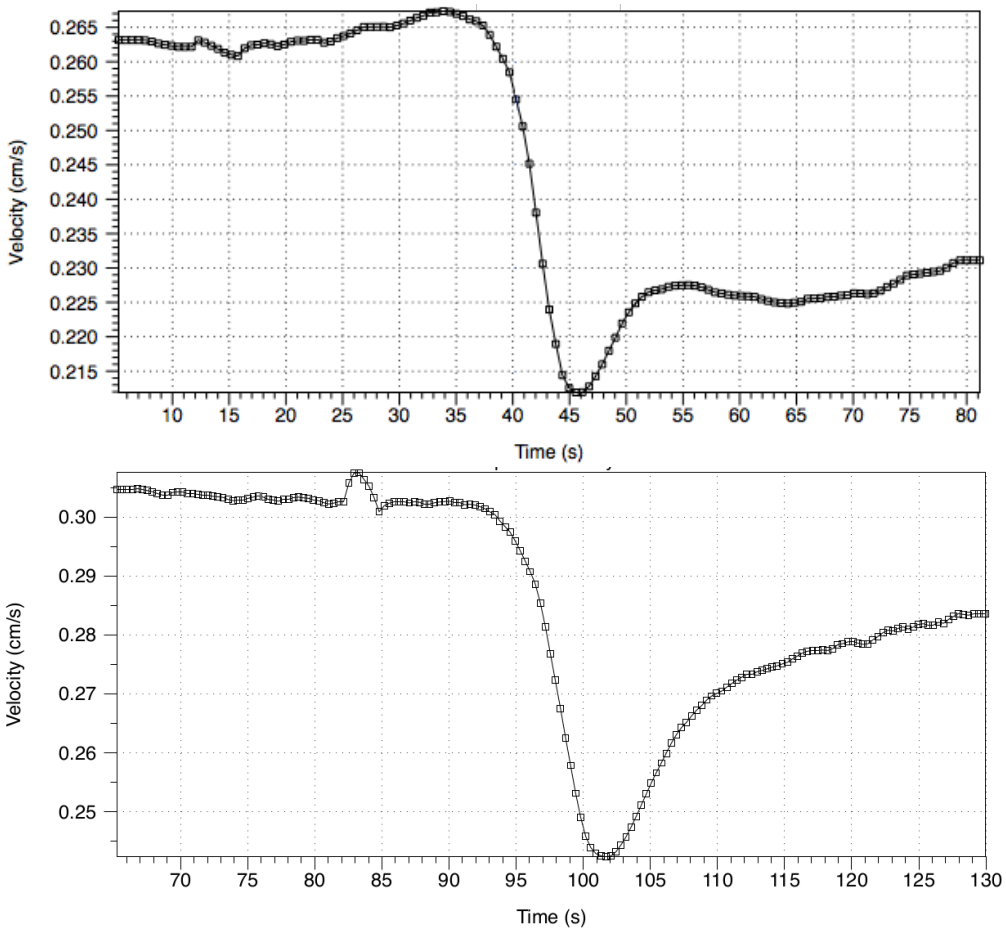

Figure 4.8: Comparison between drop with time-varying surfactant effects (top) and solid sphere of comparable size and density in the same ambient fluid (bottom).

CHAPTER 5

Conclusions and Future Work

Conclusion

In this dissertation, we have presented a numerically-assisted, experimentally-verified, first-principles model of an oil drop rising through ambient fluid with a sharp two-layer density stratification. Our model, based upon similar work for the case of a solid sphere, is the first of its kind in the literature, and is consistent with our experimental data to within the bounds of its theoretical limitations and the error in our experimental measurements.

The mathematical derivation of this model is built using mathematical tools generalized from pre-existing models for solid and fluid spheres in cylindrical enclosures, and for solid spheres in sharply-stratified fluid. The generalization of these techniques, constitutes the first novel analytic work in this dissertation.

The numerical implementation of the mathematical model is also based upon pre-existing work for a solid sphere in stratified ambient fluid, but the case of a drop presents novel challenges. The rapid shedding of entrained fluid by the drop and the hyperbolic stagnation point at the trailing end of the drop necessitated the development of more refined techniques for tracking the interface defining the density transition in regions very close to the drop surface. Additionally, the evaluation of the coefficients of the Gegenbauer polynomial expansion of the tangential stress due to the density-driven perturbation portion of the flow, a key part of our model, required the derivation of a numerically tractable formulation of the perturbation stress integrand. Our implentation is ultimately very fast, with a full simulation with experimental parameters taking no more than half an hour on a single CPU. Full convergence studies are ongoing, but were not completed at the time of submission of this dissertation.

Experimental studies were conducted in order to validate the model. The experimental realization of such an idealized scenario proved to be very exacting, but ultimately feasible. In addition to the experimental validation of the mathematical model, numerous interesting observations of the

effects of surfactant accumulation on the surface of the drops were made. In particular, time-varying surfactant effects due to the interaction of the stratified ambient fluid with the accumulated surfactant were observed.

Future Work

While interesting in its own right, our extremely idealized model presents several obvious avenues for future extension of these results. In particular, surfactant effects are very important to any of the realistic applications of this work, such as oil drops from undersea seeps and spills. Additionally, density distributions other than a sharp stratification, such as multi-layered or linearly-stratified ambient fluid, are likely of interest for various applications.

The time-varying surfactant effects observed in the corn syrup experiments could be an especially interesting direction for future research. Some numerical studies exist [24], but no first-principles models of these effects can be found in the current literature. Incorporating these effects into our current model would also be a nontrivial but potentially rewarding undertaking. Additionally, as our experiments are the first to examine this scenario, no other experimental observations of these effects seem to exist. While difficult to quantitatively observe in an experimental setting, a more thorough investigation of the interaction between surfactant effects and the passage of drops through density stratifications would be highly relevant to the study of oil spill dispersion in the ocean.

An extension of this model to linearly stratified fluid is another obvious next step. A possible first approach might be to advect a grid of Lagrangian markers, each assigned a density value, then interpolating to obtain the actual density distribution. While much more computationally demanding, this could be accomplished with some relatively minor additions to the current code. It would also require volume integration of the Greens function convolution with the density characteristic function over the entire fluid volume. A multi-layer sharp stratification could likely be accomplished in much the same manner, as previously discussed in the case of a settling solid sphere [26].

Support

Portions of this work were supported by NSF RTG DMS-0943851, NSF DMS-1517879, and ONR DURIP N00014-12-1-0749.

REFERENCES

[1] C. Falcon, *Entrainment Dominated Effects in the Long Residence Times of Solid Spheres Settling in Sharply Stratified Miscible Viscous Fluids*. PhD thesis, University of North Carolina at Chapel Hill, 2016.

[2] S. Sadhal and R. Johnson, "Stokes flow past bubbles and drops partially coated with thin films. Part 1. Stagnant cap of surfactant film- exact solution," *Journal of Fluid Mechanics*, vol. 126, no. 1983, pp. 237–250, 1983.

[3] S. MacIntyre, A. L. Alldredge, and C. C. Gotschalk, "Accumulation of marine snow at density discontinuities in the water column," *Oceanographic Literature Review*, vol. 43, no. 3, p. 252, 1996.

[4] C.-N. Tzou, "Formation of underwater plumes and velocity variations due to entrainment in stratified environments," *ProQuest Dissertations and Theses*, p. 141, 2015.

[5] E. A. Widder, S. Johnsen, S. A. Bernstein, J. F. Case, and D. J. Neilson, "Thin layers of bioluminescent copepods found at density discontinuities in the water column," *Marine Biology*, vol. 134, no. 3, pp. 429–437, 1999.

[6] R. Camassa, C. Falcon, J. Lin, R. M. McLaughlin, and N. Mykins, "A first-principle predictive theory for a sphere falling through sharply stratified fluid at low Reynolds number," *Journal of Fluid Mechanics*, vol. 664, pp. 436–465, oct 2010.

[7] R. Camassa, S. Khatri, R. M. McLaughlin, J. C. Prairie, B. L. White, and S. Yu, "Retention and entrainment effects: Experiments and theory for porous spheres settling in sharply stratified fluids," *Physics of Fluids*, vol. 25, no. 8, 2013.

[8] G. G. Stokes, *On the effect of the internal friction of fluids on the motion of pendulums*, vol. 9. Pitt Press Cambridge, 1851.

[9] J. S. Hadamard, "Mouvement permanent lent d'une sphere liquide et visqueuse dans un liquide visqueux," *C. R. Acad. Sci.*, no. 152, pp. 1735–1738, 1911.

[10] W. Rybczynski, "Über die fortschreitende Bewegung einer flüssigen Kugel in einem zähen Medium," *Bull. Acad. Sci. Cracovie, A.*, pp. 40–46, 1911.

[11] C. W. OSEEN, "Uber die stokes'sche formel und uber eine verwandte aufgabe in der hydrodynamik," *Arkiv Mat., Astron. och Fysik*, vol. 6, p. 1, 1910.

[12] P. H. L. F.R.S., "Xv. on the uniform motion of a sphere through a viscous fluid," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 21, no. 121, pp. 112–121, 1911.

[13] B. I. A. N. Proudman and J. R. A. Pearson, "Expansions at small Reynolds numbers for the flow past a sphere and a circular cylinder," no. 1851, pp. 237–262, 1956.

[14] A. Acrivos and T. Taylor, "On the deformation and drag of a falling viscous drop at low Reynolds number," no. 1913, pp. 466–476, 1963.

[15] A. S. Brignell, "The deformation of a liquid drop," vol. XXVI, 1973.

[16] S. Deo and D. Datta, "Stokes Flow Past a Fluid Prolate Spheroid," 2003.

[17] D. K. Srivastava, R. R. Yadav, and S. Yadav, "Steady Stokes Flow Around Deformed Sphere. Class of Oblate Axisymmetric Bodies," *Int. J. of Appl. Math and ...*, vol. 8, no. 9, pp. 17–53, 2012.

[18] R. Ladenberg, . PhD thesis, 1907.

[19] H. Faxén, "Der Widerstand gegen die Bewegung einer starren Kugel in einer zähen Flüssigkeit, die zwischen zwei parallelen ebenen Wänden eingeschlossen ist," *Annalen der Physik*, vol. 373, pp. 89–119, 1922.

[20] J. Happel and B. Bryne, "Motion of a sphere in a cylindrical tube," *Industrial & Engineering Chemistry*, vol. 46, no. 6, pp. 1181–1186, 1954.

[21] W. Haberman and R. Sayre, "Motion of rigid and fluid spheres in stationary and moving fluids inside cylindrical tubes," 1958.

[22] C. M. Linton, "Multipole methods for boundary-value problems involving a sphere in a tube," pp. 187–204, 1995.

[23] F. Blanchette, "Flow lines and mixing within drops in microcapillaries," *Physical Review E*, vol. 80, no. 6, p. 066316, 2009.

[24] F. Blanchette and A. M. Shapiro, "Drops settling in sharp stratification with and without Marangoni effects," *Physics of Fluids*, vol. 24, no. 4, pp. 1–17, 2012.

[25] G. K. Batchelor, *An introduction to fluid dynamics*, vol. 515. 1967.

[26] J. Lin, *An Experimental and Mathematical Study on the Prolonged Residence Time of a Sphere Falling through Stratified Fluids at Low Reynolds Number*. PhD thesis.

[27] J. Happel and B. Byrne, "Motion of a sphere and fluid in a cylindrical tube," 1954.

[28] K. E. Spells, "A Study of Circulation Patterns within Liquid Drops moving through a Liquid," *Proceedings of the Physical Society. Section B*, vol. 65, no. 7, pp. 541–546, 1952.

[29] J. J. L. Higdon, "A hydrodynamic analysis of flagellar propulsion," *Journal of Fluid Mechanics*, vol. 90, no. 04, p. 685, 1979.

[30] L. Landau and E. Lifshitz, *Fluid Mechanics*. 1959.

APPENDIX A

Perturbation Velocity

The Green's function used for the computation of the perturbation flow $\mathbf{w}$ is derived by Oseen [11], and is the free-space Green's function for Stokes flow external to a sphere located at the origin. The expression and explanation given below is heavily based upon that presented by Higdon [29]. It is given by $W_j(\mathbf{x},\mathbf{y} = gT_{j3}/8\pi\mu$, where

$$\begin{aligned} T_{jk}(\mathbf{x},\mathbf{y}) =& \frac{\delta_{jk}}{r} + \frac{(x_j-y_j)(x_k-y_k)}{r^3} - \frac{a}{|\mathbf{y}|}\frac{\delta_{jk}}{r^*} - \frac{a^3}{|\mathbf{y}|^3}\frac{(x_j-y_j^*)(x_k-y_k^*)}{r^{*3}} \\ & - \frac{|\mathbf{y}|^2-a^2}{|\mathbf{y}|}\Big\{\frac{y_j^*y_k^*}{a^3r^*} - \frac{a}{|\mathbf{y}|^2r^{*3}}[y_j^*(x_k-y_k^*)+y_k^*(x_j-y_j^*)] \\ & + \frac{2y_j^*y_k^*}{a^3}\frac{y_l^*(x_l-y_l^*)}{r^{*3}}\Big\} - (|\mathbf{x}|^2-a^2)\frac{\partial\phi_k}{\partial x_j}, \end{aligned} \tag{A.1}$$

$$\begin{aligned} \frac{\partial\phi_k}{\partial x_j} =& \frac{|\mathbf{y}|^2-a^2}{2|\mathbf{y}|^3}\Big\{\frac{-3y_k(x_j-y_j^*)}{ar^{*3}} + \frac{a\delta_{jk}}{r^{*3}} - \frac{3a(x_j-y_j^*)(x_k-y_k^*)}{r^{*5}} \\ & - \frac{2y_ky_j^*}{ar^{*3}} + \frac{6y_k}{ar^{*5}}(x_j-y_j^*)(x_l-y_l^*)y_l^* \\ & + \frac{3a}{|\mathbf{y}^*|}\frac{y_k^*(x_j-y_j^*)r^{*2} + (x_j-y_j^*)(x_k-y_k^*)|\mathbf{y}^*|^2 + (r^*-|\mathbf{y}^*|)r^{*2}|\mathbf{y}^*|\delta_{jk}}{r^{*3}|\mathbf{y}^*|(|\mathbf{y}^*|r^* + x_ly_l^* - |\mathbf{y}^*|^2)} \\ & - \frac{3a}{|\mathbf{y}^*|}\frac{(|\mathbf{y}^*|(x_j-y_j^*)+r^*y_j^*)(y_k^*r^{*2} - (x_k-y_k^*)|\mathbf{y}^*|^2 + (x_k-2y_k^*)r^*|\mathbf{y}^*|)}{r^{*2}|\mathbf{y}^*|(|\mathbf{y}^*|r^* + x_ly_l^* - |\mathbf{y}^*|^2)^2} \\ & - \frac{3a}{|\mathbf{y}^*|}\frac{x_jy_k^* + |\mathbf{x}||\mathbf{y}^*|\delta_{jk}}{|\mathbf{x}||\mathbf{y}^*|(|\mathbf{x}||\mathbf{y}^*| + x_ly_l^*)} + \frac{3a}{|\mathbf{y}^*|}\frac{(|\mathbf{y}^*|x_j + |\mathbf{x}|y_j^*)(|\mathbf{y}^*|x_k + |\mathbf{x}|y_k^*)}{|\mathbf{x}||\mathbf{y}^*|(|\mathbf{x}||\mathbf{y}^*| + x_ly_l^*)^2}\Big\}, \end{aligned} \tag{A.2}$$

$\mathbf{y}^*$ is the inverse point inside the sphere of $\mathbf{y}$

$$\mathbf{y}^* = \frac{a^2}{|\mathbf{y}|^2}\mathbf{y},$$

$r = |\mathbf{x}-\mathbf{y}|$, and $r^* = |\mathbf{x}-\mathbf{y}^*|$.

Using this Green's function, we may compute $\mathbf{w}$ as

$$w_k(\mathbf{x},t) = \int_{\Omega_f} G(\mathbf{y},t)W_k(\mathbf{x},\mathbf{y})d_3y, \tag{A.3}$$

where Ω_f represents the fluid domain external to the sphere of radius a.

As discussed in Higdon [29], this Green's function is made up of a Stokeslet together with an assortment of image singularities, the latter of which cancel the velocity due to the stokeslet on the surface of the sphere in order to satisfy the no-slip boundary condition there.

As mentioned in [6] and [26], the generalized Lorentz reciprocal theorem may be used to compute the total force on the sphere due to this velocity field. We will refer to this as the "density anomaly force", and use a very similar approach in order to obtain an expression for G_2, described in Section 3.3.1.

The resulting expression for the density anomaly force is

$$f_w = \int_{\Omega_f} G(\mathbf{x}, t) \left(\frac{-3a(2Z^2 + R^2)}{4r^3} + \frac{a^3(2Z^2 - R^2)}{4r^5} \right) d\Omega \tag{A.4}$$

where $r = |\mathbf{y}|$, and $\mathbf{y} = (X, \varphi, Z)$ is the variable of integration in cylindrical coordinates. This expression is evaluated by our routine in the process of computing the drop velocity at each timestep.

APPENDIX B

Drop Deformation

Computing Drop Deformation

We here adopt the asymptotic analysis of Acrivos and Taylor [14], as presented by Brignell [15] on the deformation of a drop in infinite ambient fluid to the case of a drop in an infinite tank of stratified fluid

We first nondimensionalize by dividing all lengths by the drop radius a, all velocities by the terminal velocity of the drop V, and all stresses by $(\nu U)/a$, the typical viscous stress. The drop surface is now $R = 1 + \zeta(\cos(\theta))$, where ζ is the deviation from a spherical shape.

After nondimensionalization, the normal stress balance at the surface of the drop becomes

$$Ca(\mathbf{N} - \hat{\mathbf{N}}) = \frac{1}{R_1} + \frac{1}{R_2} \tag{B.1}$$

where $\mathbf{N}$ and $\hat{\mathbf{N}}$ are normal stress outside and inside the drop, respectively, and where $Ca = \frac{\mu U}{\sigma}$ is the capillary number. Therefore we can expand around $\sigma \to \infty$ by expanding around $Ca = 0$. The physical meaning of small Ca is that the stresses due to surface tension ($O(\sigma/a)$) are much larger than the viscous stresses ($O(\mu U/a)$).

Next, rewrite the normal stress equation in terms of ζ as

$$2 + Ca(\mathbf{N} - \hat{\mathbf{N}}) = \frac{d}{d\cos(\theta)} \left\{ (1 - \cos^2(\theta)) \frac{d\zeta}{d\cos(\theta)} \right\} + 2\zeta \tag{B.2}$$

See [30] for the details of the derivation of this equation.

This method given by Brignell for computing the deformation of a drop depends on the fact that the left hand side of the above, which contains terms dependent on the stream function, is of higher order in Ca that the right hand side, which contains terms dependent on shape. We exploit this by using the streamfunction from the flow around the drop in the preceding order to compute the next-higher order deformation, then recompute the flow around the drop with the new shape and repeat for as many terms as we like.

Rewriting the jump in normal stress on left-hand side of (B.2) as a series of Legendre polynomials so that our equation becomes

$$\gamma_0 + \sum_{n=2}^{\infty} \gamma_n P_n(\cos(\theta)) = \frac{d}{(d\cos(\theta))} \left\{ (1-\cos(\theta)^2) \frac{d\zeta}{d\cos(\theta)} \right\} + 2\zeta \tag{B.3}$$

and applying the conditions that the volume of the drop is conserved and the center of mass of the drop remains at the origin, and using a Legendre polynomial expansion to solve (B.3) we find that the deformation is given by

$$\zeta(\cos(\theta)) = \sum_{n=2}^{\infty} \frac{\gamma_n P_n(\cos(\theta))}{(2+n)(1-n)}. \tag{B.4}$$

So in order to compute the deformation of the drop, we need only plug in the coefficients of the Legendre polynomial expansion of the normal stress jump from the flow in the infinite surface tension limit, and we may obtain approximations to the drop deformation. Note that, since $n \geq 2$, the portion of the normal stress on the drop due to the Hadamard solution (the term of the streamfunction expansion detailed in Ch. 2 that has no dependence on G_2) will not cause the drop to deform. Note also that, considering the leading-order $O(Ca^0)$ balance from (B.1), we are left with

$$\frac{1}{R_1} + \frac{1}{R_2} = constant \tag{B.5}$$

Together with the conservation of drop volume condition, and the condition that the center of mass of the drop remain at the origin, this is equivalent to the condition that the drop remain spherical.

APPENDIX C

Expansion of Second Reflection

Consider the second reflection for the droplet flow (2.70). Because of the cumbersome integral form of this expression, in order to compute the third reflection we first need to seek an approximation about the center of the droplet (the origin).

As in [20], we make the change of variables $\alpha = \lambda R_0$, then expand our integrands about $(R, X) = (0, 0)$ as follows:

$$I_0(\alpha\frac{R}{R_0}) = 1 + \frac{\alpha^2 R^2}{4R_0^2} + ... \tag{C.1}$$

$$I_1(\alpha\frac{R}{R_0}) = \frac{\alpha R}{2R_0} + ... \tag{C.2}$$

$$\cos(\frac{\alpha X}{R_0}) = 1 - \frac{\alpha^2 X^2}{2R_0^2} + ... \tag{C.3}$$

$$\sin(\frac{\alpha X}{R_0}) = \frac{\alpha X}{R_0} + ... \tag{C.4}$$

$$\tag{C.5}$$

Now inserting these into (2.70), making the appropriate change of variables to integrate numerically with respect to α, we find the following leading order terms of the expansions of $u_X^{(2)}$ and $u_R^{(2)}$ about the center of the drop:

$$u_X^{(2)}(0,0) = \frac{(-1)}{(2\pi)}\frac{(aV)}{\left((\kappa+1)\mathrm{R}0^3\right)}\left(6.51555a^2\kappa + (-12.8493\kappa - 8.56623)\mathrm{R}0^2\right)$$

$$u_R^{(2)} = 0$$

This is sufficient to construct an approximation to the third reflection, so that error on the drop surface is at most $O(\frac{a}{R_0})^2$.

APPENDIX D

Alternative Flow Calculation

We here fully follow the approach of Haberman [21] to compute the flow about a drop in a cylinder of stratified fluid using the Method of Multipoles.

Setup

The setup is much the same as in the previous derivation of the Stokes velocity, only with the wall boundary condition satisfied by directly imposing it in the expansion of the 0th and 1st reflections, rather than by a method of reflections. The boundary conditions on the surface of the drop are, as before, continuity of velocity and tangential stresses. The boundary conditions away from the drop, however, become (in cylindrical coordinates (R, φ, X))

$$\mathbf{u} \rightarrow -\mathbf{U} \text{ as } X \rightarrow \infty \tag{D.1}$$

$$\mathbf{u} = -\mathbf{U} \text{ when } R = R_0 \tag{D.2}$$

$$\tag{D.3}$$

This is the no-slip condition at the cylinder wall from the drop frame of reference, with quiescent fluid at infinity; we impose this boundary condition to replace the condition at infinity in our previous boundary conditions in 2.4.2. We wind up with the following solutions for the coefficients of the Gegenbauer expansions of the Stokes streamfunctions (2.46) satisfying the conditions at the surface of the drop:

$$A_2 = -\frac{a^4 G_2 \mu + a^2 D_2 3(2\mu - 3\hat{\mu}) - 15 B_2 \hat{\mu}}{6a^3(\mu - \hat{\mu})} \tag{D.4}$$

$$C_2 = \frac{a^4 G_2 \mu - 3a^2 D_2 \hat{\mu} - 3B_2(2\mu + 3\hat{\mu})}{6a^5(\mu - \hat{\mu})} \tag{D.5}$$

$$E_2 = \frac{(15B_2 + 3a^2 D_2 - a^4 G_2)\mu}{6a^3(\mu - \hat{\mu})} \tag{D.6}$$

$$F_2 = \frac{(-15B_2 - 3a^2 D_2 - a^4 G_2)\mu}{6a^5(\mu - \hat{\mu})} \tag{D.7}$$

$$\tag{D.8}$$

In cylindrical coordinates, the streamfunction for the flow external to the droplet satisfying the boundary condition at the cylinder wall is

$$\begin{aligned}\psi &= \int_0^\infty RK_1(aR)a_1 a\cos(aX)da + \int_0^\infty R^2K_0(aR)b_0\cos(aX)da + \\ &+ \int_0^\infty \{RI_1(aR)\left[\frac{2a}{(ab)^2}S_3 - S_4\right]a_1 a + S_3 b_0\} + R^2I_0(aR)(S_2a_1a + S_4b_0)\cos(aX)da + \frac{UR^2}{2}\end{aligned}$$

By expanding this, converting this into spherical coordinates, and comparing with the streamfunction in spherical coordinates (2.46), we obtain the following relationships between the respective unknowns:

$$B_2 = a_1\pi \tag{D.9}$$

$$D_2 = b_0\pi \tag{D.10}$$

$$A_2 = -4.40866(b_0)/R\lambda - 6.55507(a_1)/(R^3)\lambda^3 + U \tag{D.11}$$

$$C_2 = 1.31101(b_0)/(R^3)\lambda + 3.57466(a_1)/(R^5)\lambda^5 \tag{D.12}$$

where $\lambda = \frac{a}{R_0}$.

Simultaneously solving these equations (D.4-D.12), the coefficients for a solution which satisfies the boundary conditions on the drop surface and approximately satisfies the boundary condition on the cylinder wall may be obtained. The full expressions are too lengthy to include here. From these final solutions, the drag correction due to the cylinder wall may be evaluated as well.

APPENDIX E

Fortran Code

The full implementation for a drop rising through sharply-stratified fluid is included below. Additional libraries for the RKF45 method, cubic spline interpolation, fast Fourier transform, and evaluation of elliptic integrals and Bessel functions are required.

Global Variables

This module defines the global parameters and variables shared by the various subroutines.

```
!----------------------------------------------------------------------
! Main Routine for Oil Droplet Rising Through Sharply Stratified Fluid
!----------------------------------------------------------------------
!
!MODULE: Global Information
!
!> @author
!> H. Arrowood, UNC-CH. Adapted from code by C. Falcon, UNC-CH
!
!DESCRIPTION:
!>Defines global parameters and initializes global variables.
!----------------------------------------------------------------------
module globalinfo
    implicit none


!========Actual Experimental Parameters================================
    real (kind=8), parameter :: rho0pctKI = 1.24017
    real (kind=8), parameter :: rho2pctKI = 1.25335
    real (kind=8), parameter :: rho4pctKI = 1.26521
    real (kind=8), parameter :: rhotop = rho0pctKI!<top density g/cc
    real (kind=8), parameter :: rhobottom = rho4pctKI!<bot density g/cc
    real (kind=8), parameter :: rhodrop = 0.87372 !<drop density g/cc
```

```
    real (kind=8), parameter :: mu = .5*(2.782011+2.712486)!<ambient
    !<dynamic viscosity mPa.s/100
    real (kind=8), parameter :: muin = .021672  !<dynamic viscosity of
        !<drop mPa.s/100
    real (kind=8), parameter :: R = .05075!<radius of drop cm
    real (kind=8), parameter :: R0 = 0.7874 !<radius cylinder cm
    real (kind=8), parameter :: y0 = -1.5 !<init interface cm

!========Experimental parameters of equivalent falling drop problem=====
    !(since Claudia's w code depends on the falling geometry)
    real (kind=8), parameter :: rhot = 2*rhodrop-rhobottom !density of
        !top-layer fluid in equivalent settling drop problem
    real (kind=8), parameter :: rhob = 2*rhodrop-rhotop !density of
        !bottom-layer fluid in equivalent settling drop problem
    real (kind=8), parameter :: rhos = rhodrop !density of the drop
    real (kind=8)            :: U = 0.0  !initial velocity of drop
    real (kind=8), parameter :: g = 981.0 !gravity cm/s**2
    real (kind=8), parameter :: pi = 2.0*dasin(1.0)

!========Numerical Parameters============================================
    real (kind=8), parameter :: maxTime = 50. !time to run to
    real (kind=8), parameter :: dt = .1 !time step
    real (kind=8), parameter :: numtrapz = 0.005 !h for trapezoidal
        !integration trapz1
    real (kind=8), parameter :: integthres = 0.1E-5!simpson integration
    real (kind=8), parameter :: singthres = 0.1E-6 !sing. threshold x=y
    real (kind=8), parameter :: logsing = 0.1 !threshold log "sing" x=y
    real (kind=8), parameter :: logtrapzbig = 0.5 !h trapz for log term
        ! around "sing"
    real (kind=8), parameter :: logtrapz = 0.1 !htrapz for log
```

```
    real (kind= 8)           :: FFR = 2.*R !rad of full soln cm
    real (kind=8) ,parameter :: dx  = 0.1*R

!==========Non-Uniform Interface=========================================
    integer (kind=4)           ::  RorZ !, XN=ceiling(R0/dx)
    real (kind=8 )             ::  R0cl      = 2.*R
    integer (kind=4),parameter ::  XNfar     = 30, XNclose=40
    integer (kind=4)           ::  XN         = XNclose+XNfar

!====================Dependent Parameters================================
     real (kind=8)     :: ms               = 4.0/3.0*pi*R**3*rhos
        !mass of the sphere
    real (kind=8)    :: oneoversixpiamuK = (1-2.10444*(R/R0) +2.08877* &
           ((R/R0)**3))* 1.0/(6.0*pi*R*mu)
    real (kind=8)    :: stresspertcoeff = -0.25*g*(rhot-rhob)*R*2.0*pi
        !coefficient of the perturbatio stress
    !real (kind=8)    :: stresspert      = 0.0;   !perturbation stress
    real (kind=8)     :: buoyancytop     = -4.0/3.0*pi*R**3*g*rhot
          !buoyant force when sphere is above the interface
    real (kind=8)     :: buoyancybottom  = -4.0/3.0*pi*R**3*g*rhob
          !buoyant force when sphere is below the interface
    real (kind=8)     :: buoyancyCoeff1  = -pi*g/3.0*(rhob-rhot)
        !first coeff for the buoyant force when sphere is in interface
    real (kind=8)    :: buoyancyCoeff2  = -2.0*pi*g/3.0*R**3*(rhob+rhot)
        !second coeff for the buoyant force when sphere is in interface
    real (kind=8)     :: drhogover8mu    = (rhob-rhot)*g/(8.0*mu)
        !coefficient for the perturbation flow
    real (kind=8)     :: myt               = 0.
```

```
!===============Interface Interpolation Stuff==========================
    real (kind=8), dimension(:), allocatable :: x, y, sx, sy, su ,sv, &
                        &wu ,wv, cinterpx, cinterpy,cwinterpx, cwinterpy
    real (kind=8)     :: yend, xflagb, xflagl, px, py, myrho, myzeta, &
                         &xvalG2, yvalG2, xvalI2,yvalI2
    integer (kind=4)  :: flagb, flagu, flagl, flagt, cinternalcount,&
        & cwinternalcount
    !flagb is the index of the interface point where the backflow begins
!===============Fourier Components=====================================
    real (kind=8), parameter    :: epsilon = 0.001
    real (kind=8), parameter    :: LZ = 10.0
        !domain window of lambda for z-component of cylinder vel
    real (kind=8), parameter    :: LR = 10.0
        !domain window of lambda for R-component of cylinder vel
    integer (kind=4), parameter :: NZ = 2**14
        !number of discretization pts, z direction
    integer (kind=4), parameter :: NR = 2**16!
        !number of discretization pts, r direction
    real (kind=8), parameter    :: upperZ = 40.0;
    integer (kind=4), parameter :: upperRangeZ  = ceiling((LZ-epsilon) &
                                                 *upperZ/(2.0*pi)+1.0)
    integer (kind=4), parameter ::    upperRangeR = ceiling(LR*upperZ/ &
                                                    (2.0*pi)+1.0)
    real (kind=8)               :: cylindervelR(upperRangeR, XNclose + &
                  &XNfar+1), cylindervelZ(upperRangeZ, XNclose+XNfar+1)
    real (kind=8)               :: zcoordinateZ(upperRangeZ), &
                                    &zcoordinateR(upperRangeR)
    complex, parameter          :: MINUSONE = -1.0
    complex                     :: imagi = zSQRT(MINUSONE)
    real (kind=8)               :: WZ(NZ), WR(NR), myHZ(NZ), myGZ(NZ), &
```

```
         &myHR(NR), myGR(NR), firstpartZ(NZ), firstpartR(NR), myk(NZ)
    real (kind=8)               :: AreaReflux,AreaSpherePortion, &
        &AreaEntrain, startx(XNclose+XNfar+1), starty(XNclose+XNfar+1),&
        &wforceE, wforceR, wforce, ArchBouyancy,ArchBE, stresspert, &
        &stresspertA,ArchBR, stresspertE,stresspertReflux,stresspertG2, &
        &stresspertAG2, stresspertEG2,stresspertRefluxG2,stresspertI2, &
        &stresspertAI2, stresspertEI2,stresspertRefluxI2

!===============Stress Coefficient Calculator============================
    integer(kind=4), parameter  :: nsurfpoints  = 30!5!number of points
        !to use in integration over surface of drop for stress coeffs.
    real (kind=8), dimension(nsurfpoints+1) :: thetavecforstress, &
        surfx1, surfy1, surfx2,surfx3, surfy2, surfy3, wU1, wV1, wU2,&
        & wV2, wU3, wV3
        !to build the points on drop surface where w will be evaluated.
    real (kind=8)               :: eps  = .00000001
        !epsilon for stress calculation
    real (kind=8)               :: I2, G2, G4, G6, G8, G2New, G2HO,&
    &Unewforthirdref, v2atorigin, G2reciprocal, I2reciprocal
        !Stress Coeffs
end module globalinfo
```

Main Routine

Main time loop; calls other subroutines to evaluate velocities and advect the interface.

```
!----------------------------------------------------------------------
!Main Routine for Droplet
!----------------------------------------------------------------------
!
!PROGRAM: Full Simulation
```

```
!
!> @author
!> H. Arrowood, UNC-CH; C. Falcon, UNC-CH
!
!Description:
!> Main routine; finds interface, drop vel, etc at each time step
!-----------------------------------------------------------------------
program fullsimulation
    use globalinfo
    implicit none
    integer (kind=4)            :: i, ierr, flag, ix, iy, iv
        !indices for various do loops
    real (kind=8)               :: velocity(ceiling(maxTime/dt)+1), &
        stresspertvect(ceiling(maxTime/dt)+1), index, abserr=0.001, &
        relerr=0.001, xout, yout, xin1, yin1, xin2, yin2
        !store sphere velocity and stress due to perturbation vel at
        !each timestep
    real (kind=8), dimension(:), allocatable :: V, VP ! V stores
        !position of interface [xvals, yvals], VP stores velocities
        ![uvals, vvals]
    Character(len=256)           :: filename
    external rhoode !ODE solver for advecting interface

!----Initialize Interface-------------------------------------------------
     allocate(x(XNfar+XNclose+1), stat=ierr)
        if (ierr /= 0) print*, "x : Allocation failed"

    allocate(y(XNfar +XNclose+1), stat=ierr)
        if (ierr /= 0) print*, "y : Allocation failed"
```

```
    do i=1,XNclose + XNfar+1 !This actually defines the values of the x
        !values. More points closer to center where they'll be advected
        if (i<XNclose+2) then
            x (i) = (i-1)**(2) *(ROcl/XNclose**(2))
        else
            x( i ) = (i - XNclose-1)* (RO -ROcl ) / XNfar + ROcl
        endif
    end do
    y = y0  !initialize all y-values undisturbed -Holly.
    startx = x;     !save beginning interface
    starty = y;
!initialize interface interpolated spherevel
    allocate(cinterpx(1000000), stat=ierr)
         if (ierr /= 0) print*, "cintertpx : Allocation failed"
    allocate(cinterpy(1000000), stat=ierr)
        if (ierr /= 0) print*, "cintertpy : Allocation failed"
!initialize interface interpolated from w
    allocate(cwinterpx(1000000), stat=ierr)
        if (ierr /= 0) print*, "cintertpx : Allocation failed"
    allocate(cwinterpy(1000000), stat=ierr)
        if (ierr /= 0) print*, "cintertpy : Allocation failed"
!---------------keep track of how many interface points there are, for
!use in defining various arrays later -Holly----------------------------
    cinternalcount=XN+1;
    cwinternalcount=XN+1;
!------Initialize the interpolated interface points---------------------
    do ix=1, max(cinternalcount,cwinternalcount)
        cinterpx(ix)  = x(ix);
        cinterpy(ix)  = y(ix);
        cwinterpx(ix) = x(ix);
```

```
        cwinterpy(ix)= y(ix);
    end do
!--------Set up files to save entrainment volume and WForce-------------
    open (unit =9,file = 'VolumeTrack.dat')
        write(9,*) "Volume of Entrainment, Volume of Reflux, Volume of &
            &Portion of Sphere"
    open (unit =8,file = 'WForce.dat')
        write(8,*) " ArchBER, Wforce = 6pimuAstresscoeff(wFE - wFR), &
            &ArchBR,wforceR,wforceE"
!----------Compute spacial component of second reflection on a grid, to
!be used to interpolate second reflection values later on---------------
    call cylindervelinit()
!----------Initialize Stress Coefficients-------------------------------
    I2 = 0.0
    G2 = 0.0
    G4 = 0.0
    G6 = 0.0
    G8 = 0.0
    G2New = 0.0
    G2HO = 0.0
!---------------------------------------------------------------------------
!**********************  TIME LOOP ************************************
!---------------------------------------------------------------------------
    do index = 0,ceiling(maxTime/dt)

!allocate all the positions and velocities. XN changes at each timestep,
!so this has to be inside the time loop
    allocate(V(2*(XN+1)), stat=ierr)
        if (ierr /= 0) print*, "V : Allocation failed"
    allocate(VP(2*(XN+1)), stat=ierr)
```

```
        if (ierr /= 0) print*, "VP : Allocation failed"
!initialize stokes flow
    allocate(su(XN+1), stat=ierr)
        if (ierr /= 0) print*, "x : Allocation failed"
    allocate(sv(XN+1), stat=ierr)
        if (ierr /= 0) print*, "y : Allocation failed"
        su = 0.0
        sv = 0.0
!initialize w flow
    allocate(wu(XN+1), stat=ierr)
        if (ierr /= 0) print*, "x : Allocation failed"
    allocate(wv(XN+1), stat=ierr)
        if (ierr /= 0) print*, "y : Allocation failed"
        wu = 0.0
        wv = 0.0
!Stores the interface x and y coordinates in a single vector
        V(1:XN+1)        = x
        V(XN+2:2*(XN+1)) = y
!====================== Write  interface ================================
    write (filename, fmt='(a,f10.2,a)') 'interface',index+1,'.dat'
    open (unit =2,file = filename,form='formatted')
        write(2,*) "x,y at time=", myt, "rhos", rhos
        do ix=1, XN+1
            write(2,*), x(ix), ",", y(ix),","
        end do
!====================== Write interpolated interface ====================
    write (filename, fmt='(a,f10.2,a)') 'interpolation',index +1,'.dat'
    open (unit =6,file = filename,form='formatted')
!open (unit =6,file = 'interpolation.dat',form='formatted')
    write(6,*) "interpolated interface x,y at time=", myt
```

```
        do ix=1, max(cinternalcount,cwinternalcount)
            write(6,*), cinterpx(ix), ",", cinterpy(ix),",", &
            &cwinterpx(ix), ",", cwinterpy(ix),","
        end do
!====================== ODE SOLVER ====================================
    allocate(sx(XN+1), stat=ierr)
        if (ierr /= 0) print*, "sx : Allocation failed"
    allocate(sy(XN+1), stat=ierr)
        if (ierr /= 0) print*, "sy : Allocation failed"

    flag = 1
    print*, "about to call solver"
    call r8_rkf45 (rhoode, 2*(XN+1), V, VP, index*dt, (index+1.0)*dt, &
        &relerr, abserr, flag ) !this is solves the ode to advect the
        !interface, with the Runge Kutta method found in rk4.f90 -Holly
    print*, "solver done"
    x = V(1:XN+1)   !unwrap interface positions from V
    y = V(XN+2:2*(XN+1))
!===============Write data files=======================================
    write (filename, fmt='(a,f10.2,a)') 'stokes',index+1,'.dat'
    open (unit =4,file = filename,form='formatted')
        write(4,*) "us,sv at time=", myt
        do ix=1, XN+1
            write(4,*), su(ix), ",", sv(ix), ","
        end do
    write (filename, fmt='(a,f10.2,a)') 'wpert',index+1,'.dat'
    open (unit =5,file = filename,form='formatted')
        write(5,*) "wu,wv at time=", myt
        do ix=1, XN+1
            write(5,*), wu(ix), ",", wv(ix), ","
```

```
        end do
    open (unit =9,file = 'VolumeTrack.dat')
        write(9,*), AreaEntrain, ",", AreaReflux, ",", AreaSpherePortion
    open (unit =8,file = 'WForce.dat')
        write(8,*), ArchBE, ",", wforce, ",", ArchBouyancy, ",", Arch&
        &BR,",",wforceR,",", wforceE
    open (unit =11,file = 'sphereVel.dat')
        write(11,*), U, ",", myt, ",", yend

    call fillgaps() !add more interface points as needed.
    velocity(index+1) = U
    stresspertvect(index+1)=stresspert
!=====+==Deallocate velocities and positions============================
    if (allocated(V)) deallocate(V,stat=ierr)
    if (allocated(VP)) deallocate(VP,stat=ierr)

    if (allocated(wu)) deallocate(wu,stat=ierr)
    if (allocated(wv)) deallocate(wv,stat=ierr)

    if (allocated(su)) deallocate(su,stat=ierr)
    if (allocated(sv)) deallocate(sv,stat=ierr)

    if (allocated(sx)) deallocate(sx, stat=ierr)
    if (allocated(sy)) deallocate(sy, stat=ierr)

    end do
!======================End of Time Loop=================================
!=======Print velocities and stresses at the end=======================
    print *, "velocity"
        do iv = 1, ceiling(maxTime/dt)+1
```

```
            print *, velocity(iv), ","
        end do


    print *, "stress"
        do iv = 1, ceiling(maxTime/dt)+1
            print *, stresspertvect(iv), ","
        end do
!=======Print interface points at the end==============================
    print *, " "
    print *, "************************************************************"
    print *, "x"
        do ix=1, XN+1
            print *, x(ix), ","
        end do
    print *, "y"
        do iy=1, XN+1
            print *, y(iy), ","
        end do
    print *, "************************************************************"
    print *, " "
end program fullsimulation
```

Main Subroutine to Evaluate Velocities

Called by RKF45 solver to evaluate velocities as needed.

```
!---------------------------------------------------------------------
!> @author
!> H. Arrowood, UNC-CH; based on work of C. Falcon, UNC-CH
!
! DESCRIPTION:
!> Computes all necessary velocities on interface to feed into the Runge
```

```
!!Kutta Fehlberg solver that advects the interface
!>@return V   Points on new interface
!>@return VP  Velocities at interface points
!----------------------------------------------------------------------
subroutine rhoode(T, V, VP)
    use globalinfo
    implicit none

    real (kind=8)     :: T, sr(XN+1), V(2*(XN+1)), VP(2*(XN+1)), sunew, svnew
        !time, r-values of points, positions, velocities
    integer (kind=4)  :: ierr, i, ix, iy
        ! I don't know what ierr does exactly, but i, ix, and iy are
        !just various indices
    myt = T
    print*, "time", T

!-----Unwrap interface points from V-----------------------------------
    sx = V(1:XN+1)
    sy = V(XN+2:2*(XN+1))

!------This sets up the integrands for w calculation; determines where
!the perturbed interface intersects the original interface I think------

    call setSpecialPositions(sx, sy, XN, R, flagb, flagu, flagl, flagt,&
    &xflagb, xflagl, yend)
!------Find perturbation velocity---------------------------------------
    call wStressTN() !Finds stress coeffs using analytic derivative of
    !Oseen's Greens function.
    call wTN() !wu,wv
!------computes  coefficients of stresses, using velocities computed on
```

```
! the drop surface in wTN() (numerical derivative approach--------------
    call stresscoeffs()
!----Find velocity of sphere based on w, ustokes of previous timestep---
    call sphvel() !This finds drop velocity U
!-----------Find Stokes velocity in a cylinder, uses stress coeffs------
    call stokes() !su, sv
!------Set pert vel to zero if interface inside the drop----------------
    sr = dsqrt(sx**2+sy**2)
    do i = 1, XN+1
        if (sr(i) <= R*1.000000001) then
            wu(i) = 0.0
            wv(i) = 0.0
        elseif(i==XN+1) then
            wu(i) = 0.0
            wv(i) = 0.0
        print*, "fake velocity stuff happening"
        endif
    end do
!----Put together full velocities-------------------------------------
    VP(1:XN+1) = su+wu
    VP(XN+2:2*(XN+1)) = (sv+wv)
end subroutine rhoode
```

Drag and Drop Velocity

```
!----------------------------------------------------------------------
!> @author
!>H. Arrowood, UNC-CH, C. Falcon, UNC-CH
!
! DESCRIPTION:
!> Balances stokes and perturbation forces with buoyancy force to find
```

```
! velocity of drop
subroutine sphvel()
    use globalinfo
    implicit none
    real (kind=8) :: buoyancy, stressbelowsphere, stresssidesphere, &
        stressabovesphere, stressbackflow, stressbelowsphereA, &
        stresssidesphereA, stressabovesphereA, stressbackflowA, &
        stressbelowsphereE, stresssidesphereE, stressabovesphereE, &
        stressbackflowE,cindex, ArchBuoyancyDrop, IndirectI2
    real (kind=8) :: HabermanStressPert, stressdiff, Ureflections
    real (kind=8), external :: stresstail1D, stressIntegrandFlat1D,&
        stressIntegrandsphere1D, stressIntegrand1D,stresstail1DA, &
        stressIntegrandFlat1DA, stressIntegrandsphere1DA, &
        stressIntegrand1DA, stresstail1DE, stressIntegrandFlat1DE,&
        stressIntegrandsphere1DE, stressIntegrand1DE
    real (kind=8) :: stressbelowsphereG2, stresssidesphereG2, &
        stressabovesphereG2, stressbackflowG2, stressbelowsphereAG2, &
        stresssidesphereAG2, stressabovesphereAG2, stressbackflowAG2, &
        stressbelowsphereEG2, stresssidesphereEG2, stressabovesphereEG2, &
        stressbackflowEG2
    real (kind=8), external :: stresstail1DG2, stressIntegrandFlat1DG2,&
        stressIntegrandsphere1DG2, stressIntegrand1DG2,stresstail1DAG2, &
        stressIntegrandFlat1DAG2, stressIntegrandsphere1DAG2, &
        stressIntegrand1DAG2, stresstail1DEG2, stressIntegrandFlat1DEG2,&
        stressIntegrandsphere1DEG2, stressIntegrand1DEG2
    Character(len=256):: filename
    integer i,ix, ierr
!-----buoyancy for a two layer fluid; taking care of cases where drop is
!in the interface-----------------------------------------------------
    if (sy(XN+1)>=R) then
```

```
        buoyancy = buoyancybottom;
    elseif (abs(sy(XN+1))<R) then
        buoyancy = buoyancyCoeff1*(3.0*R**2*sy(XN+1)-sy(XN+1)**3)+ &
        buoyancyCoeff2;
    else
        buoyancy = buoyancytop;
    endif
!-----initialize interpolated interface points--------------------------
    if (allocated(cinterpx)) deallocate(cinterpx, stat=ierr)
    if (allocated(cinterpy)) deallocate(cinterpy, stat=ierr)
    allocate(cinterpx(1000000), stat=ierr)
        if (ierr /= 0) print*, "cintertpx : Allocation failed"
    allocate(cinterpy(1000000), stat=ierr)
        if (ierr /= 0) print*, "cintertpy : Allocation failed"
    cinternalcount=0.0;
    cinterpx =0;
    cinterpy = 0;
    !calculate stress force
    stresspert = 0.0;
    if (maxval(sy) > minval(sy)) then
        call setSpecialPositions(sx, sy, XN, R, flagb, flagu, flagl, &
            &flagt, xflagb, xflagl, yend)
        stressbelowsphere = 0.0;
        stresssidesphere = 0.0;
        stressabovesphere = 0.0;
        stressbackflow = 0.0;
        if  (flagu /= 0) then
            call trapz1(stressIntegrand1D, sy(1), max(-R, sy(1)),&
            & numtrapz, stressbelowsphere)
            call trapz1(stressIntegrandsphere1D, max(-R, sy(1)), R,&
```

```
            & numtrapz, stresssidesphere)
            call trapz1(stressIntegrand1D, R, yend, numtrapz, stressab&
                &ovesphere)

        elseif  (flagl /= 0) then
            call trapz1(stressIntegrand1D, sy(1), max(-R, sy(1)), &
            &numtrapz, stressbelowsphere)
            call trapz1(stressIntegrandsphere1D, max(-R, sy(1)), yend,&
            & numtrapz, stresssidesphere)
        else
            call trapz1(stressIntegrandFlat1D, sx(1), xflagb, numtrapz,&
            & stressbelowsphere)



        endif

        if (flagb /= XN+1) then
            call trapz1(stresstail1D, xflagb, sx(XN+1),real(0.01,kind&
            &=8), stressbackflow)
        endif
    stresspert  = stresspertcoeff*(stressbelowsphere + stresssidesphere&
    & + stressabovesphere -stressbackflow)
!------------Calculate G2-----------------------------------------------
!Uses similar integration routine as Claudia's above, but different integrand
        stresspertG2 = 0.0;
        stressbelowsphereG2 = 0.0;
        stresssidesphereG2 = 0.0;
        stressabovesphereG2 = 0.0;
        stressbackflowG2 = 0.0;
```

```
        if  (flagu /= 0) then
            call trapz1(stressIntegrand1DG2, sy(1), max(-R, sy(1)), &
            &numtrapz, stressbelowsphereG2)
            call trapz1(stressIntegrandsphere1DG2, max(-R, sy(1)), R, &
            &numtrapz, stresssidesphereG2)
            call trapz1(stressIntegrand1DG2, R, yend, numtrapz, &
            &stressabovesphereG2)

        elseif  (flagl /= 0) then
            call trapz1(stressIntegrand1DG2, sy(1), max(-R, sy(1)), &
            &numtrapz, stressbelowsphereG2)
            call trapz1(stressIntegrandsphere1DG2, max(-R, sy(1)), &
            &yend, numtrapz, stresssidesphereG2)
        else
            call trapz1(stressIntegrandFlat1DG2, sx(1), xflagb, numtrapz,&
            & stressbelowsphereG2)
        endif
        if (flagb /= XN+1) then
            call trapz1(stresstail1DG2, xflagb, sx(XN+1),real(0.01,kind&
            &=8), stressbackflowG2)
        endif
        stresspertG2  = 2.*pi*(rhotop-rhobottom)*(stressbelowsphereG2 +&
        & stresssidesphereG2&
        & + stressabovesphereG2 +stressbackflowG2)
    endif
G2reciprocal = (3./(4.*R**2*pi*mu))*stresspertG2
!-------calculate archimedean force of fluid----------------------------
    stresspertA = 0.0;
    if (maxval(sy) > minval(sy)) then
        call setSpecialPositions(sx, sy, XN, R, flagb, flagu, flagl, &
```

```
        &flagt, xflagb, xflagl, yend)
        stressbelowsphereA = 0.0;
        stresssidesphereA = 0.0;
        stressabovesphereA = 0.0;
        stressbackflowA = 0.0;
        if  (flagu /= 0) then
            call simp(stressIntegrand1DA, sy(1), max(-R, sy(1)), &
            &integthres, stressbelowsphereA)
            call simp(stressIntegrandsphere1DA, max(-R, sy(1)), R,&
            & integthres, stresssidesphereA)
            call simp(stressIntegrand1DA, R, yend, integthres, &
            &stressabovesphereA)
        elseif  (flagl /= 0) then
            call simp(stressIntegrand1DA, sy(1), max(-R, sy(1)), &
            &integthres, stressbelowsphereA)
            call simp(stressIntegrandsphere1DA, max(-R, sy(1)), yend, &
            &integthres, stresssidesphereA)
        else
            call simp(stressIntegrandFlat1DA, sx(1), xflagb, integth&
            &res, stressbelowsphereA)
        endif
        if (flagb /= XN+1) then
            call simp(stresstail1DA, xflagb, sx(XN+1), integthres, &
            &stressbackflowA)
        endif
    stresspertA  = -g*(rhob-rhot)*(stressbelowsphereA + stresssidesphe&
    &reA + stressabovesphereA-stressbackflowA)
    endif
!-----------calculate archimedean force of fluid-----------------------
    stresspertE = 0.0;
```

```
    if (maxval(sy) > minval(sy)) then
        stressbelowsphereE = 0.0;
        stresssidesphereE = 0.0;
        stressabovesphereE = 0.0;
        stressbackflowE = 0.0;
        if  (flagu /= 0) then
            call simp(stressIntegrand1DE, sy(1), max(-R, sy(1)), &
            &integthres, stressbelowsphereE)
            call simp(stressIntegrandsphere1DE, max(-R, sy(1)), R, &
            &integthres, stresssidesphereE)
            call simp(stressIntegrand1DE, R, yend, integthres, stress&
            &abovesphereE)
        elseif  (flagl /= 0) then
            call simp(stressIntegrand1DE, sy(1), max(-R, sy(1)), integ&
            &thres, stressbelowsphereE)
            call simp(stressIntegrandsphere1DE, max(-R, sy(1)), yend,&
            & integthres, stresssidesphereE)
        else
        call simp(stressIntegrandFlat1DE, sx(1), xflagb, integthres, &
        &stressbelowsphereE)
        endif
        if (flagb /= XN+1) then
            call simp(stresstail1DE, xflagb, sx(XN+1), integthres, &
            &stressbackflowE)
        endif
    stresspertE = g*(rhob-rhot)*(stressbelowsphereE + stresssidesphereE&
    & + stressabovesphereE-stressbackflowE)
    endif
!------Now put everything together-------------------------------------
    wforceE= stressbelowsphere + stresssidesphere + stressabovesphere
```

```
    wforceR=stressbackflow
    wforce = oneoversixpiamuK*stresspert
    ArchBouyancy =oneoversixpiamuK*(g*ms + buoyancy)
    ArchBE=-stresspertA *oneoversixpiamuK
    ArchBR =-g*(rhob-rhot)*(-stressbackflowA)*oneoversixpiamuK
    stresspertReflux= -g* ( rhob-rhot)*stressbackflowE
!------stresspert in the case of the sphere, for checking computation of
!G2 and I2. The stress due purely to the perturbation velocity should
!be the same in both cases, though of course the stokes stress is also
!dependent of the pert vel stress coeffs. So I can use this setup to
!indirectly compute I2 just as in the sphere case, then end by
!assembling all the pieces for the drop case----------------------------
    HabermanStressPert = 4.0*pi*R**2*(mu/3.0)*(G2-I2)
    IndirectI2 = G2reciprocal-3/(4*pi*R**2*mu)*stresspert!
    ArchBuoyancyDrop = (g*ms+buoyancy)

    G2 = -G2reciprocal
    I2 = -IndirectI2
    U = -((-ArchBuoyancyDrop + (2.0943951023931953d0*R**2.0*mu*((G2-2.0&
        &*I2)*mu+2.0*(G1.0*I2)*muin))/(mu+muin))/(-((2.09439510239319&
        &53d0*R*mu*(6.0*mu+9.0*muin))/(mu+muin))+(2.0*R**2*mu*(mu+1.5*m&
        &uin)*((6.515546450454206d0*R**2*muin)/mu+R0**2*(-8.56622642981&
        &1636d0-(12.849339644717457d0*muin)/mu)))/(R0**3*(mu+muin)*(1.0&
        &+muin/mu))))
!------Reflections velocity for comparison
    UReflections = -((4.1887902047863905d0*R**3*-981.0d0*(-rhobottom+rh&
        &os))/(-((12.566370614359172d0*R*mu*(mu+1.5*muin))/(mu +muin))+&
        &(2.0*R**2*mu*(mu+1.5*muin)*((6.515546450454206d0*R**2*muin)/mu&
        &+R0**2*(-8.566226429811636d0-(12.849339644717457d0*muin)/mu)))&
        &/(R0**3*(mu+muin)*(1.+muin/mu))))
```

```
    v2atorigin = 1./(2.*Pi)*(1./(R0**3*(1.+ (muin/mu)))*R*U*(R0**2*(-8.&
        &5662264298116352.849339644717457*(muin/mu))+6.51554645045420&
        &6*R**2*(muin/mu))) !this has been checked! -H.
    Unewforthirdref=U + v2atorigin
!----Sanity check printouts-------------------------------------------
    print*,"U", U
    print*,"URefBottom", UReflections
    print*, "URefTop", -((4.1887902047863905d0*R**3*-981.0d0*(-rhotop+r&
        &hos))/(-((12.566370614359172d0*R*mu*(mu+1.5*muin))/(mu +muin))&
        &+(2.0*R**2*mu*(mu+1.5*muin)*((6.515546450454206d0*R**2*muin)/m&
        &u+R0**2*(-8.566226429811636d0-(12.849339644717457d0*muin)/mu))&
        &)/(R0**3*(mu+muin)*(1.+muin/mu))))
!---terminal velocity for free space drop stuff, see p. 99 of notes-----
    !U = (mu+muin)/(2*mu+3*muin)*1/(2*R*pi*mu)*(ArchBuoyancyDrop-2*R*pi&
    !    &*mu/(3*(mu+muin))*(R*(mu*(-2*IndirectI2+G2)+2*muin*(-Indirect&
    !    &I2+G2))))
end subroutine sphvel

function stresstail1D ( xval ) !has been analytically integrated in y.
    use globalinfo
    implicit none
    real (kind=8) eta, xval, stresstail1D
    integer (kind=4) startingi

    startingi = max(flagb-2, 1)
    call interpbridge(XN+2-startingi, sx(startingi:XN+1), sy(startingi:&
        &XN+1), xval, eta)

     stresstail1D =-xval*(eta*(R**2-3.0*(eta**2+xval**2))/sqrt(eta**2+&
         &xval**2)**3)+xval*(yend*(R**2-3.0*(xval**2+yend**2)))/sqrt(xv&
```

```
          &al**2+yend**2)**3-xval*6.0*log(eta+sqrt(xval**2+eta**2))+xval&
          &*6.0*log(yend+sqrt(xval**2+yend**2))
     cinternalcount = cinternalcount + 1;
     cinterpy(cinternalcount) = eta;
     cinterpx(cinternalcount) = xval;

end

function stressIntegrandFlat1D ( xval )
     use globalinfo
     implicit none

     real (kind=8) xval, eta, stressIntegrandFlat1D
     integer (kind=4) endingi

     endingi = min(flagb+2, XN+1)
     call interpbridge( endingi, sx(1:endingi), sy(1:endingi), xval, eta)
      stressIntegrandFlat1D = -xval*(yend*(R**2-3.0*(yend**2+xval**2))/sqrt&
      &(yend**2+xval**2)**3)&
          +xval*(eta*(R**2-3.0*(xval**2+eta**2)))/sqrt(xval**2+eta**2)**3&
          &-6.0*log((yend+sqrt(xval**2+yend**2))**xval)+6.0*log((eta+sqrt&
          &(xval**2+eta**2))**xval)

cinternalcount = cinternalcount +1;
cinterpy(cinternalcount) = eta;
cinterpx(cinternalcount) = xval;
end

function stressIntegrandsphere1D(yval)
     use globalinfo
```

```
    implicit none

    real (kind=8) yval, eta, stressIntegrandsphere1D
    integer (kind=4) tempflagb

    tempflagb = min(flagb+2, XN+1)

    call interpbridge( tempflagb, sy(1:tempflagb), sx(1:tempflagb), yval, eta )
    stressIntegrandsphere1D = 2.0/R*(R**2-yval**2)-eta**2*(-R**2+3.0*&
    &(yval**2+eta**2))/(yval**2+eta**2)**(1.5)

    cinternalcount = cinternalcount +1;
    cinterpy(cinternalcount) = yval;
    cinterpx(cinternalcount) = eta;
    end



function stressIntegrand1D(yval)
    use globalinfo
    implicit none

    integer (kind=4) tempflagb
    real (kind=8) yval, eta, stressIntegrand1D

    tempflagb = min(flagb+2, XN+1)
    call interpbridge ( tempflagb, sy(1:tempflagb), sx(1:tempflagb),&
        & yval, eta )
    stressIntegrand1D =- eta**2*(-R**2+3.0*(yval**2+eta**2))/((yval**2+&
        &eta**2)**(1.5));
    cinternalcount = cinternalcount +1;
```

```
    cinterpy(cinternalcount) = yval;
    cinterpx(cinternalcount) = eta;
end
```

Interpolation Routines

```
!---------------------------------------------------------------------
!> @author
!> C. Falcon, UNC-CH
!
! DESCRIPTION:
!> Adds in more interface points afer each time step to fill gaps. Takes
!! in x and y values of interface points, returns new set of x and y
!! values with the gaps filled in using cubic spline interpolation.
!---------------------------------------------------------------------

subroutine fillgaps()
    use globalinfo
    implicit none

    real (kind=8), dimension(:), allocatable :: newx, newy, newx1, newy1
        !working x and y arrays
    real (kind=8)                :: theta, dist, newpt, nx, ny !distance
        !between consecutive interface points, new x/y val from spline
    integer (kind=4)                          :: internalcount, ierr, &
        &xi, posi, hi, ii, counter1, counter2 !various indices

    !initialize new interface
    allocate(newx(2*(XN+1)), stat=ierr)
        if (ierr /= 0) print*, "newx : Allocation failed"
    allocate(newy(2*(XN+1)), stat=ierr)
```

```
        if (ierr /= 0) print*, "newy : Allocation failed"

    call setSpecialPositions(x, y, XN, R, flagb, flagu, flagl, flagt, &
        &xflagb, xflagl, yend)

    internalcount   = 0.0

!---Check distances between consecutive points and add a point if larger
!than dx-----------------------------------------------------------------
       do xi=1,XN

            internalcount       = internalcount+1.0
            newx(internalcount) = x(xi)
            newy(internalcount) = y(xi)

            dist = sqrt((x(xi+1)-x(xi))**2+(y(xi+1)-y(xi))**2)
            theta = atan(x(xi)/y(xi))
!Two cases in which interpolation happens, either close to drop and
! dist>dx or far away and dist>R/2.
            if ((sqrt(x(xi)**2+y(xi)**2) < (2*R) .and. dist > dx) .or. &
                &dist >R/2) then
                if (sqrt(x(xi)**2+y(xi)**2) < R*1.05 .and. sqrt(x(xi+1)&
                    &**2+y(xi+1)**2) < R*1.05) then
                    print*, "interpolation very close to drop happening"
!Case1:
                       internalcount           = internalcount+1
                         !Call my interpolation routine instead of hers
                        print*, "Calling Newton Interpolation"
                        call NewtonInterpolation(x(xi), y(xi), x(xi+1),&
                        & y(xi+1), nx, ny)
```

```
            newx(internalcount) = nx!newpt
            newy(internalcount) = ny



else
        print*, "Claudia's interpolation happening"
        !cubic interpolation to find point to fill gap.
        internalcount           = internalcount+1
    if(x(xi) > 2.0*R) then
        newx(internalcount) = 0.5*(x(xi+1)+x(xi))
        posi = min(XN+1.0, xi+3.0)
        call interpbridge( 7, x(posi-6.0:posi), y(posi-&
            &6.0:posi), 0.5*(x(xi+1)+x(xi)), newpt)
        newy(internalcount) = newpt
    else
        print*, "interpolation close to drop happening"
            posi = max(xi-3.0, 1.0)
            if(flagl /= 0.0 .and. xi > flagl) then
                newy(internalcount) = 0.5*(y(xi+1)+y(xi))
    if (y(xi+1) >= y(xi)) then
        call interpbridge( 7, y(posi:posi+6.0), x(posi:&
        &posi+6.0), 0.5*(y(xi+1)+y(xi)), newpt)
    else
        call interpbridge( 7, y(posi+6.0:posi:-1.0), x(&
        &posi+6.0:posi:-1.0), 0.5*(y(xi+1)+y(xi)), newpt)
    endif
    if (newpt <= max(x(xi+1), x(xi)) .and. newpt >= min&
        &(x(xi+1), x(xi))) then
        newx(internalcount) = newpt
    elseif (((0.5*(y(xi+1)+y(xi)))**2 + (0.5*(x(xi+1)+x&
```

```
                    &(xi)))**2) > R**2) then
                    newx(internalcount) = 0.5*(x(xi+1)+x(xi))
                else
                    newx(internalcount) = sqrt((R+1E-5)**2 - (0.5*(&
                    &y(xi+1)+y(xi)))**2)
                endif
                else
                    newx(internalcount) = 0.5*(x(xi+1)+x(xi))
                    call interpbridge( 7, x(posi:posi+6.0), y(posi:&
                        &posi+6.0), 0.5*(x(xi+1)+x(xi)), newpt)

                    if (newpt <= max(y(xi+1), y(xi)) .and. newpt >=&
                        & min(y(xi+ 1), y(xi))) then
                        newy(internalcount) = newpt
                    elseif (((0.5*(y(xi+1)+y(xi)))**2 + (0.5*(x(xi+&
                        &1)+x(xi)))**2) > R**2) then
                        newy(internalcount) = 0.5*(y(xi+1)+y(xi))
                    else
                        newy(internalcount) = sqrt((R+1E-5)**2 - &
                        &(0.5*(x(xi+1)+x(xi)))**2)
                    endif
                endif
            endif
        endif
    endif

end do
newx(internalcount+1) = x(XN+1)
newy(internalcount+1) = y(XN+1)
```

```
    if (allocated(x)) deallocate(x,stat=ierr)
    if (allocated(y)) deallocate(y,stat=ierr)

    XN = internalcount
    allocate(x(XN+1), stat=ierr)
        if (ierr /= 0) print*, "fillgap - x : Allocation failed"

    allocate(y(XN+1), stat=ierr)
        if (ierr /= 0) print*, "fillgap - y : Allocation failed"

    x   = newx(1:internalcount+1)
    y   = newy(1:internalcount+1)

    if (allocated(newx)) deallocate(newx,stat=ierr)
    if (allocated(newy)) deallocate(newy,stat=ierr)
end subroutine fillgaps

subroutine interpbridge(N, interpx, interpy, xval, yval)
    use globalinfo
    implicit none
    ! In/out variables
    integer (kind=4), intent(in) :: N
    real (kind=8) :: interpx(N), interpy(N), xval
    real (kind=8), intent(out) :: yval
    ! Working variables
    integer (kind=4) :: setmin(1), mini, maxi, tempi, tempj, &
        interpchecki=1
    real (kind=8) :: d(N), checkorder(N-1)
    real (kind=8) :: checkmin, checkmax
```

```
!-------make first element of interpolated y vals the same as the first
! element of original values-----------------------------------------
    if (N == 1) then
        yval = interpy(1)
!-------otherwise, determine whether data are increasing or decreasing--
    else
    !checkmin will be positive if strictly increasing, negative otherwise
        checkorder = interpx(2:N) - interpx(1:N-1)
        checkmin = minval(checkorder)
        interpchecki = 1
        if (checkmin <= 0) then !decreasing input x vals
            mini = 1
            maxi = 1

            do interpchecki=1, N
             do tempi = maxi, N-1
                    if (checkorder(tempi) > 0) then !this would be the
                    !case if the input x vals increased somewhere
                        mini = tempi  !mini replaced with the index
                        !where x vals start to be increasing
                        maxi = N   !maxi bumped up to the last index val
                        do tempj = tempi, N-1 !check through all x vals
                        ! once they start increasing
                            if (checkorder(tempj) < 0) then !if starts
                            !decreasing somewhere
                                maxi = tempj !make the spot where starts
                                ! decreasing again the new maxi
                                exit !quit the do loop
                            endif
```

```
                end do
                exit !quit once we find the end of increasing
            endif !end the case where there's an increasing region
        end do !end the do loop checking if there's a region
        ! where things are increasing
        if (xval<=interpx(maxi) .and. xval>=interpx(mini)) then
            exit !check that the value at which interpolating
            !falls within the little increasing region just set;
            !if so, move on
        endif
    end do
else !so x vals strictly increasing
    mini = 1
    maxi = N
endif

if (xval > interpx(maxi) .or. xval < interpx(mini) .or. &
    &interpchecki == N) then
    if (xval>interpx(maxi)) then
        print *, "FATAL ERROR: x value greater than largest&
        & interpolation point"
    elseif (xval < interpx(mini)) then
        print *, "FATAL ERROR: x value less than smallest inter&
        &polation point"
    else
        print *, "FATAL ERROR: interpchecki", interpchecki, N
    endif
    print *, "time", myt, "xval", xval, "flagl", flagl, &
        &"flagb", flagb, "flagu", flagu
    print *, "interpx"
```

```
            do interpchecki = 1, N
                print *, interpx(interpchecki)
            end do
            print *, "interpy"
            do interpchecki = 1, N
                print *, interpy(interpchecki)
            end do
            print *, "x"
            do interpchecki = 1, XN+1
                print *, sx(interpchecki)
            end do
            print *, "y"
            do interpchecki = 1, XN+1
                print *, sy(interpchecki)
            end do

            print *, interpx(maxi), xval, interpx(mini)
            print*, "FATAL ERROR: interpolation failure, see diagnostic&
            &s above"
            stop
        endif
        call spline_pchip_set (maxi-mini+1, interpx(mini:maxi), &
            &interpy(mini:maxi), d)
        call spline_pchip_val (maxi-mini+1, interpx(mini:maxi), &
            &interpy(mini:maxi), d, 1, xval, yval)
    endif
end subroutine interpbridge
!-----------------------------------------------------------------------
!> @author
!> H. Arrowood, UNC-CH
```

```
!
! DESCRIPTION:
!> Uses Newton's method to find a new interface point on a trajectory
!> between the trajectories of the preceding and following points.
!> @brief
!> Interpolation routine designed with geometry of our problem in mind.
!
! REVISION HISTORY:
! 11_May_2018 - Implemented - H. Arrowood
!
!> @param[in] x1in,y1in,x2in,y2in
!> @return xout, yout
!-----------------------------------------------------------------------
subroutine NewtonInterpolation(x1in,y1in,x2in,y2in,xout,yout)
    use globalinfo
    implicit none

    real(kind=8), intent(in) :: x1in,y1in,x2in,y2in !Cartesian coordinates
        ! of input points
    real(kind=8)   :: r1, r2, theta1, theta2,thet!Polar coordinates of
        !input points
    real(kind=8) :: B2, D2,K, l, ri
    real(kind=8)  :: rnew, rnew1, rnew2, r1result, r2result, rguess,&
    & thetanew, elar(2), lout,rout, left, right
    integer(kind=4) :: i
    real(kind=8), external :: NewtonIter1,NewtonIter2, GegenbauerC
    real(kind=8) :: xout, yout !! final points

!----To begin, convert input x and y vals into r and theta vals----------
    r1 = dsqrt(x1in**2+y1in**2)
```

```
    theta1 = dacos(y1in/r1)
    r2 = dsqrt(x2in**2+y2in**2)
    theta2 = dacos(y2in/r2)
    thet=(theta1+theta2)/2.
    U=Unewforthirdref !to get through third ref, U=U+v2atorigin
!Find K value, the average of the streamfunction evaluated at x1 and x2
    B2 = -(R**3*(G2*mu*R + 3.*muin*U))/(6.*(mu + muin))
    D2 = -(-(G2*mu*R**2) - 6.*mu*R*U - 9.*muin*R*U)/(6.*(mu + muin))
    k = (GegenbauerC(2,theta1)*(-U*r1**2+B2*1/r1+D2*r1)+GegenbauerC(2,&
    & theta2)*(-U*r2**2+B2*1./r2+D2*r2))/2.
!Use Newton's iteration, applied 4 times to get machine precision under
!reasonable circumstances, to find rnew
    r1result=NewtonIter1(NewtonIter1(NewtonIter1(NewtonIter1(NewtonIter&
    &1(r1,k,B2, D2, thet),k,B2,D2,thet),k,B2,D2,thet),k,B2,D2,thet),k,B&
    &2,D2,thet)
    r2result= NewtonIter1(NewtonIter1(NewtonIter1(NewtonIter1(NewtonIte&
    &r1(r2,k,B2, D2, thet),k,B2,D2,thet),k,B2,D2,thet),k,B2,D2,thet),k,&
    &B2,D2,thet)
!Check that new point falls outside the drop
    if(r1result>=R)then
        rnew=r1result
    elseif(r2result>=R)then
        rnew=r2result
    else
        rnew=max(r1,r2)
    endif
!Convert rnew,thetanew into xout and yout
    xout = rnew*dsin(thet)
    yout = rnew*dcos(thet)
end subroutine NewtonInterpolation
```

```
!-----------------------------------------------------------------------
!> @author
!> H. Arrowood, UNC-CH
!
! DESCRIPTION:
!> Newton's method for the cubic function with roots equal to desired rnew
!> for the above subroutine to find a new interface point on a trajectory
!> between the trajectories of the preceding and following points.
!> @brief
!> Interpolation routine designed with geometry of our problem in mind.
!
! REVISION HISTORY:
! 11_May_2017 - Implemented - H. Arrowood
!
!> @param[in]  r
!> @return rout
!-----------------------------------------------------------------------



function NewtonIter1(rr, k, B2, D2,theta)

    use globalinfo
    implicit none

    real(kind=8), intent(in) :: rr, k, B2, D2, theta
    real(kind=8) :: f, fp
    real(kind=8) :: NewtonIter1
    real(kind=8), external :: GegenBauerC
```

```
    f=-U*rr**3+D2*rr**2-K/GegenbauerC(2,theta)*rr+B2
    fp=-3.*U*rr**2+2.*D2*rr-K/GegenbauerC(2,theta)

    NewtonIter1 = rr-f/fp
    return
end function NewtonIter1
```

Perturbation Velocity

Routine for dissection of displaced volume and routine for evaluation of perturbation velocity.

```
!---------------------------------------------------------------------
!> @author
!> H. Arrowood, UNC-CH; adapted from work of C. Falcon, UNC-CH
! DESCRIPTION:
!> Computes perturbation velocity using trapezoidal integration
!---------------------------------------------------------------------
subroutine wTN () !(wu, wv)
    use globalinfo
    implicit none

    real (kind = 8)    :: wbackflow(XN+2*nsurfpoints+3+nsurfpoints+1), &
        &wsidesphere(XN+2*nsurfpoints+3+nsurfpoints+1), wbelowsphere(XN&
        &+2*nsurfpoints+3+nsurfpoints+1), wabovesphere(XN+2*nsurfpoints&
        &+3+nsurfpoints+1),tempbackflow, tempsidesphere, tempbe&
        &lowsphere, tempabovesphere

    real (kind=8), external :: w2IntegrandBackflow, &
        &w2IntegrandBelowSphere, w2IntegrandPartialSphere,&
        &w2IntegrandZetaSphere, w2IntegrandZetaVert, w2IntegrandR, &
        &w2IntegrandZ
```

```
    integer (kind=4) wi, wi2, wi3
    real (kind=8) AEbelow,AEside,AEabove
    real (kind=8), external :: RefluxAreaFunction,EntrainPartialSphere,&
        &EntrainZetaSphere,EntrainZetaVert,EntrainBelowSphere, &
        &AreaElementZeta,AreaElementRho

    integer  ierr, I

!--------For stress calculator-----------------------------------------
!first initialize n evenly-spaced integration points on drop surface

    thetavecforstress=(/(Real(I),I=0,nsurfpoints)/)*pi/Real(nsurfpoints)

!x and y vectors at which to evaluate velocities for stress coeffs.
! At r=R+eps, r=R+2*eps, and r=R+3.*eps
    surfx1 = (R+eps)*sin(thetavecforstress)
    surfy1 = (R+eps)*cos(thetavecforstress)
    surfx2 = (R+2.0*eps)*sin(thetavecforstress)
    surfy2 = (R+2.0*eps)*cos(thetavecforstress)
    surfx3 = (R+3.0*eps)*sin(thetavecforstress)
    surfy3 = (R+3.0*eps)*cos(thetavecforstress)

    wbackflow    = real(0.0, kind=8)
    wsidesphere     = real(0.0, kind=8)
    wbelowsphere     = real(0.0, kind=8)
    wabovesphere     = real(0.0, kind=8)
    AreaReflux      = real(0.0, kind=8)
    AreaEntrain     = real(0.0, kind=8)

!----Find entrainment and reflux volumes-------------------------------
```

```
    if(flagb <XN+1 ) then
        call simp(RefluxAreaFunction, xflagb, sx(XN+1), integthres, &
            &AreaReflux)
    endif
    if (flagu /= 0) then
        AreaSpherePortion= (4./3.*pi*R**3);
        call simp(EntrainPartialSphere,real(0.0,kind=8),xflagl,&
            &integthres,AEbelow)
        call simp(EntrainZetaSphere, max(-R, sy(1)), R, integthres, &
            &AEside)
        call simp(EntrainZetaVert, R, yend, integthres, AEabove)
        AreaEntrain=AEbelow+AEside+AEabove;
    elseif (flagl /= 0) then
        !find area of the sphere
        AreaSpherePortion= (pi*(yend+R)**2/3.)*(3.*R - (yend+R));
        call simp(EntrainPartialSphere, real(0.0, kind=8), xflagl, &
            &integthres,AEbelow)
        call simp(EntrainZetaSphere, max(-R, sy(1)), yend, integthres, &
            &AEside)
        AreaEntrain=AEbelow+AEside;
    else
        AreaSpherePortion= 0.0;
        call simp(EntrainBelowSphere, max(sx(1), real(0.0, kind=8)), &
            &xflagb, integthres, AEbelow)
        AreaEntrain=AEbelow;
    endif
!========== Find perturbation velocity w at interface points =========!
do RorZ = 0, 1
    if (minval(sy) < maxval(sy)) then !check that interface is perturbed
!----------------Initialize Interpolated Interface----------------------
```

```
if (allocated(cwinterpx)) deallocate(cwinterpx, stat=ierr)
if (allocated(cwinterpy)) deallocate(cwinterpy, stat=ierr)
allocate(cwinterpx(1000000), stat=ierr)
if (ierr /= 0) print*, "cwinterpx : Allocation failed"
allocate(cwinterpy(1000000), stat=ierr)
if (ierr /= 0) print*, "cwinterpy : Allocation failed

do wi = 1, XN+3*nsurfpoints+4
    if (wi<XN+2) then
        px = sx(wi)
        py = sy(wi)
    elseif (wi>XN+1 .and. wi<XN+3+nsurfpoints) then
        px = surfx1(wi-XN-1)
        py = surfy1(wi-XN-1)
    elseif (wi>XN+2+nsurfpoints .and. wi<XN+1+2*nsurfpoints+3) then
        px = surfx2(wi-XN-nsurfpoints-2)
        py = surfy2(wi-XN-nsurfpoints-2)
    else
        px = surfx3(wi-XN-2*nsurfpoints-3)
        py = surfy3(wi-XN-2*nsurfpoints-3)
    endif
    cwinternalcount=0.0;
    cwinterpx = 0;
    cwinterpy = 0;

    tempbackflow    = real(0.0, kind=8)
    tempsidesphere     = real(0.0, kind=8)
    tempbelowsphere    = real(0.0, kind=8)
    tempabovesphere    = real(0.0, kind=8)
```

```
            if(flagb <XN+1 ) then
            call trapz1(w2IntegrandBackflow, xflagb, sx(XN+1), &
                &numtrapz, tempbackflow)
            endif
            if (flagu /= 0) then
                call trapz1(w2IntegrandPartialSphere, real(0.0, kind=8), &
                    &xflagl, numtrapz, tempbelowsphere)
                call trapz1(w2IntegrandZetaSphere, max(-R, sy(1)), R, &
                    &numtrapz, tempsidesphere)
                call trapz1(w2IntegrandZetaVert, R, yend, numtrapz, &
                    &tempabovesphere)
            elseif (flagl /= 0) then
                call trapz1(w2IntegrandPartialSphere, real(0.0, kind=8), &
                    &xflagl, numtrapz, tempbelowsphere)
                call trapz1(w2IntegrandZetaSphere, max(-R, sy(1)), yend, &
                    &numtrapz, tempsidesphere)
            else
                call trapz1(w2IntegrandBelowSphere, max(sx(1), real(0.0, &
                    &kind=8)), xflagb, numtrapz, tempbelowsphere)
            endif
            wbackflow(wi)         = tempbackflow
            wbelowsphere(wi)     = tempbelowsphere
            wsidesphere(wi)     = tempsidesphere
            wabovesphere(wi)     = tempabovesphere
        end do
    endif

!----Break things back up into vectors for interface and drop surface---
    if (RorZ > 0) then
        do wi2=1, XN+1+3*nsurfpoints+3
```

```
        if (wi2<XN+2) then
            wu(wi2) = (-wbackflow(wi2)+wbelowsphere(wi2)+ &
                &wsidesphere(wi2)+ wabovesphere(wi2))*drhogover8mu
        elseif (wi2>XN+1 .and. wi2<XN+nsurfpoints+3) then
            wU1(wi2-XN-1) = (-wbackflow(wi2)+wbelowsphere(wi2)+ &
                &wsidesphere(wi2)+wabovesphere(wi2))*drhogover8mu
        elseif (wi2>XN+1+nsurfpoints+1 .and. wi2<XN+1+2*nsurfpoints+3) then
            wU2(wi2-XN-1-nsurfpoints-1) = (-wbackflow(wi2)+ &
                &wbelowsphere(wi2)+wsidesphere(wi2)+ &
                &wabovesphere(wi2))*drhogover8mu

        else
            wU3(wi2-XN-1-nsurfpoints-1-nsurfpoints-1) = (-wbackflow(wi2)+ &
                &wbelowsphere(wi2)+wsidesphere(wi2)+ &
                &wabovesphere(wi2))*drhogover8mu
        endif
    end do
else
    do wi3=1,XN+3*nsurfpoints+4
        if (wi3<XN+2) then
            wv(wi3) = (-wbackflow(wi3)+wbelowsphere(wi3)+ &
                &wsidesphere(wi3)+wabovesphere(wi3))*drhogover8mu
        elseif (wi3>XN+1 .and. wi3<XN+1+nsurfpoints+2) then
            wV1(wi3-XN-1) = (-wbackflow(wi3)+wbelowsphere(wi3)+ &
                &wsidesphere(wi3)+wabovesphere(wi3))*drhogover8mu
        elseif (wi3>XN+2+nsurfpoints .and. wi3<XN+4+2*nsurfpoints) then
            wV2(wi3-XN-1-nsurfpoints-1) = (-wbackflow(wi3)+ &
                &wbelowsphere(wi3)+wsidesphere(wi3)+ &
                &wabovesphere(wi3))*drhogover8mu
        else
```

```
                wV3(wi3-XN-1-nsurfpoints-1-nsurfpoints-1) = (-wbackflow(wi3)+ &
                    &wbelowsphere(wi3)+wsidesphere(wi3)+ &
                    &wabovesphere(wi3))*drhogover8mu
            endif
        end do
    endif
end do
    if(abs(wbelowsphere(1)) > 1000.0) then
    stop
    endif
end subroutine wTN
!========== FOR VOLUME TRACK =========!
function AreaElementRho (rho)
    use globalinfo
    implicit none
    real (kind =8) rho, AreaElementRho
    AreaElementRho = 2*pi*rho;
    return
end function AreaElementRho

function AreaElementZeta (zeta)
    use globalinfo
    implicit none
    real (kind =8) zeta, AreaElementZeta

    AreaElementZeta = 2*pi*myrho;
    return
end function AreaElementZeta

function RefluxAreaFunction(rho)
```

```
    use globalinfo
    implicit none

    real (kind=8) rho, zcoord, RefluxAreaFunction
    real (kind=8), external :: AreaElementZeta

    call interpbridge( XN+2-max(flagb-2, 1), sx(max(flagb-2,1):XN+1),&
        &sy(max(flagb-2, 1):XN+1), rho, zcoord)

    myrho   = rho
    call simp2(AreaElementZeta, yend, zcoord, integthres, RefluxAreaF&
        &unction)
end function RefluxAreaFunction

function EntrainZetaSphere(zeta)
    use globalinfo
    implicit none

    real (kind=8) zeta, xupper, xlower, EntrainZetaSphere
    real (kind=8), external :: AreaElementRho

    myzeta  = zeta
    call interpbridge(min(flagb+2, XN+1), sy(1:min(flagb+2, XN+1)), &
        &sx(1:min(flagb+2, XN+1)), zeta, xupper)

    xlower  = sqrt(R**2 - zeta**2)

    call simp2(AreaElementRho, xlower, xupper, integthres, EntrainZeta&
        &Sphere)
end function EntrainZetaSphere
```

```
function EntrainPartialSphere(rho)
    use globalinfo
    implicit none

    real (kind=8) rho, zcoord, EntrainPartialSphere
    real (kind=8), external :: AreaElementZeta

    myrho = rho
    call interpbridge(min(flagl+3, XN+1), sx(1:min(flagl+3, XN+1)),&
        &sy(1:min(flagl+3, XN+1)), rho, zcoord)



    call simp2(AreaElementZeta, zcoord, -R, integthres, EntrainPartial&
        &Sphere)

end function EntrainPartialSphere

function EntrainZetaVert(zeta)
    use globalinfo
    implicit none

    real (kind=8) zeta, xupper, EntrainZetaVert
    real (kind=8), external :: AreaElementRho
    integer (kind=4) temp(1), tempmini

    myzeta  = zeta
    call interpbridge(min(flagb+2, XN+1), sy(1:min(flagb+2, XN+1)),&
        &sx(1:min(flagb+2, XN+1)), zeta, xupper)
```

```
    call simp2(AreaElementRho, real(0.0, kind=8), xupper, integthres,&
        &EntrainZetaVert)

end function EntrainZetaVert

function EntrainBelowSphere(rho)
    use globalinfo
    implicit none

    real (kind=8) rho, zcoord, EntrainBelowSphere
    real (kind=8), external :: AreaElementZeta



    call interpbridge(max(flagl+2, XN+1), sx(1:max(flagl+2, XN+1)), &
        &sy(1:max(flagl+2, XN+1)), rho, zcoord)

    myrho   = rho
    call simp2(AreaElementZeta, zcoord, yend, integthres, EntrainBelow&
        &Sphere)
end function EntrainBelowSphere
!========== end of functions for volume tracking=========!

function w2IntegrandBackflow(rho)
    use globalinfo
    implicit none
    real (kind=8) rho, zcoord, w2IntegrandBackflow
    real (kind=8), external :: w2IntegrandZeta

    call interpbridge( XN+2-max(flagb-2, 1), sx(max(flagb-2,1):XN+1),&
        &sy(max(flagb-2, 1):XN+1), rho, zcoord)
```

```
    myrho   = rho
    call trapz1(w2IntegrandZeta, yend, zcoord, real(min(0.001,numtrapz)&
        &,kind=8), w2IntegrandBackflow)
    cwinternalcount = cwinternalcount +1;
    cwinterpx(cwinternalcount) = rho;
    cwinterpy(cwinternalcount) = zcoord;
end function w2IntegrandBackflow

function w2IntegrandZetaSphere(zeta)
    use globalinfo
    implicit none
    real (kind=8) zeta, xupper, xlower, w2IntegrandZetaSphere
    real (kind=8), external :: w2IntegrandRho
    myzeta  = zeta
    call interpbridge(min(flagb+2, XN+1), sy(1:min(flagb+2, XN+1)),&
        &sx(1:min(flagb+2, XN+1)), zeta, xupper)
    xlower  = dsqrt(R**2 - zeta**2)
    !Catch NaN that seems to be happening when zeta**2~R**2 -Holly
    if(isnan(xlower))then
    xlower = 0.0
    endif
    call trapz1(w2IntegrandRho, xlower, xupper, numtrapz, &
        &w2IntegrandZetaSphere)

    cwinternalcount = cwinternalcount +1;
    cwinterpx(cwinternalcount) = xupper;
    cwinterpy(cwinternalcount) = zeta;
end function w2IntegrandZetaSphere

function w2IntegrandPartialSphere(rho)
```

```
    use globalinfo
    implicit none
    real (kind=8) rho, zcoord, w2IntegrandPartialSphere
    real (kind=8), external :: w2IntegrandZeta

    myrho = rho
    call interpbridge(min(flagl+3, XN+1), sx(1:min(flagl+3, XN+1)),&
        &sy(1:min(flagl+3, XN+1)), rho, zcoord)
    call trapz1(w2IntegrandZeta, zcoord, -R, numtrapz, w2IntegrandParti&
        &alSphere)
    cwinternalcount = cwinternalcount +1;
    cwinterpx(cwinternalcount) = rho;
    cwinterpy(cwinternalcount) = zcoord;
end function w2IntegrandPartialSphere

function w2IntegrandZetaVert(zeta)
    use globalinfo
    implicit none

    real (kind=8) zeta, xupper, w2IntegrandZetaVert
    real (kind=8), external :: w2IntegrandRho
    integer (kind=4) temp(1), tempmini

    myzeta  = zeta

    call interpbridge(min(flagb+2, XN+1), sy(1:min(flagb+2, XN+1)), &
        &sx(1:min(flagb+2, XN+1)), zeta, xupper)

    call trapz1(w2IntegrandRho, real(0.0, kind=8), xupper, numtrapz,&
        &w2IntegrandZetaVert)
```

```
    cwinternalcount = cwinternalcount +1;
    cwinterpx(cwinternalcount) = xupper;
    cwinterpy(cwinternalcount) = zeta;
end function w2IntegrandZetaVert


function w2IntegrandBelowSphere(rho)
    use globalinfo
    implicit none
    real (kind=8) rho, zcoord, w2IntegrandBelowSphere
    real (kind=8), external :: w2IntegrandZeta
    call interpbridge(max(flagl+2, XN+1), sx(1:max(flagl+2, XN+1)), &
        &sy(1:max(flagl+2, XN+1)), rho, zcoord)
    myrho   = rho
    call trapz1(w2IntegrandZeta, zcoord, yend, numtrapz, &
        &w2IntegrandBelowSphere)
    cwinternalcount = cwinternalcount +1;
    cwinterpx(cwinternalcount) = rho;
    cwinterpy(cwinternalcount) = zcoord;
end function w2IntegrandBelowSphere

function w2IntegrandZeta(zeta)
    use globalinfo
    implicit none

    real (kind=8) zeta, w2IntegrandZeta
    real (kind=8), external :: w2IntegrandR, w2IntegrandZ, &
        &w2IntegrandRFF, w2IntegrandZFF
    real (kind=8) ellipticE, ellipticK,ellipticE1, ellipticK1
```

```
DOUBLE PRECISION k,k1, kbar, tempK, tempE, DRF, DRD, ex, ey, ez
integer ier
myzeta = zeta



if ( ((py-myzeta)**2+(px-myrho)**2)>singthres) then
    !============================================================
    !inputs for ellipticK, ellipticE,ellipticK1, and ellipticE1
    !============================================================
    k       = 4.0*px*myrho/((py-myzeta)**2+(px-myrho)**2)
    kbar    = k/(k+1.0)
    ex = 0.0
    ey = 1.0-kbar
    ez = 1.0
    tempK     = DRF(ex, ey, ez, ier)
    tempE     = tempK-1.0/3.0*kbar*DRD(ex, ey, ez, ier)
    ellipticE    = sqrt(1.0+k)*tempE
    ellipticK    = (1.0/sqrt(1.0+k))*tempK
endif
k1 = -((-4.0)*R**2.0*px*myrho*(R**4.0+(-2.0)* R**2.0*(px*myrho+py*&
    &myzeta)+(px**2.0+py**2.0)*(myrho**2.0+myzeta**2.0))**(-1.0))
!print*, k1
kbar    = k1/(k1+1.0)
ex = 0.0
ey = 1.0-kbar
ez = 1.0
tempK     = DRF(ex, ey, ez, ier)
tempE     = tempK-1.0/3.0*kbar*DRD(ex, ey, ez, ier)
ellipticE1    = sqrt(1.0+k1)*tempE
ellipticK1    = (1.0/sqrt(1.0+k1))*tempK
```

```
    if (RorZ >0.0) then
        if (px**2+py**2 >= (FFR)**2) then
            w2IntegrandZeta= w2IntegrandRFF()
        else
            w2IntegrandZeta =w2IntegrandR(ellipticK,ellipticE,elliptic&
                &K1,ellipticE1)
        endif
    else
        if (px**2+py**2 >= (FFR)**2) then
            w2IntegrandZeta = w2IntegrandZFF()
        else
        w2IntegrandZeta =w2IntegrandZ(ellipticK,ellipticE,ellipticK1,&
            &ellipticE1)
        endif
    endif
end function w2IntegrandZeta




function w2IntegrandRho(rho)
    use globalinfo
    implicit none
    real (kind=8), intent(in) :: rho
    real(kind=8) ::w2IntegrandRho
    real (kind=8), external :: w2IntegrandR, w2IntegrandZ, w2Integrand&
        &RFF, w2IntegrandZFF
    real (kind=8) ellipticE, ellipticK,ellipticE1, ellipticK1
    real (kind=8) k, k1, kbar, tempK, tempE, DRF, DRD, ex, ey, ez
    integer ier
```

```
myrho = rho
if ( ((py-myzeta)**2+(px-myrho)**2)>singthres) then
    !=====================================================
    ! inputs for ellipticK, ellipticE,ellipticK1, and ellipticE1
    !=====================================================
    k       = 4.0*px*myrho/((py-myzeta)**2+(px-myrho)**2)
    kbar    = k/(k+1.0)
    ex = 0.0
    ey = 1.0-kbar
    ez = 1.0
    tempK     = DRF(ex, ey, ez, ier)
    tempE     = tempK-1.0/3.0*kbar*DRD(ex, ey, ez, ier)
    ellipticE    = sqrt(1.0+k)*tempE
    ellipticK    = (1.0/sqrt(1.0+k))*tempK
endif
k1 = -((-4.0)*R**2*px*myrho*(R**4.0+(-2.0)* &
R**2.0*(px*myrho+py*myzeta)+(px**2.0+py**2.0)*(myrho**2.0+myzeta&
    &**2.0))**(-1.0));
kbar    = k1/(k1+1.0)
ex = 0.0
ey = 1.0-kbar
ez = 1.0
tempK     = DRF(ex, ey, ez, ier)
tempE     = tempK-1.0/3.0*kbar*DRD(ex, ey, ez, ier)
ellipticE1    = sqrt(1.0+k1)*tempE
ellipticK1    = (1.0/sqrt(1.0+k1))*tempK
if (RorZ >0.0) then
    if (px**2+py**2 >= (FFR)**2) then
        w2IntegrandRho = w2IntegrandRFF()
    else
```

```
            w2IntegrandRho = w2IntegrandR(ellipticK,ellipticE,elliptic&
                &K1, ellipticE1)
        endif
    else
        if (px**2+py**2 >= (FFR)**2) then
            w2IntegrandRho = w2IntegrandZFF()
        else
            w2IntegrandRho = w2IntegrandZ(ellipticK,ellipticE,elliptic&
                &K1, ellipticE1)
        endif
    endif
end function w2IntegrandRho

!========== Far Field kernel for px^2+py^2 > 4R^2 ====================!
function w2IntegrandRFF()
    use globalinfo
    implicit none
    real (kind=8) w2IntegrandRFF, ellipticE, ellipticK
    DOUBLE PRECISION k, kbar, tempK, tempE, DRF, DRD, ex, ey, ez
    integer ier

    w2IntegrandRFF = 0.0
    if (px > 0.0 .and. ((py-myzeta)**2+(px-myrho)**2)>singthres) then
        k       = 4.0*px*myrho/((py-myzeta)**2+(px-myrho)**2)
        kbar    = k/(k+1.0)
        ex = 0.0
        ey = 1.0-kbar
        ez = 1.0
        tempK     = DRF(ex, ey, ez, ier)
        tempE     = tempK-1.0/3.0*kbar*DRD(ex, ey, ez, ier)
```

```
        ellipticE    = sqrt(1.0+k)*tempE
        ellipticK    = (1.0/sqrt(1.0+k))*tempK

        w2IntegrandRFF = 2.0*(py-myzeta)/(pi*px*sqrt((px-myrho)**2+(py-&
            &myzeta)**2)*((px+myrho)**2+(py-myzeta)**2))*((px**2-myrho&
            &**2-(py-myzeta)**2)*ellipticE+((myrho+px)**2+(py-myzeta)&
            &**2)*ellipticK)-3.0*R*px*py*(2.0*myzeta**2+myrho**2)/(2.0*&
            &sqrt(px**2+py**2)**3*sqrt(myzeta**2+myrho**2)**3)+(R**3*px&
            &*(px**2*(py + 5.0*myzeta)* (2.0*myzeta**2 - myrho**2)+ py*&
            &(py**2*(2.0*myzeta**2 - myrho**2) + 10.0*py*myzeta*(-2.0*&
            &myzeta**2 + myrho**2)+3.0*(2.0*myzeta**4 + 3.0*myzeta**2*&
            &myrho**2 + myrho**4))))/(2.0*(px**2 + py**2)**(2.5)*(myzet&
            &a**2 + myrho**2)**(2.5)) - (3.0*R**5*px*(8.0*px**4*(2.0*&
            &myzeta**3 - 3.0*myzeta*myrho**2) +px**2* (-8.0*py**2*(2.0*&
            &myzeta**3 - 3.0*myzeta*myrho**2) + py*(-136.0*myzeta**4 + &
            &296.0*myzeta**2*myrho**2 - 23.0*myrho**4) + 8.0*myzeta*&
            &(2.0*myzeta**4 + myzeta**2*myrho**2 - myrho**4)) -4.0*py&
            &**2*(4.0*py**2*(2.0*myzeta**3 - 3.0*myzeta*myrho**2) + &
            &8.0*myzeta*(2.0*myzeta**4 + myzeta**2*myrho**2 - myrho**4)&
            &- 3.0*py*(12.0*myzeta**4 - 22.0*myzeta**2* myrho**2 +&
            & myrho**4))))/(16.0*(px**2 + py**2)**(3.5)*(myzeta**2 + &
            &myrho**2)**(3.5))

        w2IntegrandRFF = w2IntegrandRFF*myrho
    endif
end function w2IntegrandRFF

function w2IntegrandZFF()
    use globalinfo
    implicit none
```

```
real (kind=8) w2IntegrandZFF, ellipticE, ellipticK
DOUBLE PRECISION k, kbar, tempK, tempE, DRF, DRD, ex, ey, ez
integer ier

w2IntegrandZFF = 0.0
if (((py-myzeta)**2+(px-myrho)**2)>singthres) then
k       = 4.0*px*myrho/((py-myzeta)**2+(px-myrho)**2)
kbar    = k/(k+1.0)
ex = 0.0
ey = 1.0-kbar
ez = 1.0
tempK     = DRF(ex, ey, ez, ier)
tempE     = DRF(ex, ey, ez, ier)-1.0/3.0*kbar*DRD(ex, ey, ez, ier)
ellipticE    = sqrt(1.0+k)*tempE
ellipticK    = (1.0/sqrt(1.0+k))*tempK
w2IntegrandZFF = 4.0*((py-myzeta)**2*ellipticE+((px+myrho)**2+(py-&
    &myzeta)**2)*ellipticK)/(pi*sqrt((px-myrho)**2+(py-myzeta)**2)*&
    &((px+myrho)**2+(py-myzeta)**2))-3.0*R*(px**2+2.0*py**2)*(2.0*&
    &myzeta**2+myrho**2)/(2.0*sqrt(px**2+py**2)**3*sqrt(myzeta**2+&
    &myrho**2)**3)-R**3/(2.0*sqrt(px**2+py**2)**5*sqrt(myrho**2+&
    &myzeta**2)**5)*(px**4*(-2.0*myzeta**2 + myrho**2) - 2.0*py**2*&
    &(2.0*myzeta**4 + 3.0*myzeta**2*myrho**2+ myrho**4 + py**2*(2.0&
    &*myzeta**2 - myrho**2) + 5.0*py*myzeta*(-2.0*myzeta**2 + myrho&
    &**2))+ px**2*(2.0*myzeta**4 + 3.0*myzeta**2*myrho**2 + myrho&
    &**4 + 5.0*py*myzeta*(-2.0*myzeta**2 + myrho**2)+py**2* (-6.0*&
    &myzeta**2 + 3.0*myrho**2)))-(3.0*R**5*(px**4*(8.0*myzeta**4 -&
    & 24.0*myzeta**2*myrho**2 + 3.0*myrho**4 + 8.0*py*(2.0*myzeta**&
    &3- 3.0*myzeta*myrho**2)) - 8.0*py**3*(4.0*myzeta**5 + 2.0*&
    &myzeta**3*myrho**2 - 2.0*myzeta*myrho**4+ py**2*(4.0*myzeta**3&
    & - 6.0*myzeta*myrho**2) - py*(12.0*myzeta**4 - 22.0*myzeta**2*&
```

```
        & myrho**2+ myrho**4)) - 8.0*px**2*py*(-6.0*myzeta**5 - 3.0*&
        &myzeta**3* myrho**2 + 3.0*myzeta* myrho**4    + py**2* (2.0* &
        &myzeta**3 - 3.0* myzeta* myrho**2) + py* (22.0*myzeta**4 - &
        &45.0*myzeta**2* myrho**2 + 3.0*myrho**4))))/(16.0*(px**2 + &
        &py**2)**(3.5)* (myzeta**2 + myrho**2)**(3.5))
    w2IntegrandZFF = w2IntegrandZFF*myrho
    endif
end function w2IntegrandZFF

!========== Far Field kernel for px^2+py^2 > 4R^2 =====================!
function w2IntegrandR(ellipticK,ellipticE,ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) w2IntegrandR,ellipticE, ellipticK,ellipticE1, &
        &ellipticK1,ILogR
    real (kind=8), external :: I1R,I2R,I3R,I4R,I5R,I6R,I7R,I8R,&
        &LogTermThetaR
    w2IntegrandR=0.0;
    ILogR = 0.0;
    if (px > 0.0) then
        if (abs (myzeta*px + py * myrho ) < logsing) then
            call trapz1(LogTermThetaR,  real(0.0,kind=8),real(2.0*pi,&
                &kind=8), logtrapzbig, ILogR)
        else
            call trapz1(LogTermThetaR,  real(0.0,kind=8),real(2.0*pi,&
                &kind=8), logtrapz, ILogR)
        endif
        w2IntegrandR = I1R(ellipticK,ellipticE)+I2R(ellipticK1,&
            &ellipticE1)+I3R(ellipticK1,ellipticE1)+I4R(ellipticK1,&
            &ellipticE1)+I5R(ellipticK1,ellipticE1)+I6R(ellipticK1,&
```

```
            &ellipticE1)+I7R(ellipticK1,ellipticE1)+I8R(ellipticK1,&
            &ellipticE1)+ILogR
        w2IntegrandR = w2IntegrandR*myrho/pi
    endif
end function w2IntegrandR

function w2IntegrandZ(ellipticK, ellipticE,ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) w2IntegrandZ,ellipticE, ellipticK,ellipticE1, &
        &ellipticK1,ILogZ
    real (kind=8), external :: I1Z,I2Z,I3Z,I4Z,I5Z,I6Z,I7Z,I8Z,&
        &LogTermThetaZ

    ILogZ = 0.0;
    if (abs (myzeta*px + py * myrho ) < logsing .and. abs (myzeta*px +&
        &py * myrho ) > 0.0) then
        call trapz1(LogTermThetaZ,real(0.0,kind=8),real(2.0*pi,kind=8),&
            &logtrapzbig, ILogZ)
    elseif (abs (myzeta*px + py * myrho ) > logsing )then
        call trapz1(LogTermThetaZ, real(0.0,kind=8),real(2.0*pi,kind=8)&
            &,logtrapz, ILogZ)
    endif
    w2IntegrandZ = I1Z(ellipticK,ellipticE)+I2Z(ellipticK1,ellipticE1)&
        +I3Z(ellipticK1,ellipticE1)+I4Z(ellipticK1,ellipticE1)&
        +I5Z(ellipticK1,ellipticE1)+I6Z(ellipticK1,ellipticE1) &
        +I7Z(ellipticK1,ellipticE1)+I8Z(ellipticK1,ellipticE1)+ILogZ
    w2IntegrandZ = w2IntegrandZ*myrho/pi
end function w2IntegrandZ
```

```
function I1R(ellipticK,ellipticE)
    use globalinfo
    implicit none
    real (kind=8) I1R, ellipticE, ellipticK

    I1R=0
    if ( ((py-myzeta)**2+(px-myrho)**2)>singthres) then
    I1R= 2.0*px**(-1.0)*(px**2.0+(-2.0)*px*myrho+myrho**2.0+(py+(-1.0)*&
        &myzeta)**2.0)**(-1.0/2.0)*(px**2.0+2.0*px*myrho+myrho**2.0+&
        &(py+(-1.0)*myzeta)**2.0)**(-1.0)*(py+(-1.0)*myzeta)*((px**2.0+&
        &(-1.0)*myrho**2.0+(-1.0)*(py+(-1.0)*myzeta)**2.0)*ellipticE+&
        &(px**2.0+2.0*px*myrho+myrho**2.0+(py+(-1.0)*myzeta)**2.0)*&
        &ellipticK)
    endif
end function I1R

function I1Z(ellipticK,ellipticE)
    use globalinfo
    implicit none
    real (kind=8) I1Z, ellipticE, ellipticK

    I1Z=0
    if ( ((py-myzeta)**2+(px-myrho)**2)>singthres) then
        I1Z=4.0*(py+(-1.0)*myzeta)**2.0*(px**2.0+(-2.0)*px*myrho+myrho&
            &**2.0+((-1.0)*py+myzeta)**2.0)**(-1.0/2.0)*(px**2.0+2.0*px&
            &*myrho+myrho**2.0+((-1.0)*py+myzeta)**2.0)**(-1.0)*&
            &ellipticE+4.0*(px**2.0+(-2.0)*px*myrho+myrho**2.0+((-1.0)*&
            &py+myzeta)**2.0)**(-1.0/2.0)*ellipticK;
    endif
end function I1Z
```

```
function I2R (ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) I2R, ellipticE1, ellipticK1
    real (kind=8) ::  a

    a=R;
    I2R = (-2*(a**2/(myrho**2 + myzeta**2))**2.5*(a**2*myzeta - (myrho&
        &**2 + myzeta**2)*py)*((a**4 - 2*a**2*myzeta*py - (myrho**2 + &
        &myzeta**2)*(px**2 - py**2))*ellipticE1 - (a**4 + 2*a**2*(myrho&
        &*px - myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2))*&
        &ellipticK1))/(a**2*px*(a**4 + 2*a**2*(myrho*px - myzeta*py) + &
        &(myrho**2 + myzeta**2)*(px**2 + py**2))*Sqrt((a**4 - 2*a**2*&
        &(myrho*px + myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2&
        &))/(myrho**2 + myzeta**2)))
end function I2R

function I2Z(ellipticK1,ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) I2Z, ellipticE1, ellipticK1
    real (kind=8) ::  a

    a=R;
    I2Z= 4.0*(a**2.0*(myrho**2.0+myzeta**2.0)**(-1.0))**(1.0/2.0)*&
        &((myrho**2.0+myzeta**2.0)**(-1.0)*(a**4.0+(-2.0)*a**2.0*(px*&
        &myrho+py*myzeta)+(px**2.0+py**2.0)*(myrho**2.0+myzeta**2.0) &
        &))**(-1.0/2.0)*((-1.0)*a**2.0*(myrho**2.0+myzeta**2.0)**(-2.0)&
        &*(myrho**2.0*py+myzeta*((-1.0)*a**2.0+py*myzeta))**2.0*(a**4.0&
```

```
        &+2.0*a**2.0*(px*myrho+(-1.0)*py*myzeta)+(px**2.0+py**2.0)*&
        &(myrho**2.0+myzeta**2.0))**(-1.0)*ellipticE1+(-1.0)*ellipticK1)
end function I2Z

function I3R (ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) :: I3R, ellipticE1, ellipticK1,coeff
    real (kind=8) ::  a

    a=R;
    coeff=(a**2+(-1.0)*myrho**2+(-1.0)*myzeta**2)*(myrho**2+myzeta**2)&
        &**(-1.0/2.0);
    I3R = coeff*(  (-2*myzeta*((a**4 - 2*a**2*(myrho*px + myzeta*&
        &py) + (myrho**2 + myzeta**2)*(px**2 + py**2))*ellipticE1 - &
        &(a**4 - 2*a**2*myzeta*py + (myrho**2 + myzeta**2)*(px**2 + py&
        &**2))*ellipticK1   ))/(a*(myrho**2 + myzeta**2)**2*px*&
        &Sqrt((a**4 - 2*a**2*(myrho*px + myzeta*py) + (myrho**2 + &
        &myzeta**2)*(px**2 + py**2))/(myrho**2 + myzeta**2))))
    return
end function I3R

function I3Z (ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) :: I3Z, ellipticE1, ellipticK1,coeff
    real (kind=8) ::  a

    a=R;
    coeff=(a**2+(-1.0)*myrho**2+(-1.0)*myzeta**2)*(myrho**2+myzeta**2)&
```

```
        &**(-1.0/2.0);
    I3Z=coeff*(4.0*a*myzeta**2.0*(myrho**2.0+myzeta**2.0)**(-2.0)*((&
        &myrho**2.0+myzeta**2.0)**(-1.0)*(a**4.0+(-2.0)*a**2.0*(px*&
        &myrho+py*myzeta)+(px**2.0+py**2.0)*(myrho**2.0+myzeta**2.0))) &
        &**(-1.0/2.0)*ellipticK1)
    return
end function I3Z

function I4R(ellipticK1, ellipticE1)
use globalinfo
    implicit none
    real (kind=8) :: I4R, ellipticE1, ellipticK1,coeff,a

    a=R;
    coeff=(a**2+(-1.0)*myrho**2+(-1.0)*myzeta**2)*(myrho**2+myzeta**2)&
        &**(-1.0/2.0);
    I4R=-coeff* ((2*a*(-((a**2*myzeta)/(myrho**2 + myzeta**2)) + py)*&
        &((a**4 - 2*a**2*myzeta*py + (myrho**2 + myzeta**2)*(px**2 + &
        &py**2))*(a**4 - 2*a**2*(myrho*px + myzeta*py) + (myrho**2 + &
        &myzeta**2)*(px**2 + py**2))*ellipticE1 - (a**8 - 4*a**6*myzeta&
        &*py - 4*a**2*myzeta*(myrho**2 + myzeta**2)*py*(px**2 + py**2)&
        & + (myrho**2 + myzeta**2)**2*(px**2 + py**2)**2 + a**4*(-2*&
        &myrho**2*(px**2 - py**2) + 2*myzeta**2*(px**2 + 3*py**2)))*&
        &ellipticK1))/((myrho**2 + myzeta**2)**2*px*(a**4 + 2*a**2*&
        &(myrho*px - myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2&
        &))*((a**4 - 2*a**2*(myrho*px + myzeta*py) + (myrho**2 + myzeta&
        &**2)*(px**2 + py**2))/(myrho**2 + myzeta**2))**1.5))
    return
end function I4R
```

```
function I4Z(ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) :: I4Z, ellipticE1, ellipticK1,coeff
    real (kind=8) ::  a

    a=R;
    coeff=(a**2+(-1.0)*myrho**2+(-1.0)*myzeta**2)*(myrho**2+myzeta**2)&
        &**(-1.0/2.0);
    I4Z=-coeff* ((4*a**3*myzeta*(-(a**2*myzeta) + (myrho**2 + myzeta**2&
        &)*py)*ellipticE1)/((myrho**2 + myzeta**2)**2*(a**4 + 2*a**2*&
        &(myrho*px - myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2&
        &))*Sqrt((a**4 - 2*a**2*(myrho*px + myzeta*py) + (myrho**2 +&
        &myzeta**2)*(px**2 + py**2))/(myrho**2 + myzeta**2))))
    return
end function I4Z

function I5R(ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) ::  ellipticE1, ellipticK1,coeff,I5R
    real (kind=8) ::  a

    a=R;
    coeff=(a**2+(-1.0)*myrho**2+(-1.0)*myzeta**2)*(myrho**2+myzeta**2)&
        &**(-1.0/2.0);
    I5R= -coeff*((-2*a**3*myzeta*((a**4 - 2*a**2*myzeta*py - (myrho**2&
        &+ myzeta**2)*(px**2 - py**2))*ellipticE1 - (a**4 + 2*a**2*&
        &(myrho*px - myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2&
        &))*ellipticK1))/((myrho**2 + myzeta**2)**2*px*(a**4 + 2*a**2*&
```

```
        &(myrho*px - myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2&
        &))*sqrt((a**4 - 2*a**2*(myrho*px + myzeta*py) + (myrho**2 + &
        &myzeta**2)*(px**2 + py**2))/(myrho**2 + myzeta**2))))
    return
end function I5R

function I5Z (ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) :: I5Z, ellipticE1, ellipticK1,coeff
    real (kind=8) ::  a

    a=R;
    coeff=(a**2+(-1.0)*myrho**2+(-1.0)*myzeta**2)*(myrho**2+myzeta**2)&
        &**(-1.0/2.0);
    I5Z= -coeff*( (4*a**3*myzeta*(-(a**2*myzeta) + (myrho**2 + myzeta**&
        &2)*py)*ellipticE1 )/((myrho**2 + myzeta**2)**2*(a**4 + 2*a**2*&
        &(myrho*px - myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2&
        &))*Sqrt((a**4 - 2*a**2*(myrho*px + myzeta*py) + (myrho**2 + &
        &myzeta**2)*(px**2 + py**2))/(myrho**2 + myzeta**2))) )
end function I5Z

function I6R (ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) :: I6R, ellipticE1, ellipticK1,coeff
    real (kind=8) ::  a

    a=R;
    coeff=(a**2+(-1.0)*myrho**2+(-1.0)*myzeta**2)*(myrho**2+myzeta**2)&
```

```
        &**(-1.0/2.0);
    I6R= coeff* ((4*myzeta*((a**4 - 2*a**2*(myrho*px + myzeta*py) + &
        &(myrho**2 + myzeta**2)*(px**2 + py**2))*((a**8 - 4*a**6*myzeta&
        &*py - 4*a**2*myzeta*(myrho**2 + myzeta**2)*py*(px**2 + py**2) &
        &+(myrho**2 + myzeta**2)**2*(px**2 + py**2)**2 + 2*a**4*(myrho&
        &**2*py**2 + myzeta**2*(px**2 + 3*py**2)))*ellipticE1- (a**4 - &
        &2*a**2*myzeta*py + (myrho**2 + myzeta**2)*(px**2 + py**2))*&
        &(a**4 + 2*a**2*(myrho*px - myzeta*py) + (myrho**2 + myzeta**2)&
        &*(px**2 + py**2))*ellipticK1 ) + a**2*(-a**2 + myzeta*py)*((a&
        &**4 - 2*a**2*myzeta*py + (myrho**2 + myzeta**2)*(px**2 + py**2&
        &))*(a**4 - 2*a**2*(myrho*px + myzeta*py) + (myrho**2 + myzeta&
        &**2)*(px**2 + py**2))*ellipticE1- (a**8 - 4*a**6*myzeta*py - &
        &4*a**2*myzeta*(myrho**2 + myzeta**2)*py*(px**2 + py**2) + &
        &(myrho**2 + myzeta**2)**2*(px**2 + py**2)**2 + a**4*(-2*myrho&
        &**2*(px**2 - py**2) + 2*myzeta**2*(px**2 + 3*py**2)))*&
        &ellipticK1 )))/(a*(myrho**2 + myzeta**2)**3*px*(a**4 + 2*a**2&
        &*(myrho*px - myzeta*py) + (myrho**2+ myzeta**2)*(px**2 + py**2&
        &))*((a**4 - 2*a**2*(myrho*px + myzeta*py) + (myrho**2 + myzeta&
        &**2)*(px**2 +py**2))/(myrho**2 + myzeta**2))**1.5))
end function I6R

function I6Z (ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) :: I6Z, ellipticE1, ellipticK1,coeff
    real (kind=8) ::  a

    a=R;
    coeff=(a**2+(-1.0)*myrho**2+(-1.0)*myzeta**2)*(myrho**2+myzeta**2)&
        &**(-1.0/2.0);
```

```
    I6Z= coeff* (   (-4*a*myzeta**2*((a**4 - (myrho**2 + myzeta**2)*&
        &(px**2 + py**2))*ellipticE1 + (a**4 + 2*a**2*(myrho*px - &
        &myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2))*&
        &ellipticK1))/((myrho**2 + myzeta**2)**2*(a**4 + 2*a**2*(myrho*&
        &px - myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2))*&
        &Sqrt((a**4 - 2*a**2*(myrho*px + myzeta*py) + (myrho**2 + &
        &myzeta**2)*(px**2 + py**2))/(myrho**2 + myzeta**2))))
end function I6Z

function I7R (ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) :: I7R, ellipticE1, ellipticK1,coeff
    real (kind=8) ::  a

    a=R;
    coeff=(-1.0/2.0)*(a**2.0+(-1.0)*px**2.0+(-1.0)*py**2.0)*(myrho**2.0&
        &+myzeta**2.0)**(-3.0/2.0)*((-1.0)*a**2.0+myrho**2.0+myzeta**&
        &2.0)
    I7R=- coeff* (  (2*((-3*myzeta*(a**4 + 2*a**2*(myrho*px - myzeta*&
        &py) + (myrho**2 + myzeta**2)*(px**2 + py**2))*(a**4 - 2*a**2*&
        &(myrho*px + myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2&
        &))*(-((a**4 - 2*a**2*myzeta*py - (myrho**2 + myzeta**2)*(px**2&
        & - py**2))*ellipticE1 ) + (a**4 + 2*a**2*(myrho*px - myzeta*py&
        &) + (myrho**2 + myzeta**2)*(px**2 + py**2))*ellipticK1 ))/&
        &(myrho**2 + myzeta**2) - a**2*(-((a**2*myzeta)/(myrho**2 + &
        &myzeta**2)) + py)*(-((a**8 - 4*a**6*myzeta*py - 4*a**2*myzeta*&
        &(myrho**2 + myzeta**2)*py*(px**2 + py**2) + (myrho**2 + myzeta&
        &**2)**2*(px**2 + py**2)**2 + 2*a**4*(myrho**2*(7*px**2 + py**2&
        &) + myzeta**2*(px**2 + 3*py**2)))*ellipticE1 ) + (a**4 - 2*a**&
```

```
        &2*myzeta*py + (myrho**2 + myzeta**2)*(px**2 + py**2))*(a**4 + &
        &2*a**2*(myrho*px - myzeta*py) + (myrho**2 + myzeta**2)*(px**2 &
        &+ py**2))*ellipticK1 + 2*(myrho**2 + myzeta**2)*px**2*(4*(a**4&
        & -    2*a**2*myzeta*py + (myrho**2 + myzeta**2)*(px**2 + py**2))*&
        &ellipticE1 - (a**4 + 2*a**2*(myrho*px - myzeta*py) + (myrho**2&
        & + myzeta**2)*(px**2 + py**2))*ellipticK1 ))))/(a*px*(a**4 + &
        &2*a**2*(myrho*px - myzeta*py) + (myrho**2 + myzeta**2)*(px**2&
        & + py**2))**2*((a**4 - 2*a**2*(myrho*px + myzeta*py) + (myrho&
        &**2 + myzeta**2)*(px**2+py**2))/(myrho**2 + myzeta**2))**1.5))
end function I7R

function I7Z (ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) :: I7Z, ellipticE1, ellipticK1,coeff
    real (kind=8) ::  a

    a=R;
    coeff=(-1.0/2.0)*(a**2.0+(-1.0)*px**2.0+(-1.0)*py**2.0)*(myrho**2.0&
        &+myzeta**2.0)**(-3./2.)*((-1.0)*a**2.0+myrho**2.0+myzeta**2.0)
    I7Z = -coeff* ((4*(myrho**2 + myzeta**2)**2*(((a**2*(myrho**2 + 4*&
        &myzeta**2) - 3*myzeta*(myrho**2 + myzeta**2)*py)*(a**4 + &
        &2*a**2*(myrho*px - myzeta*py) + (myrho**2 + myzeta**2)*(px**2&
        & + py**2))*(a**4 - 2*a**2*(myrho*px + myzeta*py) + (myrho**2 +&
        & myzeta**2)*(px**2 + py**2))*ellipticE1 )/(a*(myrho**2 + &
        &myzeta**2)**3) - (a*(-((a**2*myzeta)/(myrho**2 + myzeta**2)) +&
        & py)**2*(4*(a**4 - 2*a**2*myzeta*py + (myrho**2 + myzeta**2)*&
        &(px**2 + py**2))*ellipticE1-(a**4 + 2*a**2*(myrho*px - myzeta*&
        &py) + (myrho**2 + myzeta**2)*(px**2 + py**2))*ellipticK1 ))/&
        &(myrho**2 + myzeta**2)))/((a**4 + 2*a**2*(myrho*px-myzeta*py) &
```

```
        &+(myrho**2 + myzeta**2)*(px**2 + py**2))**2*((a**4 - 2*a**2*&
        &(myrho*px + myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**&
        &2))/(myrho**2 + myzeta**2))**1.5))
end function I7Z

function I8R (ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) :: I8R, ellipticE1, ellipticK1,coeff
    real (kind=8) ::  a

    a=R;
    coeff=(-1.0/2.0)*(a**2.0+(-1.0)*px**2.0+(-1.0)*py**2.0)*(myrho**&
        &2.0+myzeta**2.0)**(-3.0/2.0)*((-1.0)*a**2.0+myrho**2.0+myzeta&
        &**2.0)
    I8R = -coeff*( (4*myzeta*Sqrt(myrho**2 + myzeta**2)*((a**12 - 5*a**&
        &10*myzeta*py - a**8*(myrho**2 + 5*myzeta**2)*(px**2 - 2*py**2)&
        & + (myrho**2 + myzeta**2)**3*px**2*(px**2 + py**2)**2 + a**2*&
        &myzeta*(myrho**2 + myzeta**2)**2*py*(3*px**4 + 2*px**2*py**2 -&
        & py**4) + 2*a**6*myzeta*py*(myzeta**2*(7*px**2 - 5*py**2) + &
        &myrho**2*(9*px**2 - 3*py**2)) - a**4*(myzeta**4*(5*px**4 +&
        & 12*px**2*py**2 - 5*py**4) + 6*myrho**2*myzeta**2*(px**4 + 4*&
        &px**2*py**2 - py**4) + myrho**4*(px**4 + 12*px**2*py**2 - py**&
        &4)))*ellipticE1 - (a**12 + a**10*(2*myrho*px - 5*myzeta*py) + &
        &(myrho**2 + myzeta**2)**3*px**2*(px**2 + py**2)**2 + a**2*&
        &(myrho**2 + myzeta**2)**2*(px**2 + py**2)*(2*myrho*px**3 - &
        &myzeta*py*(3*px**2 + py**2)) - a**4*(myrho**2 + myzeta**2)*&
        &(px**2 + py**2)*(2*myrho*myzeta*px*py + myrho**2*(px**2 - &
        &py**2)    - myzeta**2*(px**2 + 5*py**2)) + a**8*(-6*myrho*myzeta*&
        &px*py - myrho**2*(px**2 - 2*py**2) + myzeta**2*(px**2 + 10*py&
```

```
        &**2)) - 2*a**6*(-3*myrho*myzeta**2*px*py**2 + 3*myrho**2*&
        &myzeta*py**3 + myzeta**3*py*(2*px**2 + 5*py**2) + myrho**3*(2*&
        &px**3 - px*py**2)))*ellipticK1 ))/(a*px*(a**4 + 2*a**2*(myrho*&
        &px - myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2))**2*&
        &(a**4 - 2*a**2*(myrho*px + myzeta*py) + (myrho**2 + myzeta**2)&
        &*(px**2 + py**2))**1.5))
end function I8R

function I8Z (ellipticK1, ellipticE1)
    use globalinfo
    implicit none
    real (kind=8) :: I8Z, ellipticE1, ellipticK1,coeff
    real (kind=8) ::  a

    a=R;
    coeff = (-1.0/2.0)*(a**2.0+(-1.0)*px**2.0+(-1.0)*py**2.0)*(myrho**&
        &2.0+myzeta**2.0)**(-3.0/2.0)*((-1.0)*a**2.0+myrho**2.0+myzeta&
        &**2.0)
    I8Z = -coeff* ( (4*a*myzeta*((-2*myzeta*(a**4 + 2*a**2*(myrho*px - &
        &myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2))*(a**4 - &
        &2*a**2*(myrho*px + myzeta*py) + (myrho**2 + myzeta**2)*(px**2 &
        &+ py**2))*ellipticE1 )/(myrho**2 + myzeta**2) + (-((a**2*&
        &myzeta)/(myrho**2 + myzeta**2)) + py)*(2*(-a**2 + myzeta*py)*&
        &(4*(a**4 - 2*a**2*myzeta*py + (myrho**2 + myzeta**2)*(px**2 + &
        &py**2))*ellipticE1 - (a**4 + 2*a**2*(myrho*px - myzeta*py) + &
        &(myrho**2 + myzeta**2)*(px**2 + py**2))*ellipticK1 ) + ((a**8&
        & - 4*a**6*myzeta*py - 4*a**2*myzeta*(myrho**2 + myzeta**2)*py*&
        &(px**2 + py**2) + (myrho**2 + myzeta**2)**2*(px**2 + py**2)**2&
        & + 2*a**4*(myrho**2*(7*px**2 + py**2) + myzeta**2*(px**2 +&
        & 3*py**2)))*ellipticE1 - (a**4 - 2*a**2*myzeta*py + (myrho**2 &
```

```
        &+ myzeta**2)*(px**2 + py**2))*(a**4 + 2*a**2*(myrho*px - &
        &myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2))*&
        &ellipticK1 )/a**2)))/((a**4 + 2*a**2*(myrho*px - myzeta*py) + &
        &(myrho**2 + myzeta**2)*(px**2 + py**2))**2*((a**4 - 2*a**2*&
        &(myrho*px + myzeta*py) + (myrho**2 + myzeta**2)*(px**2 + py**2&
        &))/(myrho**2 + myzeta**2))**1.5))
end function I8Z


function LogTermThetaR(theta)
    !---------------------------------------
    !  Integrand coming from log term to integrate 3D
    !---------------------------------------
    use globalinfo
    implicit none
    real (kind=8) ::  phi,x2,coeffn,LogTermThetaR
    real (kind=8) :: theta,pxx,y1,y2,y3,x1,x3,pxy,ys1,ys2,ys3,pxs,pxxys
    real (kind=8) ::  a


    a=R;
    y1 = myrho*cos(theta)
    y2 = myrho*sin(theta)
    y3 = myzeta
    x1 = px
    x2= 0
    x3 = py
    pxy=(myrho**2.d0+myzeta**2.d0)**(1.d0/2.d0)
    pxx=(px**2.d0+py**2.d0)**(1.d0/2.d0)
    ys1 = a**2/pxy**2.d0*y1
    ys2= a**2/pxy**2.d0*y2
    ys3 = a**2/pxy**2.d0*y3
```

```
    coeffn = (-3.0*(a**2 - x1**2 - x2**2 - x3**2)*(a**2 - y1**2 - y2**2&
        &- y3**2))/(2.0*a*(y1**2 + y2**2 + y3**2))
    pxs=(ys1**2.d0+ys2**2.d0+ys3**2.d0)**(1.d0/2.d0)
    pxxys=((x1-ys1)**2 + (x2-ys2)**2 +(x3-ys3)**2)**(1.d0/2.d0)
    LogTermThetaR= coeffn *((((pxs*x1 + pxx*ys1)*(pxs*x3 + pxx*ys3))/ &
        &(pxx*(pxs*pxx + x1*ys1 + x2*ys2 + x3*ys3)**2) - (x1*ys3)/(pxx*&
        &(pxs*pxx + x1*ys1 + x2*ys2 + x3*ys3)) + ((pxs - pxxys)*(pxs*&
        &(x1 - ys1) + pxxys*ys1)*(pxs*(x3 - ys3) + pxxys*ys3))/(pxxys**&
        &2*(-pxs**2 + pxs*pxxys + x1*ys1 + x2*ys2 + x3*ys3)**2) + &
        &((x1 - ys1)*(pxs**2*(x3 - ys3) + pxxys**2*ys3))/(pxxys**3*(&
        &-pxs**2 + pxs*pxxys + x1*ys1 + x2*ys2 + x3*ys3)))/pxs)
    return
end function LogTermThetaR


function LogTermThetaZ(theta)
!----------------------------------------
!  Integrand coming from log term to integrate 3D
!----------------------------------------
    use globalinfo
    implicit none

    real (kind=8) :: phi,x2,coeffn,LogTermThetaZ
    real (kind=8) :: theta,pxx,y1,y2,y3,x1,x3,pxy,ys1,ys2,ys3,pxs,pxxys
    real (kind=8) ::  a

    a=R;
    phi=0
    y1 = myrho*cos(theta)
    y2 = myrho*sin(theta)
```

```
    y3 = myzeta
    x1 = px
    x2= 0
    x3 = py
    pxy=(myrho**2.d0+myzeta**2.d0)**(1.d0/2.d0)
    pxx=(px**2.d0+py**2.d0)**(1.d0/2.d0)
    ys1 = a**2/pxy**2.d0*y1
    ys2= a**2/pxy**2.d0*y2
    ys3 = a**2/pxy**2.d0*y3
    coeffn = (-3.0*(a**2 - x1**2 - x2**2 - x3**2)*(a**2 - y1**2 - y2**2&
        &- y3**2))/(2.0*a*(y1**2 + y2**2 + y3**2))
    pxs=(ys1**2+ys2**2+ys3**2)**(1.d0/2.d0)
    pxxys=((x1-ys1)**2 + (x2-ys2)**2 +(x3-ys3)**2)**(1.d0/2.d0)
    LogTermThetaZ=coeffn*(((pxs*x3 + pxx*ys3)**2/(pxx*(pxs*pxx + x1*ys1&
        &+ x2*ys2 + x3*ys3)**2) - (pxs*pxx + x3*ys3)/(pxx*(pxs*pxx +&
        & x1*ys1 + x2*ys2 + x3*ys3)) + ((pxs - pxxys)*(pxs*(x3 - ys3) +&
        & pxxys*ys3)**2)/ (pxxys**2*(-pxs**2 + pxs*pxxys + x1*ys1 + x2*&
        &ys2 + x3*ys3)**2) + (pxs*pxxys**2*(-pxs + pxxys) + pxs**2*(x3 &
        &- ys3)**2 + pxxys**2*(x3 - ys3)*ys3)/(pxxys**3*(-pxs**2 + pxs*&
        &pxxys + x1*ys1 + x2*ys2 + x3*ys3)))/pxs )
    return
end
!---------------------------------------------------------------------
!> @author
!> C. Falcon, UNC-CH
!
!DESCRIPTION:
! Determines a number of special positions on the interface that define
!! the intersections of the current interface with the unperturbed
!! interface. See Claudia's notebook for details.
```

```
!
! @param myx the x positions of the interface
! @param myy the y positions of the interface
!
! Does not return anything, but sets the following variables that
! define the aforementioned intersection cases and points that are
! contained in globalinfo.

! Flags that define which cases are active. All integers with 0=inactive,
! 1=active:
! - flagb
! - flagu
! - flagl
! - flagt
! Flags that define where the specific points are in the case that they
! are active:
! - xflagb : x value of point of intersection of perturbed interface
! with original interface
! - xflagl: The point on the interface w/ same y val as bottom of sphere
! - yend
subroutine setSpecialPositions(myx, myy, XN, R, flagb, flagu, flagl, &
    &flagt, xflagb, xflagl, yend)
    !determine special positions on the interface
    implicit none

    ! In
    integer (kind=4), intent(in)     :: XN
    real (kind=8)                     :: myx(XN+1), myy(XN+1)
    real (kind=8), intent(in)          :: R
```

```
    ! Out
    integer (kind=4), intent(out)    :: flagb, flagu, flagl, flagt
    real (kind=8), intent(out)         :: xflagb, xflagl, yend

    ! Local
    integer (kind=4)                   :: temp(1), tempmaxi

    flagl = 0
    flagu = 0

!----find position of backflow; that is, bit that's higher than starting
    yend = myy(XN+1)

    if(maxval(myy) > yend) then
        !flagb should be the index of the last point below the original interface
        flagb = XN+1
        do tempmaxi = 1, XN
            if (myy(tempmaxi+1) > yend .and. myy(tempmaxi) <= yend) then
                flagb = tempmaxi
                exit
            endif
        end do
        !tempmaxi will now hold the starting index of the window of length
        !5 around the intersection of the new interface with the starting position
        tempmaxi = min(flagb-2, XN-3)
        tempmaxi = max(tempmaxi, 1)
        !print *, 'sp 1'
        !finds the x value where the interface crosses the original interface
        !within the above window
        call interpbridge(5, myy(tempmaxi:tempmaxi+4), myx(tempmaxi: &
```

```
            &tempmaxi+4),  yend, xflagb)
         !print *, 'sp 2'
    else
       flagb = XN+1;
       xflagb = myx(XN+1);
    endif

!-----find x position of bottom of sphere-------------------------------

    if (yend >= -R) then
        temp = minloc(abs(myy+R))
        flagl = temp(1)
        if (myy(1) >= -R) then
            xflagl = 0
        else
            !print *, 'sp 3'
             print*, min(flagb+2,XN+1), "N test1"
            call interpbridge(min(flagb+2, XN+1), myy(1:min(flagb+2, &
                &XN+1)), myx(1:min(flagb+2, XN+1)), -R, xflagl)
            !print *, 'sp 4'
        endif
    endif

!------find x position of top of sphere---------------------------------

    if (yend >= R) then

            !flagu=is 1 if interface is past sphere top
        flagu = 1;
```

```
        temp = minloc(abs(myy-R))
        flagt = temp(1)
    endif
end subroutine setSpecialPositions
```

Stokes Velocity

```
!---------------------------------------------------------------------
!> @author
!> H. Arrowood, UNC-CH; adapted from work of C. Falcon, UNC-CH
!
! DESCRIPTION:
!> Initializes spacial comp of stokes flow in a cylinder, used for
!> interpolation later rather than recomputing every timestep
!> @brief
!> I reworked Claudia's code, replacing her reflections with the drop
!> reflection, and the third reflection with my crude first approx
!> to the third reflection
!---------------------------------------------------------------------
subroutine cylindervelinit()
    use globalinfo
    implicit none
    integer index

!------Set up discretization pts---------------------------------------
    do index = 1, NZ
        WZ(index) = real(index-1.0, kind=8)*(LZ-epsilon)/NZ
        myk(index) = real(index-1.0, kind =8)*2.0*pi/(LZ-epsilon)
    end do

    do index = 1, NR
```

```
        WR(index) = real(index-1.0, kind=8)*LR/NR
    end do
!------Now call H and G functions to compute reflections----------------

    call Hfunc(NZ, WZ+epsilon, myHZ)
    call Gfunc(NZ, WZ+epsilon, myGZ)
    call Hfunc(NR, WR, myHR)
    call Gfunc(NR, WR, myGR)

    firstpartZ = (WZ+epsilon)/2.0*(myHZ+myGZ);
    firstpartR = WR/2.0*(myHR+myGR);

    !print *, "firstpartZ", firstpartZ(10)
    !print *, "firstpartR", firstpartR(10)

    do index = 1, upperRangeZ
        zcoordinateZ(index) = (index-1.0)*2.0*pi/(LZ-epsilon)
    end do

    do index = 1, upperRangeR
        zcoordinateR(index) = (index-1.0)*2.0*pi/LR
    end do

    call cylindervelgrid()

!    print *, cylindervelZ
!    stop
end subroutine cylindervelinit

!---------------------------------------------------------------------
```

```
!> @author
!> C. Falcon, UNC-CH; later edited by H. Arrowood, UNC-CH
!
! DESCRIPTION:
!> Used in cylindervelinit, which initializes the spacial component of
!> Stokes flow in a cylinder
!> @brief
!>
! REVISION HISTORY:
! 16_Jan_2018 - Added documentation - H. Arrowood
!
!------------------------------------------------------------------------
subroutine cylindervelgrid()
use globalinfo
    implicit none

    real (kind=8) :: FZ(NZ), FR(NR), BESSI
    !real (kind=8), external :: sign
    integer myi, WRi, WZi
      real    ( kind = 4 ) wsavez(4*NZ+15), wsaver(4*NR+15)
      complex ( kind = 4 ) tempFR(NR), tempFZ(NZ)
      real (kind = 8) :: besseli0R(NR), besseli1R(NR), besseli0Z(NZ), besseli1Z(NZ)



    do myi = 1, XN+1
        !r-component of velocity
        do WRi = 1, NR
            besseli0R(WRi) = BESSI(0,WR(WRi)*x(myi))
            besseli1R(WRi) = BESSI(1,WR(WRi)*x(myi))
        end do
```

```
        tempFR = real((x(myi)*firstpartR*besseli0R-myGR*besseli1R)*0.5,&
            & kind=4)
        tempFR(1) = 0.0
!        print *, firstpartR
!
!        print *, myGR
!
!        print *, tempFR(16384)
!        stop
        call cffti ( NR, wsaver )
          call cfftb ( NR, tempFR, wsaver )
          FR = real(aimag(tempFR)/NR*LR/pi, kind=8)
          cylindervelR(1:upperRangeR, myi) = FR(1:upperRangeR)

        !z-component of velocity
        do WZi = 1, NZ
            besseli0Z(WZi) = BESSI(0,(WZ(WZi)+epsilon)*x(myi))
            besseli1Z(WZi) = BESSI(1,(WZ(WZi)+epsilon)*x(myi))
        end do

        tempFZ = real((x(myi)*firstpartZ*besseli1Z+myHZ*besseli0Z)*0.5,&
            & kind=4)
        call cffti ( NZ, wsavez )
        call cfftb ( NZ, tempFZ, wsavez )
        FZ = real(exp(imagi*epsilon*abs(myk))*real(tempFZ/NZ, kind=8)*&
            &(LZ-epsilon)/pi+3.0*R/pi*epsilon*((-2.0*R**2/(3.0*R0**2)+&
            &1.0)*x(myi)**2/R0**2+log(epsilon*0.5*R0)-1.0), kind=8)
        cylindervelZ(1:upperRangeZ, myi) = FZ(1:upperRangeZ)
```

```
    end do

end subroutine cylindervelgrid




!---------------------------------------------------------------------
!> @author
!> H. Arrowood, UNC-CH
!
! DESCRIPTION:
!> Uses stress coeffs and calculation based on the work of Haberman
!> to compute Stokes velocity
!
! REVISION HISTORY:
! 07_Nov_2017 - Added documentation - H. Arrowood
!
!> @param[in] G2,G4, G6, G8 pert stress coeffs
!> @param[out] su, sv components of stokes velocity on interface pts
!
!---------------------------------------------------------------------
subroutine stokes () !(su, sv)
    use globalinfo
    implicit none

    real (kind=8) :: k1(XN+1), k2(XN+1),u3r(XN+1),u3z(XN+1),myr(XN+1), &
        mytheta(XN+1), tempx(4), tempy(4), myinterp1, myinterp2, &
        myinterp3, myinterp4
    real(kind=8)    :: B2, B4, B6, B8, D2, D4, D6, D8!, u3constapprox
```

```
    real(kind=8)    :: stokesvelocityr(XN+1), stokesvelocitytheta(XN+1)&
        &, stokesvelocity3r(XN+1), stokesvelocity3theta(XN+1), stokeste&
        &stx(XN+1), stokestesty(XN+1)
     real(kind=8), external   :: LegendreP, GegenbauerC
    real (kind=8), external :: sign
    integer (kind=4) :: starti, sizecyl, i, j, k, ii

    k1 = 0
    k2 = 0
    sizecyl = size(cylindervelR, 2)
    do i=1, XN+1
        myr(i)  = dsqrt(sx(i)**2+sy(i)**2)
        mytheta(i) = dacos(sy(i)/myr(i))
       ! print*, "arctan", atan(sx(i)/sy(i))
        !print*, "adcos", adcos(sy(i)/myr(i))

        !for non-uniform interface
        if (sx(i)<=R0cl) then
        starti = floor((sx(i)*(XNclose**2.0)/R0cl)**(1.0/2.0)+1.0)
        else
        starti = floor((sx(i)-R0cl)*XNfar/(R0-R0cl)+XNclose +1.0)
        endif

        starti = max(starti, 1);
        starti = min(starti, sizecyl-3);

!---------horizontal velocity component (second reflection)-------------
!for reference,  interpbridge(N, interpx, interpy, xval, yval)----------
        call interpbridge(upperRangeR, zcoordinateR, cylindervelR(1:upp&
        &erRangeR, starti), abs(sy(i)), myinterp1)
```

```
        call interpbridge(upperRangeR, zcoordinateR, cylindervelR(1:upp&
        &erRangeR, starti+1), abs(sy(i)), myinterp2)
        call interpbridge(upperRangeR, zcoordinateR, cylindervelR(1:upp&
        &erRangeR, starti+2), abs(sy(i)), myinterp3)
        call interpbridge(upperRangeR, zcoordinateR, cylindervelR(1:upp&
        &erRangeR, starti+3), abs(sy(i)), myinterp4)


        tempx = (/(startx(starti+j), j=0,3)/)
        tempy = (/ myinterp1, myinterp2, myinterp3, myinterp4 /)
         tempy=tempy*sign(sy(i))
        call interpbridge(4, tempx, tempy, max(min(sx(i), R0), real(0.0&
        &, kind=8)), k1(i))

!----------vertical velocity component (second reflection)--------------
        call interpbridge(upperRangeZ, zcoordinateZ, cylindervelZ(1:upp&
        &erRangeZ, starti), abs(sy(i)), myinterp1)
        call interpbridge(upperRangeZ, zcoordinateZ, cylindervelZ(1:upp&
        &erRangeZ, starti+1), abs(sy(i)), myinterp2)
        call interpbridge(upperRangeZ, zcoordinateZ, cylindervelZ(1:upp&
        &erRangeZ, starti+2), abs(sy(i)), myinterp3)
        call interpbridge(upperRangeZ, zcoordinateZ, cylindervelZ(1:upp&
        &erRangeZ, starti+3), abs(sy(i)), myinterp4)

        tempy = (/ myinterp1, myinterp2, myinterp3, myinterp4 /)
        call interpbridge(4, tempx, tempy, max(min(sx(i), R0), real(0.0&
        &, kind=8)), k2(i))


    end do
```

```
      !r and theta components of stokes velocities (first reflection)

      B2 = -R**3*(3*muin*U+R*mu*G2)/(6*(mu+muin))
      D2 = -(-6*R*mu*U-9*R*muin*U-R**2*mu*G2)/(6*(mu+muin))

      B4 = 0.0!R**6*mu*G4/(14*(mu+muin))
      D4 = 0.0!R**4*mu*G4/(14*(mu+muin))

      B6 = 0.0!R**8*mu*G6/(22*(mu+muin))
      D6 = 0.0!R**6*mu*G6/(22*(mu+muin))

      B8 = 0.0!R**10*mu*G8/(30*(mu+muin))
      D8 = 0.0!R**8*mu*G8/(30*(mu+muin))

v2atorigin = 1./(2.*Pi)*(1./(R0**3*(1.+ (muin/mu)))*R*U*(R0**2*(-8.566&
     &2264298116352.849339644717457*(muin/mu))+6.515546450454206*R**2&
     &*(muin/mu))) !this has been checked! -H.
!     print*, "U in cylindervel after addition", U
      do k=1, XN+1
          stokesvelocityr(k)=-(LegendreP(1,mytheta(k))*(-(U)+B2*1/(myr(k)&
              &**3)+D2*(1/myr(k)))+LegendreP(3,mytheta(k))*(B4*1/(myr(k)&
              &**5)+D4*(1/myr(k)**3))+LegendreP(5,mytheta(k))*(B6*1/&
              &(myr(k)**7)+D6*(1/myr(k)**5))+LegendreP(7,mytheta(k))*&
              &(B8*1/(myr(k)**9)+D8*(1/myr(k)**7)))

          if (k==1) then
              stokesvelocitytheta(k)=0.0
          else
              stokesvelocitytheta(k)=1/dsin(mytheta(k))*(GegenbauerC(2,&
```

```
                &mytheta(k))*(-2*(U)-B2*1/(myr(k)**3)+D2*1/myr(k))+Gegen&
                &bauerC(4,mytheta(k))*(-3*B4*1/(myr(k)**5)-D4*1/(myr(k)&
                &**3))+GegenbauerC(6,mytheta(k))*(-5*B6*1/(myr(k)**7)-&
                &3*D6*1/(myr(k)**5))+GegenbauerC(8,mytheta(k))*(-7*B8*1&
                &/(myr(k)**9)-5*D8*1/(myr(k)**7)))
        endif
    end do

!------Approx Third Reflection/Leading Order Error on Drop Surface------
!------This causes the code to violate boundary conditions on both the
!surface of drop and cylinder walls, but is the approximation I used to
!compute the drag-----------------------------------------------------



do k=1, XN+1
        stokesvelocity3r(k)=-LegendreP(1,mytheta(k))*(-(v2atorigin)+(-R&
        &**3*(3*muin*v2atorigin+R*mu*G2)/(6*(mu+muin)))*1./(myr(k)&
        &**3)+((6*R*mu*v2atorigin+9*R*muin*v2atorigin+R**2*mu*G2)/&
        &(6*(mu+muin)))*(1/myr(k)))

        if (k==1) then
            stokesvelocity3theta(k)=0
        else
            stokesvelocity3theta(k)=1/dsin(mytheta(k))*(GegenbauerC(2,&
            &mytheta(k))*(-2*(v2atorigin)-(-(R**3*( 3.*muin*v2atorigi&
            &n))/(6.*(mu + muin)))*1/(myr(k)**3)+((6*R*mu*v2atorigin+9*&
            &R*muin*v2atorigin)/(6*(mu+muin)))*1/myr(k)))
        endif
    end do
```

```
!------Stokes Velocity Full Assembly--------------------------
    do ii=1,XN+1
        if(ii==1)then !force first point horizontal vel to be 0 to fix
        !floating-point error
            if (myr(ii)>1.2*R)then
                su(ii) = 0.0!-(dsin(mytheta(ii))*stokesvelocityr(ii)+dc&
                &os(mytheta(ii))*stokesvelocitytheta(ii))
                sv(ii) = -(dcos(mytheta(ii))*stokesvelocityr(ii)-dsin(m&
                &ytheta(ii))*stokesvelocitytheta(ii))+ k2(ii)*U
            else
                su(ii) = 0.0!-((dsin(mytheta(ii))*stokesvelocityr(ii)+d&
                &cos(mytheta(ii))*stokesvelocitytheta(ii)))
                sv(ii) = -((dcos(mytheta(ii))*stokesvelocityr(ii)-dsin(&
                &mytheta(ii))*stokesvelocitytheta(ii))+(dcos(mytheta(ii&
                &))*stokesvelocity3r(ii)-dsin(mytheta(ii))*stokesveloci&
                &ty3theta(ii)))
            endif
        else
            if (myr(ii)>1.2*R)then
                su(ii) = -(dsin(mytheta(ii))*stokesvelocityr(ii)+dcos(m&
                &ytheta(ii))*stokesvelocitytheta(ii))+ k1(ii)*U
                sv(ii) = -(dcos(mytheta(ii))*stokesvelocityr(ii)-dsin(m&
                &ytheta(ii))*stokesvelocitytheta(ii))+ k2(ii)*U
            else
                su(ii) = -((dsin(mytheta(ii))*stokesvelocityr(ii)+dcos(&
                &mytheta(ii))*stokesvelocitytheta(ii))+(dsin(mytheta(ii&
                &))*stokesvelocity3r(ii)+dcos(mytheta(ii))*stokesveloci&
                &ty3theta(ii)))
                sv(ii) = -((dcos(mytheta(ii))*stokesvelocityr(ii)-dsin(&
```

```
                &mytheta(ii))*stokesvelocitytheta(ii))+(dcos(mytheta(ii&
                &))*stokesvelocity3r(ii)-dsin(mytheta(ii))*stokesveloci&
                &ty3theta(ii)))
            endif
        endif
    end do
end subroutine stokes



!-----------------------------------------------------------------------
!> @author
!> H. Arrowood, UNC-CH; adapted from work of C. Falcon, UNC-CH
!
! DESCRIPTION:
!> Evaluates the H function of the second reflection for a drop in a
!> cylinder
!> @brief
!> I reworked Claudia's code, adding in the reflections for the drop
!> rather than the sphere case. Note that this is the spacial part only,
!> all V(t)-dependence has been factored out.
! REVISION HISTORY:
!> 12_Dec_2017 - adapted H expression - H. Arrowood
!> 16_Jan_2018 - added documentation - H. Arrowood
!> 23_Jan_2018 - verified H - H. Arrowood
!
! @param Num - the number of points at which to evaluate the function
! @param lambda - the variable of integration in Hfunc
! @return ReturnH - the H-vals at each point
!-----------------------------------------------------------------------
subroutine Hfunc(Num, lambda, ReturnH)
```

```
use globalinfo
    implicit none

    integer index, Num
    real (kind=8) :: lambda(Num), besselk0(Num), besselk1(Num),&
          besseli1(Num), besseli2(Num), besseli0(Num), ReturnH(Num),&
        BESSK, BESSI

    do index = 1, Num
        besselk0(index) = BESSK(0,R0*lambda(index))
        besselk1(index) = BESSK(1,R0*lambda(index))
        besseli0(index) = BESSI(0,R0*lambda(index))
        besseli1(index) = BESSI(1,R0*lambda(index))
        besseli2(index) = BESSI(2,R0*lambda(index))
    end do

    ReturnH = (R*(R0*lambda*besseli0*(-(4. + 6.*(muin/mu) + &
        &R**2*(muin/mu)*lambda**2)*real(besselk0) + R0*(2. + 3.*&
        &(muin/mu))*lambda*real(besselk1)) + besseli1*((8. + 12.*&
        &(muin/mu) + (2.*R**2*(muin/mu) + R0**2*(2. + 3.*(muin/mu)))*&
        &lambda**2)*real(besselk0) - R0*lambda*(4. + 6.*(muin/mu) + &
        &R**2*(muin/mu)*lambda**2)*real(besselk1))))/((1. + (muin/mu))*&
        &(R0*lambda*besseli0**2 - 2.* besseli0*besseli1 - R0*lambda*&
        &besseli1**2))



end subroutine

!-----------------------------------------------------------------------
!> @author
```

```
!> H. Arrowood, UNC-CH; adapted from work of C. Falcon, UNC-CH
!
! DESCRIPTION:
!> Evaluates the G function of the second reflection for a drop in a
!> cylinder
!> @brief
!> I reworked Claudia's code, adding in the reflections for the drop
!> rather than the sphere case. Note that this is the spacial part only,
!> all V(t)-dependence has been factored out.
! REVISION HISTORY:
!> 12_Dec_2017 - adapted G expression - H. Arrowood
!> 16_Jan_2018 - added documentation - H. Arrowood
!> 23_Jan_2018 - final verification - H. Arrowood
!
! @param Num - the number of points at which to evaluate the function
! @param lambda - the variable of integration in Gfunc
! @return ReturnH - the G-vals at each point
!-----------------------------------------------------------------------
subroutine Gfunc(Num, lambda, ReturnG)
    use globalinfo
    implicit none

    integer index, Num
    real (kind=8) :: lambda(Num), besselk0(Num), besselk1(Num), &
        & besselk2(Num), besseli1(Num), besseli2(Num), besseli0(Num), &
        & ReturnG(Num), BESSK, BESSI

    do index = 1, Num
        besselk0(index) = BESSK(0,R0*lambda(index))
        besselk1(index) = BESSK(1,R0*lambda(index))
```

```
        besselk2(index) = BESSK(2,R0*lambda(index))
        besseli0(index) = BESSI(0,R0*lambda(index))
        besseli1(index) = BESSI(1,R0*lambda(index))
        besseli2(index) = BESSI(2,R0*lambda(index))
    end do

     ReturnG = (lambda**2*(-R**3*R0*(muin/mu)*lambda*besseli1*&
        &real(besselk0) + 2.*R*R0**2*besseli1*real(besselk0) + 3.*R*&
        &R0**2*(muin/mu)*besseli1*real(besselk0) + 2.*R*R0**2*besseli0*&
        &real(besselk1) - 2.*R**3*(muin/mu)*besseli0*real(besselk1) + &
        &3.*R*R0**2*(muin/mu)*besseli0*real(besselk1) - R**3*R0*&
        &(muin/mu)*lambda*besseli1*real(besselk1)))/((1. + (muin/mu))*&
        &(-R0*lambda*besseli0**2 + 2.*besseli0*besseli1 + R0*lambda*&
        &besseli1**2))

end

function sign(val)
    implicit none

    real(kind=8) val, sign

    if (val< 0) then
        sign = -1.0
    else
        sign = 1.0
    endif
end function
```

Gegenbauer and Legendre Polynomials

A function to evaluate Gegenbauer $C_n^{-1/2}(\cos(\theta))$ and Legendre polyomials $P_n(\cos(\theta))$ up to order $n = 20$.

```
!-----------------------------------------------------------------------
!> @author
!> H. Arrowood, UNC-CH
!
! DESCRIPTION:
!> Computes the first few Gegenbauer polynomials of degree n.
!> @brief
!> Compute \f$ C^{-1/2}_n(\dcos(\theta) \f$.
!
! REVISION HISTORY:
! 07_Nov_2017 - Added documentation, standardized - H. Arrowood
!
!> @param[in] n
!> @param[in] theta
!> @return GegenbauerC
!-----------------------------------------------------------------------
function GegenbauerC(n,theta)
    use globalinfo
    implicit none
    integer, intent(in)          :: n !>degree of Gegenbauer polynomial
    real(kind=8), intent(in)  :: theta    !>angle
    real(kind=8)              :: GegenbauerC  !>function value


    if(n==2) then
        GegenbauerC=.5*(1.0-dcos(theta)**2)
         return
```

```
end if
if(n==4) then
    GegenbauerC=.125*(1.0-dcos(theta)**2)*(5.0*dcos(theta)**1.0)
    return
end if

if(n==6) then
    GegenbauerC=.0625*(1.0-dcos(theta)**2)*(21.0*dcos(theta)**34.&
    &0*dcos(theta)**3.0)
    return
end if

if(n==8) then
    GegenbauerC=1.0/128.0*(1.0-dcos(theta)**2)*(429.0*dcos(theta)**&
    &6-495.0*dcos(theta)**535.0*dcos(theta)**2-5.0)
    return
end if
if(n==10) then
    GegenbauerC=0.02734375 - (315*Cos(theta)**2)/256. + (1155*Cos(t&
    &heta)**4)/128. - (3003*Cos(theta)**6)/128. + (6435*Cos(theta)*&
    &*8)/256. - (2431*Cos(theta)**10)/256.
    return
end if
if(n==12) then
    GegenbauerC=-0.0205078125 + (693*Cos(theta)**2)/512. - (15015*C&
    &os(theta)**4)/1024. + (15015*Cos(theta)**6)/256. - (109395*Cos&
    &(theta)**8)/1024. + (46189*Cos(theta)**10)/512. - (29393*Cos(t&
    &heta)**12)/1024.
    return
end if
```

```
if(n==14) then
    GegenbauerC=0.01611328125 - (3003*Cos(theta)**2)/2048. + (45045&
    &*Cos(theta)**4)/2048. - (255255*Cos(theta)**6)/2048. + (692835&
    &*Cos(theta)**8)/2048. - (969969*Cos(theta)**10)/2048. + &
    &(676039*Cos(theta)**12)/2048. - (185725*Cos(theta)**14)/2048.
    return
end if
if(n==16) then
    GegenbauerC=        -0.013092041015625 + (6435*Cos(theta)**2)/4&
    &096. - (255255*Cos(theta)**4)/8192. + (969969*Cos(theta)**6)/4&
    &096. - (14549535*Cos(theta)**8)/16384. + (7436429*Cos(theta)**&
    &10)/4096. -(16900975.*Cos(theta)**12)/8192. + (5014575.*Cos(th&
    &eta)**14)/4096. - (9694845*Cos(theta)**16)/32768.
    return
end if
if(n==18) then
    GegenbauerC=0.0109100341796875 - (109395*Cos(theta)**2)/65536. &
    &+ (692835*Cos(theta)**4)/16384. - (6789783*Cos(theta)**6)/1638&
    &4. + (66927861*Cos(theta)**8)/32768. - (185910725*Cos(theta)**&
    &10)/32768. + (152108775*Cos(theta)**12)/16384. - (145422675*Co&
    &s(theta)**14)/16384. + (300540195*Cos(theta)**16)/65536. - (64&
    &822395*Cos(theta)**18)/65536.
    return
end if
if(n==20) then
    GegenbauerC=  -0.009273529052734375 + (230945*Cos(theta)**2)/13&
    &1072. - (14549535*Cos(theta)**4)/262144. + (22309287*Cos(theta&
    &)**6)/32768. - (557732175*Cos(theta)**8)/131072. + (1003917915&
    &*Cos(theta)**10)/65536. - (4411154475*Cos(theta)**12)/131072. &
    &+ (1502700975*Cos(theta)**14)/32768. - (9917826435*Cos(theta)*&
```

```
        &*16)/262144. + (2268783825*Cos(theta)**18)/131072. - (88363159&
        &5*Cos(theta)**20)/262144.
        return
    end if


end function GegenbauerC

!-----------------------------------------------------------------------
!> @author
!> H. Arrowood, UNC-CH
!
! DESCRIPTION:
!> Computes the first few Legendre polynomials of degree n, evaluated
!!at dcos(theta).
!> @brief
!> Compute \f$ P_n(\dcos(\theta) \f$.
!
! REVISION HISTORY:
! 07_Nov_2017 - Added documentation, standardized - H. Arrowood
!
!> @param[in] n
!> @param[in] theta
!> @return LegendreP
!-----------------------------------------------------------------------

function LegendreP(n,theta)
    implicit none

    integer, intent(in)          :: n         !>degree of polynomial
```

```
real(kind=8), intent(in)  :: theta     !>angle
real(kind=8)              :: LegendreP !>value of polynomial

if(n==1) then
    LegendreP=dcos(theta)
     return
end if
if(n==3) then
    LegendreP=.5*(-3.0*dcos(theta)+5.0*dcos(theta)**3)
    return
end if

if(n==5) then
    LegendreP=.125*(15.0*dcos(theta)-70.0*dcos(theta)**3+60.0* &
        &dcos(theta)**5)
    return
end if

if(n==7) then
    LegendreP=.0625*(-35.0*dcos(theta)+315.0*dcos(theta)**3-693.0* &
        &dcos(theta)**5+429.0*dcos(theta)**7)
    return
end if
if(n==9) then
    LegendreP=(315*Cos(theta) - 4620*Cos(theta)**3 + 18018*Cos(thet&
    &a)**5 - 25740*Cos(theta)**7 + 12155*Cos(theta)**9)/128.
    return
end if
if(n==11) then
    LegendreP=(-693*Cos(theta) + 15015*Cos(theta)**3 - 90090*Cos(th&
```

```
    &eta)**5 + 218790*Cos(theta)**7 - 230945*Cos(theta)**9 + 88179*&
    &Cos(theta)**11)/256.
    return
end if
if(n==13) then
    LegendreP=(3003*Cos(theta) - 90090*Cos(theta)**3 + 765765*Cos(t&
    &heta)**5 - 2771340*Cos(theta)**7 + 4849845*Cos(theta)**9 - 405&
    &6234*Cos(theta)**11 + 1300075*Cos(theta)**13)/1024.
    return
end if
if(n==15) then
    LegendreP=        (-6435*Cos(theta) + 255255*Cos(theta)**3 - &
    &2909907*Cos(theta)**5 + 14549535*Cos(theta)**7 - 37182145*Cos&
    &(theta)**9 + 50702925*Cos(theta)**11 - 35102025*Cos(theta)**13&
    & + 9694845*Cos(theta)**15)/2048.
    return
end if
if(n==17) then
    LegendreP=(109395*Cos(theta) - 5542680*Cos(theta)**3 + 81477396&
    &*Cos(theta)**5 - 535422888*Cos(theta)**7 + 1859107250*Cos(thet&
    &a)**9 - 3650610600*Cos(theta)**11 + 4071834900*Cos(theta)**13 &
    &-2404321560*Cos(theta)**15 + 583401555*Cos(theta)**17)/32768.
    return
end if
if(n==19) then
    LegendreP=  (-230945*Cos(theta) + 14549535*Cos(theta)**3 - 2677&
    &11444*Cos(theta)**5 + 2230928700*Cos(theta)**7 - 10039179150&
    &*Cos(theta)**9 + 26466926850*Cos(theta)**11 - 42075627300*Cos(&
    &theta)**13 + 39671305740*Cos(theta)**15 - 20419054425*Cos(thet&
    &a)**17 + 4418157975*Cos(theta)**19)/65536.
```

```
        return
    end if

end function LegendreP
```

Alternative Stress Coefficient Routines

Routines using analytic and numerical derivatives of **w** to evaluate the stress coefficients G_2 and I_2

```
!---------------------------------------------------------------------
!> @author
!> H. Arrowood, UNC-CH
! DESCRIPTION:
!> Uses velocities computed on surface of drop to find the
!>coefficients of the Gegenbauer expansion of the stresses.
!>Run this subroutine right after computing w flow.
!> @brief
!> Numerically integrates w.r.t. theta (trapezoidal method for now)
!> Computes \f$I_n,G_n\f$ from series of the form
!> \f$\Sum_{n=2}^{\infty}I_n(r)C_n^{-1/2}(\cos(\theta)) \f$
!> for velocities and stresses
!> @param[in] wu, wv  Cart components of pertvel near surface of drop
!> @param[in] thetavecforstress  theta-coords of points
!> @param[in] nsurfacepoints   number of integration pts near surface
!> @param[out] I2, G2, G4, G6, G8, G2HO
!---------------------------------------------------------------------
subroutine stresscoeffs
    use globalinfo
    real(kind=8),external                   :: GegenbauerC !>Gegen fcn
    real(kind=8), dimension(nsurfpoints+1) :: e21int, h21int, h22int, &
        &h41int, h61int, h81int, d2h2int, h21intHO
```

```
    real(kind=8)                        :: e21, h21, h22, h41, h61, h81,&
        &d2dh2
        !> gegen coeffs of vel. e21 is coeff of radial velocity for n=2;
        !> hn1, hn2, are coeffs of tangential velocity,
        !>hn1 at r=a+epsilon,and  hn2 at r=a+2*epsilon

!first compute coefficients of wtheta and wr at n evenly spaced points.
!I need two values, for r=a+epsilon and r=a+2*epsilon, in order to
!compute the first and second derivatives of H2, and the rest just at
!r=a+epsilon

!evaluation of wu and wv happens inside pertvelT

!-------Find integrands for stress coeffs-------------------------------
        do istress=1, nsurfpoints+1
    !r derivative of radial velocity for n=2 at r=a+epsilon
        e21int(istress) = -3.0/2.0*(sin(thetavecforstress(istress))*1.0&
        &/eps*wU1(istress)+cos(thetavecforstress(istress))*1.0/eps*wV1(&
        &istress))*sin(thetavecforstress(istress))*cos(thetavecforstres&
        &s(istress))
    !r derivative of tangential velocity at r=a+epsilon
        h21int(istress) =  3.0*(cos(thetavecforstress(istress))*1.0/eps&
        &*wU1(istress)-sin(thetavecforstress(istress))*1.0/eps*wV1(istr&
        &ess))*GegenbauerC(2,thetavecforstress(istress))
        h21intHO(istress) =  3.0*(3.0*(cos(thetavecforstress(istress))*&
        &1.0/eps*wU1(istress)-sin(thetavecforstress(istress))*1.0/eps*w&
        &V1(istress))-1.5*(cos(thetavecforstress(istress))*1.0/eps*wU2(&
        &istress)-sin(thetavecforstress(istress))*1.0/eps*wV2(istress))&
        &+1./3.*(cos(thetavecforstress(istress))*1.0/eps*wU3(istress)-s&
        &in(thetavecforstress(istress))*1.0/eps*wV3(istress)))*Gegenbau&
```

```
        &erC(2,thetavecforstress(istress))
        h41int(istress) = 42.0*(cos(thetavecforstress(istress))*1.0/eps&
            &*wU1(istress)-sin(thetavecforstress(istress))*1.0/eps*wV1(&
            &istress))*GegenbauerC(4,thetavecforstress(istress))
        h61int(istress) = 165.0*(cos(thetavecforstress(istress))*1.0/ep&
            &s*wU1(istress)-sin(thetavecforstress(istress))*1.0/eps*wV1&
            &(istress))*GegenbauerC(6,thetavecforstress(istress))
        h81int(istress) = 420.0*(cos(thetavecforstress(istress))*1.0/ep&
            &s*wU1(istress)-sin(thetavecforstress(istress))*1.0/eps*wV1&
            &(istress))*GegenbauerC(8,thetavecforstress(istress))
        !tangential velocity at r=a+2*epsilon, divided by eps already in
        !preparation for the second derivative w.r.t. r
        h22int(istress) = 3.0*(cos(thetavecforstress(istress))*1.0/eps*&
            &wU2(istress)-sin(thetavecforstress(istress))*1.0/eps*wV2(i&
            &stress))*GegenbauerC(2,thetavecforstress(istress))

        !Second r derivative of tan velocity
        d2h2int = 1/eps*h22int-2/eps*h21int
    end do

!--------Integrate to obtain r derivs of velocity coeffs----------------
!@fixme maybe derive a Gauss quadrature for this instead of using trap.
    e21 =  (2.0*sum(e21int)-e21int(1)-e21int(nsurfpoints+1))* &
        &(pi/(2.0*nsurfpoints))

    h21 = (2.0*sum(h21int)-h21int(1)-h21int(nsurfpoints+1))* &
        &(pi/(2.0*nsurfpoints))
    G2H0 = (2.0*sum(h21intH0)-h21intH0(1)-h21intH0(nsurfpoints+1))* &
        &(pi/(2.0*nsurfpoints))
    h41 = (2.0*sum(h41int)-h41int(1)-h41int(nsurfpoints+1))* &
```

```
        &(pi/(2.0*nsurfpoints))
    h61 = (pi/(2.0*nsurfpoints))*(2.0*sum(h61int)-h61int(1)- &
        &h61int(nsurfpoints+1))
    h81 = (pi/(2.0*nsurfpoints))*(2.0*sum(h81int)-h81int(1)- &
        &h81int(nsurfpoints+1))


    h22 = (pi/(2.0*nsurfpoints))*(2.0*sum(h22int)-h22int(1)- &
        &h22int(nsurfpoints+1))


    d2dh2 = (pi/(2.0*nsurfpoints))*(2.0*sum(d2h2int)-d2h2int(1)-&
        &d2h2int(nsurfpoints+1))



!---now assemble into stress coeffs-----------------------------------

!tangential stress coefficients are just r-derivatives
    G2 = h21
    G4 = h41
    G6 = h61
    G8 = h81

!normal stress coeff for n=2
    I2 = G3.0/2.0*R*d2dh2-2.0*e21

end subroutine stresscoeffs
!
!---------------------------------------------------------------------
```

```
!> @author
!> H. Arrowood, UNC-CH; adapted from work of C. Falcon, UNC-CH
!
! DESCRIPTION:
!> Computes perturbation velocity using trapezoidal integration
!> @brief
!> Computes r derivatives perturbation velocities, using  3D
!> integrals.
!-----------------------------------------------------------------------
subroutine wStressTN () !(wu, wv)
    use globalinfo
    implicit none

    real (kind = 8)    :: wstbackflow(nsurfpoints+1), wstsidesphere(nsu&
        &rfpoints+1), wstbelowsphere(nsurfpoints+1), wstabovesphere(nsu&
        &rfpoints+1),tempbackflowst, tempsidespherest, tempbelowsphere&
        &st, tempabovespherest, G2int(nsurfpoints+1), dwthetadrintegran&
        &d(nsurfpoints+1)
    real (kind=8), external :: w2StIntegrandBackflow,GegenbauerC, &
        &w2StIntegrandBelowSphere, w2StIntegrandPartialSphere,&
        &w2StIntegrandZetaSphere, w2StIntegrandZetaVert, w2StIntegrand&
        &R, w2StIntegrandZ, dwthetadr
    integer (kind=4) :: wsti, wsti2, wsti3
    integer (kind=4) ::  ierr,i, ist, istressnew

!--------For stress calculator-----------------------------------------
!first initialize n evenly-spaced integration points on drop surface
    thetavecforstress=(/(Real(i),i=0,nsurfpoints)/)*pi/Real(nsurfpoints)
!----Initialize the velocities at each region of the perturbed interface
```

```
    wstbackflow     = real(0.0, kind=8)
    wstsidesphere   = real(0.0, kind=8)
    wstbelowsphere  = real(0.0, kind=8)
    wstabovesphere  = real(0.0, kind=8)
    AreaReflux      = real(0.0, kind=8)
    AreaEntrain     = real(0.0, kind=8)

    if (minval(sy) < maxval(sy)) then !check that interface is perturbed
        !print*,"newstress is happening!"
!----------------Initialize Interpolated Interface----------------------

        if (allocated(cwinterpx)) deallocate(cwinterpx, stat=ierr)
        if (allocated(cwinterpy)) deallocate(cwinterpy, stat=ierr)

        allocate(cwinterpx(1000000), stat=ierr)
        if (ierr /= 0) print*, "cwinterpx : Allocation failed"
        !> @fixme  make this abort

        allocate(cwinterpy(1000000), stat=ierr)
        if (ierr /= 0) print*, "cwinterpy : Allocation failed"

!----Do loop over dwthetadr stress evaluation pts----------------------
        do wsti = 1, nsurfpoints+1!+2+XN+

            px=thetavecforstress(wsti)
!--Initialize these integration vars to 0-----------------------------
            cwinternalcount=0.0;
            cwinterpx = 0;
            cwinterpy = 0;
```

```
            tempbackflowst    = real(0.0, kind=8)
            tempsidespherest     = real(0.0, kind=8)
            tempbelowspherest    = real(0.0, kind=8)
            tempabovespherest    = real(0.0, kind=8)

!-----Now actually evaluate final integrals over each region------------
            if(flagb <XN+1 ) then
                call trapz1(w2StIntegrandBackflow, xflagb, sx(XN+1), &
                &numtrapz, tempbackflowst)
            endif
            if (flagu /= 0) then
                call trapz1(w2StIntegrandPartialSphere, real(0.0, kind&
                &=8), xflagl, numtrapz, tempbelowspherest)
                call trapz1(w2StIntegrandZetaSphere, max(-R, sy(1)), R,&
                &numtrapz, tempsidespherest)
                call trapz1(w2StIntegrandZetaVert, R, yend, numtrapz, &
                &tempabovespherest)
            elseif (flagl /= 0) then
                call trapz1(w2StIntegrandPartialSphere, real(0.0, kind=&
                &8), xflagl, numtrapz, tempbelowspherest)
                call trapz1(w2StIntegrandZetaSphere, max(-R, sy(1)), ye&
                &nd, numtrapz, tempsidespherest)
            else
                call trapz1(w2StIntegrandBelowSphere, max(sx(1), real(0&
                &.0, kind=8)), xflagb, numtrapz, tempbelowspherest)
            endif

            wstbackflow(wsti)         = tempbackflowst
            wstbelowsphere(wsti)     = tempbelowspherest
            wstsidesphere(wsti)     = tempsidespherest
```

```
            wstabovesphere(wsti)     = tempabovespherest


        end do
    endif !the if that checks if the interface is perturbed

!----Make vector of dwthetadr values-----------------------------------


        do wsti2=1, nsurfpoints+1
            if(wsti2==nsurfpoints+1) then
                print*,"actual dwthetadrintegrand final element", (-ws&
                &tbackflow(wsti2)+wstbelowsphere(wsti2)+ &
                &wstsidesphere(wsti2)+ wstabovesphere(wsti2))*drhogover8mu
                dwthetadrintegrand(wsti2) = 0.0
            !Detect NaNs due to integrator issues
            elseif(isnan(wstsidesphere(wsti2)))then
                wstsidesphere(wsti2)=0
                dwthetadrintegrand(wsti2) = (-wstbackflow(wsti2)+wstbel&
                &owsphere(wsti2)+ wstsidesphere(wsti2)+ wstabovesphere(&
                &wsti2))*drhogover8mu
            else
                dwthetadrintegrand(wsti2) = (-wstbackflow(wsti2)+wstbel&
                &owsphere(wsti2)+wstsidesphere(wsti2)+ wstabovesphere(w&
                &sti2))*drhogover8mu
            endif
        end do
    do istressnew = 1, nsurfpoints+1
        G2int(istressnew) =  3.0*(dwthetadrintegrand(istressnew))*&
        &GegenbauerC(2,thetavecforstress(istressnew))
    end do
    G2new = (2.0*sum(G2int)-G2int(1)-G2int(nsurfpoints+1))*(pi/(2.0*nsu&
```

```
    &rfpoints))
end subroutine wStressTN

!========== For Velocity Integrands =========!
function w2StIntegrandBackflow(rho)
    use globalinfo
    implicit none
    real (kind=8) rho, zcoord, w2StIntegrandBackflow
    real (kind=8), external :: w2StIntegrandZeta

    call interpbridge( XN+2-max(flagb-2, 1), sx(max(flagb-2,1):XN+1),&
    & sy(max(flagb-2, 1):XN+1), rho, zcoord)
    myrho = rho !this is where the value of myrho is assigned
    call trapz1(w2StIntegrandZeta, yend, zcoord, numtrapz, w2StIntegran&
    &dBackflow)
    cwinternalcount = cwinternalcount +1;
    cwinterpx(cwinternalcount) = rho;
    cwinterpy(cwinternalcount) = zcoord;
end

function w2StIntegrandZetaSphere(zeta)
    use globalinfo
    implicit none
    real (kind=8) zeta, xupper, xlower, w2StIntegrandZetaSphere
    real (kind=8), external :: w2StIntegrandRho

    myzeta  = zeta
    call interpbridge(min(flagb+2, XN+1), sy(1:min(flagb+2, XN+1)), sx(&
    &1:min(flagb+2, XN+1)), zeta, xupper)
    xlower  = sqrt(R**2 - zeta**2)
```

```
    if(isnan(xlower))then !to fix the case due to numerical error where
        xlower=0.0    !xlower became a NaN because zeta was off by 1E-16
    endif
    !Now to check that xupper isn't accidentally inside the drop due to
    !interpolation error
    if(xupper**2+zeta**2<R**2)then
        print*, "upper bound was inside!!"
        xupper=xlower !So basically, if the interpolation accidentally
    endif          !puts it inside, just let the layer thickness go to 0
    call trapz1(w2StIntegrandRho, xlower, xupper, numtrapz, w2StIntegra&
    &ndZetaSphere)
    cwinternalcount = cwinternalcount +1;
    cwinterpx(cwinternalcount) = xupper;
    cwinterpy(cwinternalcount) = zeta;
end

function w2StIntegrandPartialSphere(rho)
    use globalinfo
    implicit none
    real (kind=8) rho, zcoord, w2StIntegrandPartialSphere
    real (kind=8), external :: w2StIntegrandZeta

    myrho = rho
    call interpbridge(min(flagl+3, XN+1), sx(1:min(flagl+3, XN+1)), sy(&
    &1:min(flagl+3, XN+1)), rho, zcoord)
    call trapz1(w2StIntegrandZeta, zcoord, -R, numtrapz, w2StIntegrandP&
    &artialSphere)
    cwinternalcount = cwinternalcount +1;
    cwinterpx(cwinternalcount) = rho;
    cwinterpy(cwinternalcount) = zcoord;
```

```
end


function w2StIntegrandZetaVert(zeta)
    use globalinfo
    implicit none
    real (kind=8) zeta, xupper, w2StIntegrandZetaVert
    real (kind=8), external :: w2StIntegrandRho
    integer (kind=4) temp(1), tempmini

    myzeta  = zeta
    call interpbridge(min(flagb+2, XN+1), sy(1:min(flagb+2, XN+1)), sx(&
    &1:min(flagb+2, XN+1)), zeta, xupper)
    call trapz1(w2StIntegrandRho, real(0.0, kind=8), xupper, numtrapz,&
    &w2StIntegrandZetaVert)
    cwinternalcount = cwinternalcount +1;
    cwinterpx(cwinternalcount) = xupper;
    cwinterpy(cwinternalcount) = zeta;
end



function w2StIntegrandBelowSphere(rho)
    use globalinfo
    implicit none
    real (kind=8) rho, zcoord, w2StIntegrandBelowSphere
    real (kind=8), external :: w2StIntegrandZeta

    call interpbridge(max(flagl+2, XN+1), sx(1:max(flagl+2, XN+1)), sy(&
    &1:max(flagl+2, XN+1)), rho, zcoord)
    myrho   = rho
    call trapz1(w2StIntegrandZeta, zcoord, yend, numtrapz, w2StIntegran&
```

```
    &dBelowSphere)
end

function w2StIntegrandZeta(zeta)
    use globalinfo
    implicit none
    real (kind=8) zeta, w2StIntegrandZeta
    real (kind=8), external :: w2StIntegrandR

    myzeta = zeta
    w2StIntegrandZeta =w2StIntegrandR()
end




function w2StIntegrandRho(rho)
    use globalinfo
    implicit none
    real (kind=8) rho, w2StIntegrandRho
    real (kind=8), external :: w2StIntegrandR!

    myrho = rho
    w2StIntegrandRho = w2StIntegrandR()
end




function w2StIntegrandR
! this is the integrand after theta integration
```

```
    use globalinfo
    implicit none
    real (kind=8) w2StIntegrandR,ILogR
    real (kind=8), external :: dWthetadr

    w2StIntegrandR=0.0;
    ILogR = 0.0;
    if (px > 0.0) then

!for now, assume the pseudosingularities are the same as the velocities

        if (abs (myzeta*px + py * myrho ) < logsing) then
        call trapz1(dWthetadr,  real(0.0,kind=8),real(2.0*pi,kind=8), &
        &logtrapzbig, ILogR)
        else
        call trapz1(dWthetadr,  real(0.0,kind=8),real(2.0*pi,kind=8), &
        &logtrapz, ILogR)
        endif
        w2StIntegrandR = ILogR*myrho!/pi
    endif
end

function dWthetadr(theta)
    use globalinfo
    implicit none
    real (kind=8), intent(in) ::  theta
    real (kind=8) :: dWthetadr,thetax, y1, y2, y3
    real (kind=8) :: ry, dWU1, dWU2, phicoeffderiv, dphidx1term1, dphid&
        &x1term2,dphidx1term3, dphidx1term4, dphidx1term5, dWV1, dWV2, &
        &dphidx3term1, dphidx3term2, dphidx3term3, dphidx3term4, dphidx&
```

```
    &3term5, dWU, dWV


!First, we need the wu and wv terms, differentiated w.r.t. r.
!x-values on drop surface will be input as R (drop radius) from
!globalinfo and a vector of thetax values
thetax = px
y1 = myrho *cos(theta)
y2= myrho* sin(theta)
y3 = myzeta
ry = sqrt(y1**2+y2**2+y3**2)
dWU1 =  -((R**4*(-(R*y3) + (y1**2 + y2**2 + y3**2)*Cos(thetax))*Sin&
    &(thetax))/ ((y1**2 + y2**2 + y3**2)**2.5*((R**2*(R**2 + y1**2 &
    &+ y2**2 + y3**2 - 2*R*y3*Cos(thetax) - 2*R*y1*Sin(thetax)))/(y&
    &1**2 + y2**2 + y3**2))**1.5)) - (R**4*Cos(thetax)*(-(R*y1) + (&
    &y1**2 + y2**2 + y3**2)*Sin(thetax)))/((y1**2 + y2**2 + y3**2)*&
    &*2.5*((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thetax)&
    & - 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2))**1.5&
    & ) + (3*(-(R*y3) + (y1**2 + y2**2 + y3**2)*Cos(thetax))*&
    & Sqrt((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thetax)&
    &- 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2))*(y1**2 + y2**2&
    &+ y3**2 - R*y3*Cos(thetax) - R*y1*Sin(thetax))*&
    & (-(R*y1) + (y1**2 + y2**2 + y3**2)*Sin(thetax)))/&
    &  ((y1**2 + y2**2 + y3**2)**1.5*(R**2 + y1**2 + y2**2 + y3**2 &
    &- 2*R*y3*Cos(thetax) - 2*R*y1*Sin(thetax))**3) - &
    &  (3*(y3 - R*Cos(thetax))*(-y1 + R*Sin(thetax))*(-R + y3*Cos&
    &(thetax) + y1*Sin(thetax)))/(y2**2 + (y3 - R*Cos(thetax))**2 +&
    & (y1 - R*Sin(thetax))**2)**2.5 + ((-y3 + R*Cos(thetax))*Sin(th&
    &etax))/(y2**2 + (y3 - R*Cos(thetax))**2 + (y1 - R*Sin(thetax))&
    &**2)**1.5 + (Cos(thetax)*(-y1 + R*Sin(thetax)))/(y2**2 + (y3 -&
    & R*Cos(thetax))**2 + (y1 - R*Sin(thetax))**2)**1.5
```

```
    dWU2 =          ((-R**2 + y1**2 + y2**2 + y3**2)*(R*Cos(thetax)*&
         &  (-(y1*(2*y1**4 + 2*y2**4 + 3*y2**2*y3**2 + y3**4 - R**2*(y1&
         &**2 + y2**2 - 2*y3**2) +&
         &  y1**2*(4*y2**2 + 3*y3**2))) + R*(y1**4 + y1**2*y2**2 &
         &+ y2**2*y3**2 + y3**4)*Sin(thetax)) + &
         &  y3*(-(R*(y1**4 + R**2*(2*y1**2 - y2**2 - y3**2) + 3*y1**2*&
         &(y2**2 + y3**2) + 2*(y2**2 + y3**2)**2)*Sin(thetax)) +&
         &     y1*((y1**2 + y2**2 + y3**2)*(2*R**2 + y1**2 + y2**2 + &
         &y3**2) + R**2*y1*y3*Sin(2*thetax)))))/&
         &  ((y1**2 + y2**2 + y3**2)**2.5*(R**2 + y1**2 + y2**2 + y3**2 &
         &- 2*R*y3*Cos(thetax) - 2*R*y1*Sin(thetax))**2*&
         &Sqrt((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thetax) &
         &- 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2)))
!Note that the coefficient of the dphi term is zero on the drop surface
!So there is no need to take a derivative of the phi term, just the
!coeff when applying the product rule to this term.
    phicoeffderiv = -((R*(-R**2+ry**2))/ry**3)
    dphidx1term1 = (4*R**3*y1*y3 + 7*R*y1**3*y3 + 7*R*y1*y2**2*y3 + 7*R&
        &*y1*y3**3 - 6*R*y1*y3*(y1**2 + y2**2 + y3**2)*Cos(2*thetax) - &
        &14*R**2*y1**2*y3*Sin(thetax) - 3*y1**4*y3*Sin(thetax) - &
        &6*R**2*y2**2*y3*Sin(thetax) - 6*y1**2*y2**2*y3*Sin(thetax) - &
        &3*y2**4*y3*Sin(thetax) - 6*R**2*y3**3*Sin(thetax) - &
        &6*y1**2*y3**3*Sin(thetax) - 6*y2**2*y3**3*Sin(thetax) - &
        &3*y3**5*Sin(thetax) + R*Cos(thetax)*(R*y1*(3*y1**2 + 3*y2**2 -&
        & 5*y3**2) - 3*(y1**4 + y2**4 - 3*y3**4)*Sin(thetax)) - &
        &3*R*y1**2*y2**2*Sin(2*thetax) + 3*R*y1**2*y3**2*Sin(2*thetax)+&
        & 3*R*y2**2*y3**2*Sin(2*thetax))/(R**2*(R**2 + y1**2 + y2**2 + &
        &y3**2 - 2*R*y3*Cos(thetax) - 2*R*y1*Sin(thetax))**2*&
        &Sqrt((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thetax)&
        & - 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2)))
```

```
dphidx1term2 = (3*R**2*(-(R*y1) + (y1**2 + y2**2 + y3**2)*Sin(theta&
    &x))*(R*(y1**2 + y2**2 - y3**2)*Cos(thetax) + y3*(y1**2 + y2**2&
    & + y3**2 - 2*R*y1*Sin(thetax))))/((y1**2 + y2**2 + y3**2)*((R*&
    &*2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thetax) - 2*R*y1&
    &*Sin(thetax)))/(y1**2 + y2**2 + y3**2))**1.5*(-R**4 + R**3*y3*&
    &Cos(thetax) + R**3*y1*Sin(thetax) +  y1**2*Sqrt(R**4/(y1**2 + &
    &y2**2 + y3**2))*Sqrt((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R&
    &*y3*Cos(thetax) - 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2)&
    &) + y2**2*Sqrt(R**4/(y1**2 + y2**2 + y3**2))*Sqrt((R**2*(R**2 &
    &+ y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thetax) - 2*R*y1*Sin(thet&
    &ax)))/(y1**2 + y2**2 + y3**2)) + y3**2*Sqrt(R**4/(y1**2 + y2**&
    &2 + y3**2))*Sqrt((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*&
    &Cos(thetax) - 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2))))
dphidx1term3 = (-3*(y1**2 + y2**2 + y3**2)*((R**4*Sin(thetax))/Sqrt&
    &(R**4/(y1**2 + y2**2 + y3**2)) + R*y1*(-Sqrt(R**4/(y1**2 + y2&
    &**2 + y3**2)) + Sqrt((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*&
    &R*y3*Cos(thetax) - 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2&
    &))))*(R**3*y3*(2*R**2 + y1**2 + y2**2 + y3**2 - 2*R*y1*Sin(the&
    &tax) - (2*R**2*Sqrt((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*&
    &y3*Cos(thetax) - 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2))&
    &)/Sqrt(R**4/(y1**2 + y2**2 + y3**2))) + Cos(thetax)*(-(R**4*(y&
    &1**2 + y2**2 + 3*y3**2)) + Sqrt(R**4/(y1**2 + y2**2 + y3**2))*&
    &(y1**2 + y2**2 + y3**2)**2*Sqrt((R**2*(R**2 + y1**2 + y2**2 + &
    &y3**2 - 2*R*y3*Cos(thetax) - 2*R*y1*Sin(thetax)))/(y1**2 + y2*&
    &*2 + y3**2)))))/(R**3*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*&
    &Cos(thetax) - 2*R*y1*Sin(thetax))*(-R**4 + R**3*y3*Cos(thetax)&
    & + R**3*y1*Sin(thetax) + y1**2*Sqrt(R**4/(y1**2 + y2**2 + y3**&
    &2))*Sqrt((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thet&
    &ax) - 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2)) + y2**2*Sq&
    &rt(R**4/(y1**2 + y2**2 + y3**2))*Sqrt((R**2*(R**2 + y1**2 + y2&
```

```
     &**2 + y3**2 - 2*R*y3*Cos(thetax) - 2*R*y1*Sin(thetax)))/(y1**2&
     & + y2**2 + y3**2)) + y3**2*Sqrt(R**4/(y1**2 + y2**2 + y3**2))*&
     & Sqrt((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thetax)&
     & - 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2)))**2)
dphidx1term4 = (-3*y3*(y1**2 + y2**2 + y3**2)*Sin(thetax))/&
     &(R**2*(R**4/Sqrt(R**4/(y1**2 + y2**2 + y3**2)) + R*Sqrt(R**2)&
     &*y3*Cos(thetax) + R*Sqrt(R**2)*y1*Sin(thetax)))
dphidx1term5 = (3*(y1**2 + y2**2 + y3**2)*(R*Sqrt(R**2)*y3 + (R**4*&
     &Cos(thetax))/Sqrt(R**4/(y1**2 + y2**2 + y3**2)))*(R*Sqrt(R**2)&
     &*y1 + (R**4*Sin(thetax))/Sqrt(R**4/(y1**2 + y2**2 + y3**2))))/&
     & (R**7*Sqrt(R**2)*((R*Sqrt(R**2))/Sqrt(R**4/(y1**2 + y2**2 +&
     & y3**2)) + y3*Cos(thetax) + y1*Sin(thetax))**2)
WV1 = (-3*(-y3 + R*Cos(thetax))**2*(2*Cos(thetax)*(-y3 + R*Cos(thet&
     &ax)) + 2*Sin(thetax)*(-y1 + R*Sin(thetax))))/(2.*(y2**2 + (-y3&
     & + R*Cos(thetax))**2 + (-y1 + R*Sin(thetax))**2)**2.5) +(2*Cos&
     &(thetax)*(-y3 + R*Cos(thetax)))/(y2**2 + (-y3 + R*Cos(thetax))&
     &**2 + (-y1 + R*Sin(thetax))**2)**1.5 - (2*Cos(thetax)*(-y3 + R&
     &*Cos(thetax)) + 2*Sin(thetax)*(-y1 + R*Sin(thetax)))/ (2.*(y2*&
     &*2 + (-y3 + R*Cos(thetax))**2 + (-y1 + R*Sin(thetax))**2)**1.5&
     &) + (3*R**3*(-((R**2*y3)/(y1**2 + y2**2 + y3**2)) + R*Cos(thet&
     &ax))**2*(2*Cos(thetax)*(-((R**2*y3)/(y1**2 + y2**2 + y3**2)) +&
     & R*Cos(thetax)) + 2*Sin(thetax)*(-((R**2*y1)/(y1**2 + y2**2 + &
     &y3**2)) + R*Sin(thetax))))/(2.*(y1**2 + y2**2 + y3**2)**1.5*((&
     &R**4*y2**2)/(y1**2 + y2**2 + y3**2)**2 + (-((R**2*y3)/(y1**2 +&
     & y2**2 + y3**2)) + R*Cos(thetax))**2 + (-((R**2*y1)/(y1**2 + y&
     &2**2 + y3**2)) + R*Sin(thetax))**2)**2.5) - (2*R**3*Cos(thetax&
     &)*(-((R**2*y3)/(y1**2 + y2**2 + y3**2)) + R*Cos(thetax)))/((y1&
     &**2 + y2**2 + y3**2)**1.5*((R**4*y2**2)/(y1**2 + y2**2 + y3**2&
     &)**2 + (-((R**2*y3)/(y1**2 + y2**2 + y3**2)) + R*Cos(thetax))*&
     &*2 + (-((R**2*y1)/(y1**2 + y2**2 + y3**2)) + R*Sin(thetax))**2&
```

```
     &)**1.5) + (R*(2*Cos(thetax)*(-((R**2*y3)/(y1**2 + y2**2 + y3**&
     &2)) + R*Cos(thetax)) + 2*Sin(thetax)*(-((R**2*y1)/(y1**2 + y2*&
     &*2 + y3**2)) + R*Sin(thetax))))/(2.*Sqrt(y1**2 + y2**2 + y3**2&
     &)*((R**4*y2**2)/(y1**2 + y2**2 + y3**2)**2 + (-((R**2*y3)/(y1*&
     &*2 + y2**2 + y3**2)) + R*Cos(thetax))**2 + (-((R**2*y1)/(y1**2&
     & + y2**2 + y3**2)) + R*Sin(thetax))**2)**1.5)
WV2 = (y3*(-R**2+ry**2)*(2.*R**2*y1**2*y3+y1**4*y3+2.*R**2*y2**2*y&
     &3+2.*y1**2*y2**2*y3+y2**4*y3+2.*R**2*y3**3+2.*y1**2*y3**3+2.*y&
     &2**2*y3**3+y3**5+R*(-4.*y1**4-4.*y2**4-7.*y2**2*y3**2-3.*y3**4&
     &+R**2*(2.*y1**2+2.*y2**2-y3**2)-y1**2*(8.*y2**2+7.*y3**2))*cos&
     &(thetax)+R**2*y3*ry**2*cos(2.*thetax)-3.*R**3*y1*y3*sin(thetax&
     &)+R*y1**3*y3*sin(thetax)+R*y1*y2**2*y3*sin(thetax)+R*y1*y3**3*&
     &sin(thetax)+R**2*y1**3*sin(2.*thetax)+R**2*y1*y2**2*sin(2.*the&
     &tax)+R**2*y1*y3**2*sin(2.*thetax)))/(ry**5*(R**2+ry**2-2.*R*y3&
     &*cos(thetax)-2.*R*y1*sin(thetax))**2*sqrt((R**2*(R**2+ry**2-2.&
     &*R*y3*cos(thetax)-2.*R*y1*sin(thetax)))/ry**2))
dphidx3term1 = (-y3*(3.*ry**4+R**2*(5.*y1**2+5.*y2**33.*y3**2))*&
     &cos(thetax)+1./2.*R*(2.*R**2*y1**2-y1**4+2.*R**2*y2**2-2.*y1**&
     &2*y2**2-y2**50.*R**2*y3**32.*y1**2*y3**32.*y2**2*y3**3&
     &3.*y3**1.*(y1**4+y2**4-2.*y2**2*y3**2-3.*y3**4+2.*y1**2*(y2*&
     &*2-y3**2))*cos(2.*thetax)-4.*R*y1*(y1**2+y2**2+5.*y3**2)*sin(t&
     &hetax)+12.*y1**3*y3*sin(2.*thetax)+12.*y1*y2**2*y3*sin(2.*thet&
     &ax)+12.*y1*y3**3*sin(2.*thetax)))/(R**2*(R**2+ry**2-2.*R*y3*co&
     &s(thetax)-2.*R*y1*sin(thetax))**2*sqrt((R**2*(R**2+ry**2-2.*R*&
     &y3*cos(thetax)-2.*R*y1*sin(thetax)))/ry**2))
dphidx3term2 = (3*(y1**2 + y2**2 + y3**2)**2*((R**6*(R*y3 - (y1**2&
     & + y2**2 + y3**2)*Cos(thetax))**2)/(y1**2 + y2**2 + y3**2)**3 &
     &- (R**5*y3*(R*y3 - (y1**2 + y2**2 + y3**2)*Cos(thetax))*(R**2 &
     &+ y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thetax) - 2*R*y1*Sin(thet&
     &ax)))/(y1**2 + y2**2 + y3**2)**3 - ((R**4/(y1**2 + y2**2 + y3*&
```

```
     &*2))**1.5*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thetax) -&
     & 2*R*y1*Sin(thetax))*(Sqrt(R**4/(y1**2 + y2**2 + y3**2)) -&
     &Sqrt((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thetax) &
     & - 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2))))/R**2))/&
     &  (R**3*((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thet&
     &ax) - 2*R*y1*Sin(thetax)))/&
     &  (y1**2 + y2**2 + y3**2))**1.5*(-R**4 + R**3*y3*Cos(thetax) +&
     & R**3*y1*Sin(thetax) + y1**2*Sqrt(R**4/(y1**2 + y2**2 + y3**2)&
     &)*Sqrt((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thetax&
     &) - 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2)) + y2**2*Sqrt&
     &(R**4/(y1**2 + y2**2 + y3**2))*Sqrt((R**2*(R**2 + y1**2 + y2**&
     &2 + y3**2 - 2*R*y3*Cos(thetax) - 2*R*y1*Sin(thetax)))/(y1**2 +&
     & y2**2 + y3**2)) + y3**2*Sqrt(R**4/(y1**2 + y2**2 + y3**2))*&
     &Sqrt((R**2*(R**2 + y1**2 + y2**2 + y3**2 - 2*R*y3*Cos(thetax)&
     & - 2*R*y1*Sin(thetax)))/(y1**2 + y2**2 + y3**2))))
   dphidx3term3 = -((3.*ry**2*((R**4*cos(thetax))/(R**2/ry)+R*y3*&
     &(-(R**2/ry)+sqrt((R**2*(R**2+ry**2-2.*R*y3*cos(thetax)-2.*R*y1&
     &*sin(thetax)))/(ry**2))))*(R**3*y3*(2.*R**2+ry**2-2.*R*y1*sin(&
     &thetax)-(2.*R**2*sqrt((R**2*(R**2+ry**2-2.*R*y3*cos(thetax)-2.&
     &*R*y1*sin(thetax)))/ry**2))/(R**2/ry))+cos(thetax)*(-R**4*(y1*&
     &*2+y2**2+3.*y3**2)+(R**2/ry)*ry**4*sqrt((R**2*(R**2+ry**2-2.*R&
     &*y3*cos(thetax)-2.*R*y1*sin(thetax)))/(ry**2)))))/(R**3*(R**2+&
     &ry**2-2.*R*y3*cos(thetax)-2.*R*y1*sin(thetax))*(-R**4+R**3*y3*&
     &cos(thetax)+R**3*y1*sin(thetax)+y1**2*(R**2/ry)*sqrt((R**2*(R*&
     &*2+ry**2-2.*R*y3*cos(thetax)-2.*R*y1*sin(thetax)))/ry**2)+y2**&
     &2*(R**2/ry)*sqrt((R**2*(R**2+ry**2-2.*R*y3*cos(thetax)-2.*R*y1&
     &*sin(thetax)))/ry**2)+y3**2*(R**2/ry)*sqrt((R**2*(R**2+ry**2-2&
     &.*R*y3*cos(thetax)-2.*R*y1*sin(thetax)))/ry**2))**2))
   dphidx3term4 = -((3.*ry**2*(ry+y3*cos(thetax)))/(R**2*(R**2*ry+&
     &R**2*y3*cos(thetax)+R**2*y1*sin(thetax))))
```

```
      dphidx3term5 = (3.*ry**2*(R**2*y3+(R**4*cos(thetax))/(R**2/ry))**&
         &2)/(R**7*R*(ry+y3*cos(thetax)+y1*sin(thetax))**2)


     dWU = dWU1+dWU2+phicoeffderiv*(dphidx1term1+dphidx1term2+&
         &dphidx1term3+dphidx1term4+dphidx1term5)
     dWV = dWV1+dWV2+phicoeffderiv*(dphidx3term1+dphidx3term2+&
         &dphidx3term3+dphidx3term4+dphidx3term5)
         dWthetadr = cos(thetax)*dWU - sin(thetax)*dWV
         return
end
```

Final Stress Coefficient Routine

Final routine for evaluating G_2; uses reciprocal theorem result to avoid numerical difficulties.

```
function stresstail1DG2 ( xvalG2temp)
    use globalinfo
    implicit none
    real (kind=8) :: eta, stresstail1DG2, xvalG2temp
    real (kind=8), external :: G2IntegrandY
    integer (kind=4) :: startingi


    xvalG2=xvalG2temp
    startingi = max(flagb-2, 1)
    call interpbridge(XN+2-startingi, sx(startingi:XN+1), sy(startingi:&
    &XN+1), xvalG2, eta)
    call trapz1(G2IntegrandY, yend, eta, numtrapz, stresstail1DG2)
    cinternalcount = cinternalcount +1;
    cinterpy(cinternalcount) = eta;
    cinterpx(cinternalcount) = xvalG2;
```

```
end

function stressIntegrandFlat1DG2 ( xvalG2temp )
    use globalinfo
    implicit none
    real (kind=8) ::  eta, xvalG2temp, stressIntegrandFlat1DG2
    real (kind=8), external :: G2IntegrandY
    integer (kind=4) :: endingi

    xvalG2=xvalG2temp
    endingi = min(flagb+2, XN+1)
    call interpbridge( endingi, sx(1:endingi), sy(1:endingi), xvalG2,&
    & eta)
    call trapz1(G2IntegrandY, yend, eta, numtrapz, stressIntegrandFlat&
    &1DG2)
    cinternalcount = cinternalcount +1;
    cinterpy(cinternalcount) = eta;
    cinterpx(cinternalcount) = xvalG2;
end

function stressIntegrandsphere1DG2(yvalG2temp)
    use globalinfo
    implicit none
    real (kind=8) eta, yvalG2temp, stressIntegrandsphere1DG2, xmin
    real (kind=8), external :: G2IntegrandX
    integer (kind=4) tempflagb

    yvalG2=yvalG2temp
    tempflagb = min(flagb+2, XN+1)
```

```
    call interpbridge( tempflagb, sy(1:tempflagb), sx(1:tempflagb), yva&
    &lG2, eta )
    xmin = Sqrt(R**2-yvalG2**2)
    if(isnan(xmin))then
        xmin = 0.
    endif
    call trapz1(G2IntegrandX, eta, xmin, numtrapz, stressIntegrandspher&
    &e1DG2)
    cinternalcount = cinternalcount +1;
    cinterpy(cinternalcount) = yvalG2;
    cinterpx(cinternalcount) = eta;
end



function stressIntegrand1DG2(yvalG2temp) !First integration is in X
    use globalinfo
    implicit none
    integer (kind=4) tempflagb
    real (kind=8) eta, yvalG2temp, stressIntegrand1DG2
    real (kind=8), external :: G2IntegrandX

    yvalG2 = yvalG2temp
    tempflagb = min(flagb+2, XN+1)
    call interpbridge ( tempflagb, sy(1:tempflagb), sx(1:tempflagb), yv&
    &alG2, eta )
    call trapz1(G2IntegrandX,  eta, 0.0, numtrapz, stressIntegrand1DG2)
    cinternalcount = cinternalcount +1;
    cinterpy(cinternalcount) = yvalG2;
    cinterpx(cinternalcount) = eta;
end
```

```
function G2INtegrandY(yg)
    use globalinfo
    implicit none
    real (kind=8), intent(in) ::  yg
    real (kind=8) :: thet, rt, G2IntegrandY
    real (kind=8) :: T2, T4, T6, T8, T10, T12, T14, T16, T18, T20
    real (kind=8) :: LegendreP, GegenbauerC
    real (kind=8) :: taur, taur2, taur4, taur6, taur8, taur10,taur12,ta&
        &ur14, taur16, taur18,taur20, taut, taut2, taut4, taut6, taut8,&
        & taut10,taut12, taut14, taut16, taut18, taut20


    rt = sqrt(xvalG2**2+yg**2)
    thet = acos(yg/rt)


    T2 = 3.*pi/4.
    T4 = 21.*pi/32.
    T6 = 165.*pi/256.
    T8 = 2625.*pi/4096.
    T10 = 41895.*pi/65536.
    T12 = (334719.*pi)/524288.
    T14 = (2625673.*pi)/(4194304.)
    T16 =(85579065.*pi)/134217728.
    T18 =(2737609875.*pi)/4294967296.
    T20 =(21895664505.*pi)/34359738368.


    taur2 = -LegendreP(1,thet)*T2/2.*(R**(3)/rt**(3)-R**(1)/rt**(1))
    taur4 = -LegendreP(3,thet)*T4/2.*(R**(5)/rt**(5)-R**(3)/rt**(3))
    taur6 = -LegendreP(5,thet)*T6/2.*(R**(7)/rt**(7)-R**(5)/rt**(5))
    taur8 = -LegendreP(7,thet)*T8/2.*(R**(9)/rt**(9)-R**(9)/rt**(7))
```

```
taur10 = -LegendreP(9,thet)*T10/2.*(R**(11)/rt**(11)-R**(9)/rt**(9))
taur12 = -LegendreP(11,thet)*T12/2.*(R**(13)/rt**(13)-R**(11)/rt**&
&(11))
taur14 = -LegendreP(13,thet)*T14/2.*(R**(15)/rt**(15)-R**(13)/rt**&
&(13))
taur16 = -LegendreP(15,thet)*T16/2.*(R**(17)/rt**(17)-R**(15)/rt**&
&(15))
taur18 = -LegendreP(17,thet)*T18/2.*(R**(19)/rt**(19)-R**(17)/rt**&
&(17))
taur20 = -LegendreP(19,thet)*T20/2.*(R**(21)/rt**(21)-R**(19)/rt**&
&(19))
taur=taur2+taur4+taur6+taur8+taur10+taur12+taur14+taur16+taur18+&
&taur20

if(thet==0.0)then
    taut2 = 0.0
    taut4 = 0.0
    taut6 = 0.0
    taut8 = 0.0
    taut10 = 0.0
    taut12 = 0.0
    taut14 = 0.0
    taut16 = 0.0
    taut18 = 0.0
    taut20 = 0.0
else
    taut2 = GegenbauerC(2,thet)/sin(thet)*(T2/2.)*(-1*R**(3)/&
    &rt**(3)+(-1)*R**(1)/rt**(1))
    taut4 = GegenbauerC(4,thet)/sin(thet)*(T4/2.)*(-(3)*R**(5)/&
    &rt**(5)+(1)*R**(3)/rt**(3))
```

```
        taut6 = GegenbauerC(6,thet)/sin(thet)*(T6/2.)*(-(5)*R**(7)/&
        &rt**(7)+(3)*R**(5)/rt**(5))
        taut8 = GegenbauerC(8,thet)/sin(thet)*(T8/2.)*(-(7)*R**(9)/&
        &rt**(9)+(5)*R**(7)/rt**(7))
        taut10 = GegenbauerC(10,thet)/sin(thet)*(T10/2.)*(-(9)*R**&
        &(11)/rt**(11)+(7)*R**(9)/rt**(9))
        taut12 = GegenbauerC(12,thet)/sin(thet)*(T12/2.)*(-(11)*R**&
        &(13)/rt**(13)+(9)*R**(11)/rt**(11))
        taut14 = GegenbauerC(14,thet)/sin(thet)*(T14/2.)*(-(13)*R**&
        &(15)/rt**(15)+(11)*R**(13)/rt**(13))
        taut16 = GegenbauerC(16,thet)/sin(thet)*(T16/2.)*(-(15)*R**&
        &(17)/rt**(17)+(13)*R**(15)/rt**(15))
        taut18 = GegenbauerC(18,thet)/sin(thet)*(T18/2.)*(-(17)*R**&
        &(19)/rt**(19)+(15)*R**(17)/rt**(17))
        taut20 = GegenbauerC(20,thet)/sin(thet)*(T20/2.)*(-(19)*R**&
        &(21)/rt**(21)+(17)*R**(19)/rt**(19))
    endif
    taut = taut2+taut4+taut6+taut8+taut10+taut12+taut14+taut16+taut18+taut20
    G2IntegrandY = -xvalG2*(cos(thet)*taur+sin(thet)*taut)*g
    return
end function G2IntegrandY

function G2IntegrandX(xg)
    use globalinfo
    implicit none
    real (kind=8), intent(in) ::  xg
    real (kind=8) :: G2IntegrandX
    real (kind=8) ::rt,thet, T2, T4, T6, T8, T10, T12, T14, T16, T18, T20
    real (kind=8) :: LegendreP, GegenbauerC
    real (kind=8) :: taur, taur2, taur4, taur6, taur8, taur10,taur12, &
```

```
    &taur14, taur16, taur18,taur20, taut, taut2, taut4, taut6, taut8,&
    & taut10,taut12, taut14, taut16, taut18, taut20


rt = sqrt(xg**2+yvalG2**2)
thet = acos(yvalG2/rt)
T2 = 3.*pi/4.
T4 = 21.*pi/32.
T6 = 165.*pi/256.
T8 = 2625.*pi/4096.
T10 = 41895.*pi/65536.
T12 = (334719.*pi)/524288.
T14 = (2625673.*pi)/(4194304.)
T16 =(85579065.*pi)/134217728.
T18 =(2737609875.*pi)/4294967296.
T20 =(21895664505.*pi)/34359738368.
    taur2 = -LegendreP(1,thet)*T2/2.*(R**(3)/rt**(3)-R**(1)/rt**(1))
taur4 = -LegendreP(3,thet)*T4/2.*(R**(5)/rt**(5)-R**(3)/rt**(3))
taur6 = -LegendreP(5,thet)*T6/2.*(R**(7)/rt**(7)-R**(5)/rt**(5))
taur8 = -LegendreP(7,thet)*T8/2.*(R**(9)/rt**(9)-R**(9)/rt**(7))
taur10 = -LegendreP(9,thet)*T10/2.*(R**(11)/rt**(11)-R**(9)/rt**(9))
taur12 = -LegendreP(11,thet)*T12/2.*(R**(13)/rt**(13)-R**(11)/rt**&
&(11))
taur14 = -LegendreP(13,thet)*T14/2.*(R**(15)/rt**(15)-R**(13)/rt**&
&(13))
taur16 = -LegendreP(15,thet)*T16/2.*(R**(17)/rt**(17)-R**(15)/rt**&
&(15))
taur18 = -LegendreP(17,thet)*T18/2.*(R**(19)/rt**(19)-R**(17)/rt**&
&(17))
taur20 = -LegendreP(19,thet)*T20/2.*(R**(21)/rt**(21)-R**(19)/rt**&
&(19))
```

```
taur=taur2+taur4+taur6+taur8+taur10+taur12+taur14+taur16+taur18+&
&taur20
if(thet==0.0)then
    taut2 = 0.0
    taut4 = 0.0
    taut6 = 0.0
    taut8 = 0.0
    taut10 = 0.0
    taut12 = 0.0
    taut14 = 0.0
    taut16 = 0.0
    taut18 = 0.0
    taut20 = 0.0
else
    taut2 = GegenbauerC(2,thet)/sin(thet)*(T2/2.)*(-(1)*R**(3)&
    &/rt**(3)+(-1)*R**(1)/rt**(1))
    taut4 = GegenbauerC(4,thet)/sin(thet)*(T4/2.)*(-(3)*R**(5)&
    &/rt**(5)+(1)*R**(3)/rt**(3))
    taut6 = GegenbauerC(6,thet)/sin(thet)*(T6/2.)*(-(5)*R**(7)&
    &/rt**(7)+(3)*R**(5)/rt**(5))
    taut8 = GegenbauerC(8,thet)/sin(thet)*(T8/2.)*(-(7)*R**(9)&
    &/rt**(9)+(5)*R**(7)/rt**(7))
    taut10 = GegenbauerC(10,thet)/sin(thet)*(T10/2.)*(-(9)*R**&
    &(11)/rt**(11)+(7)*R**(9)/rt**(9))
    taut12 = GegenbauerC(12,thet)/sin(thet)*(T12/2.)*(-(11)*R**&
    &(13)/rt**(13)+(9)*R**(11)/rt**(11))
    taut14 = GegenbauerC(14,thet)/sin(thet)*(T14/2.)*(-(13)*R**&
    &(15)/rt**(15)+(11)*R**(13)/rt**(13))
    taut16 = GegenbauerC(16,thet)/sin(thet)*(T16/2.)*(-(15)*R**&
    &(17)/rt**(17)+(13)*R**(15)/rt**(15))
```

```
        taut18 = GegenbauerC(18,thet)/sin(thet)*(T18/2.)*(-(17)*R**&
        &(19)/rt**(19)+(15)*R**(17)/rt**(17))
        taut20 = GegenbauerC(20,thet)/sin(thet)*(T20/2.)*(-(19)*R**&
        &(21)/rt**(21)+(17)*R**(19)/rt**(19))
    endif
    taut = taut2+taut4+taut6+taut8+taut10+taut12+taut14+taut16+taut18&
    &+taut20
    G2IntegrandX = -xg*(cos(thet)*taur+sin(thet)*taut)*g
    return
end function G2IntegrandX
```

Trapezoidal Integration

```
subroutine trapz1(f,a,b,h1,r)
!============================================================
! int_trap.f: integration by trapezoid rule of f(x) on [a,b]
!------------------------------------------------
! f     - Function to integrate (supplied above)
! a - Lower limit of integration
! b - Upper limit of integration
! R - Result of integration (out)
! n - number of intervals
!================================================
    real (kind=8) :: a, b, f, r ,dx, x,h,h1
    integer (kind=4) :: n, i

    h =  min ((b-a)/4, h1)
    !correct to even
    n= (b-a)/h + 1
    n=2*ceiling(((b-a)/h + 1)/2)
    r = 0.d0
```

```
    dx = (b-a)/(n-1)
    do i=2,n-1
        x = a+(i-1)*dx
        r = r + f(x)
    end do
    r = (r + (f(a)+f(b))/2.d0)*dx
    return
end subroutine trapz1
```

Makefile

```
FC= ifort
#FCFLAGS = -check bounds -g -O0 -traceback #Debug flags
#~ FCFLAGS = -check bounds -g -O0 -traceback -fopenmp
FCFLAGS = -g  -xHost  -r8 #-fopenmp #Release flags -fast -O3

#F95FLAGS    = -g

PROGRAMS    = DropletCode

RM        = /bin/rm -f

SHELL        = /bin/sh

all:    $(PROGRAMS)

clean:
    -$(RM) *.i
    -$(RM) *.o
    -$(RM) *~
    -$(RM) \#*
```

```
	-$(RM) a.out
	-$(RM) core core.*
	-$(RM) *.dat
	-$(RM) *.png

clobber:	distclean

distclean:	mostlyclean
	-$(RM) $(PROGRAMS)

DropletCode: main.o Besseli.o Besselk.o gegenC.o cylindervel.o
ellipke2.o fft.o interp.o  intlib3.o pertvelT.o rk4.o stress.o
stressG2.o stressI2.o stressA.o trapz1.o trapzN.o wstresscoeffcalculator.o
 StressCalculatorNew.o NewtonInterp.o
	$(FC) $(FCFLAGS) $(LDFFLAGS) -o $@ $^
Besseli.o:Besseli.f90
	$(FC) $(FCFLAGS) -c  $<
Besselk.o:Besselk.f90
	$(FC) $(FCFLAGS) -c  $<
gegenC.o:gegenC.f90
	$(FC) $(FCFLAGS) -c  $<
cylindervel.o:cylindervel.f90
	$(FC) $(FCFLAGS) -c  $<
ellipke2.o:ellipke2.f90
	$(FC) $(FCFLAGS) -c  $<
fft.o:fft.f90
	$(FC) $(FCFLAGS) -c  $<
main.o:main.f90
	$(FC) $(FCFLAGS) -c  $<
interp.o:interp.f90
```

```
    $(FC) $(FCFLAGS) -c  $<
intlib3.o:intlib3.f90
    $(FC) $(FCFLAGS) -c  $<
trapz1.o:trapz1.f90
    $(FC) $(FCFLAGS) -c  $<
trapzN.o:trapzN.f90
    $(FC) $(FCFLAGS) -c  $<
pertvelT.o:pertvelT.f90
    $(FC) $(FCFLAGS) -c  $<
rk4.o:rk4.f90
    $(FC) $(FCFLAGS) -c  $<
stress.o:stress.f90
    $(FC) $(FCFLAGS) -c  $<
stressG2.o:stressG2.f90
    $(FC) $(FCFLAGS) -c  $<
stressI2.o:stressI2.f90
    $(FC) $(FCFLAGS) -c  $<
stressA.o:stressA.f90
    $(FC) $(FCFLAGS) -c  $<
wstresscoeffcalculator.o:wstresscoeffcalculator.f90
    $(FC) $(FCFLAGS) -c  $<
StressCalculatorNew.o:StressCalculatorNew.f90
    $(FC) $(FCFLAGS) -c  $<
NewtonInterp.o:NewtonInterp.f90
    $(FC) $(FCFLAGS) -c  $<
maintainer-clean:    distclean
    @echo "This command is intended for maintainers to use;"
    @echo "it deletes files that may require special tools to rebuild."

mostlyclean: clean
```