

2003

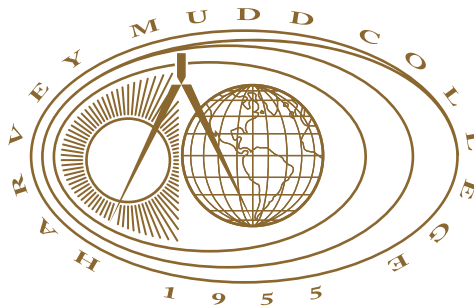
Fluid Drop Coalescence in a Hele-Shaw Cell

Daniel Gianotti
Harvey Mudd College

Recommended Citation

Gianotti, Daniel, "Fluid Drop Coalescence in a Hele-Shaw Cell" (2003). *HMC Senior Theses*. 147.
https://scholarship.claremont.edu/hmc_theses/147

This Open Access Senior Thesis is brought to you for free and open access by the HMC Student Scholarship at Scholarship @ Claremont. It has been accepted for inclusion in HMC Senior Theses by an authorized administrator of Scholarship @ Claremont. For more information, please contact scholarship@cuc.claremont.edu.



Fluid Drop Coalescence in a Hele-Shaw Cell

by
Daniel Gianotti
Ali Nadim, Advisor

Advisor: _____

Second Reader: _____

(Andrew Bernoff)

May 2003

Department of Mathematics

HARVEY MUDD
COLLEGE

Abstract

Fluid Drop Coalescence in a Hele-Shaw Cell

by Daniel Gianotti

May 2003

A fluid drop in a Hele-Shaw cell moves due to surface tension-driven potential flow. Using equations for the pressure and the Green's function for the Laplace Equation, we can formulate an integral equation that determines the motion of the boundary of the drop. By discretizing the boundary contour and following the motion of boundary nodes, the time evolution of the drop can be determined from initial conditions. Results of a numerical simulation show the movement of a drop relaxing from coalescence and the motion of a drop undergoing electrowetting.

Table of Contents

| | |
|---|------------|
| List of Figures | iii |
| Chapter 1: Introduction | 1 |
| Chapter 2: Potential Flow in a Hele-Shaw Cell | 3 |
| 2.1 Hele-Shaw Cells | 3 |
| 2.2 Potential Flow | 4 |
| 2.3 Flow in Bounded Regions | 5 |
| Chapter 3: Curvature | 7 |
| 3.1 Curvature in the Plane of the Cell | 7 |
| 3.2 Curvature in the z-direction | 9 |
| 3.3 Curvature-Driven Motion | 10 |
| Chapter 4: Formulating an Integral Equation | 12 |
| 4.1 Preliminaries | 12 |
| 4.2 Equation for constant external pressure | 12 |
| 4.3 Equations for non-constant pressure | 14 |
| Chapter 5: Discretization | 15 |
| Chapter 6: Simulated Results | 17 |
| 6.1 Computational Model | 17 |
| 6.2 Numerical Results | 18 |

| | | |
|---------------------|--|-----------|
| 6.3 | Associated Problems and Solutions | 21 |
| Chapter 7: | Summary and Conclusions | 23 |
| Appendix A: | Potential Functions and Green's Second Identity | 25 |
| Appendix B: | Formulation of Green's Function | 26 |
| Appendix C: | Integral Equation Regularization | 29 |
| Appendix D: | Simulation Source Files | 31 |
| Bibliography | | 52 |

List of Figures

| | | |
|-----|---|----|
| 1.1 | The three-dimensional coalescence process | 2 |
| 2.1 | A drop within a Hele-Shaw cell | 4 |
| 3.1 | The vectors of the Frenet frame at a point along a contour | 8 |
| 3.2 | The profile of a drop between the plates of a Hele-Shaw cell | 9 |
| 3.3 | Radius of curvature of a drop in the plane perpendicular to the cell . | 10 |
| 6.1 | 100 timesteps of a drop's evolution from initial conditions with a near-singularity | 18 |
| 6.2 | A drop subject to x^3 voltage in the $x < 0$ half-plane | 19 |
| 6.3 | A drop evolving shortly after simulated unification of two semicir- cular drops | 20 |
| 6.4 | Drop evolution with variable-length time-stepping immediately fol- lowing simulated drop unification | 21 |
| 7.1 | Electrowetting research from Nanolytics Inc. | 24 |

Acknowledgments

I would like to thank Ali Nadim for all of the time, knowledge, and advice he has contributed to this project and to my own education. Thank you also to Dmitriy Kogan for salvaging my year's work time-and-again. Ben Bryant, thank you for reading versions of this without laughing. And thank you, finally, Professor Bernoff for organizing the whole shebang.

I'm especially grateful that all of you are so patient.

Chapter 1

Introduction

Coalescence is a ubiquitous natural occurrence, ranging in physical scales from lakes and seas to droplets of water vapor. In a laboratory environment as well, fluid coalescence can play an important role in the forms of mixing, emulsification, and rheology. In a Hele-Shaw cell, drop evolution can be studied within the context of two-dimensional potential flow to gain insight into the fundamentals of coalescence.

By reformulating Laplace's Equation for pressure into a boundary integral equation, we can articulate a fluid's attributes in terms of local curve geometry. This allows us to numerically model the behavior of a drop or region purely in terms of a single boundary curve. Through proper choice of simulated initial conditions, we can recreate the moment immediately following two-drop coalescence and follow the drop through its relaxation process.

In addition to changing initial conditions, we can change the properties of the Hele-Shaw cell to effect the motion of the contained fluid. An electric potential across a fluid drop can change the local surface tension and pressure, causing a drop to move through electrowetting. Some researchers are making use of this method to create *nanolaboratories* in which nanoliter-sized drops of different composition are combined on small chips.

Starting from the work of Nadim, Borhan, and Haj-Hariri [12], we study the motion of droplets as they move and coalesce under wetting methods. We be-

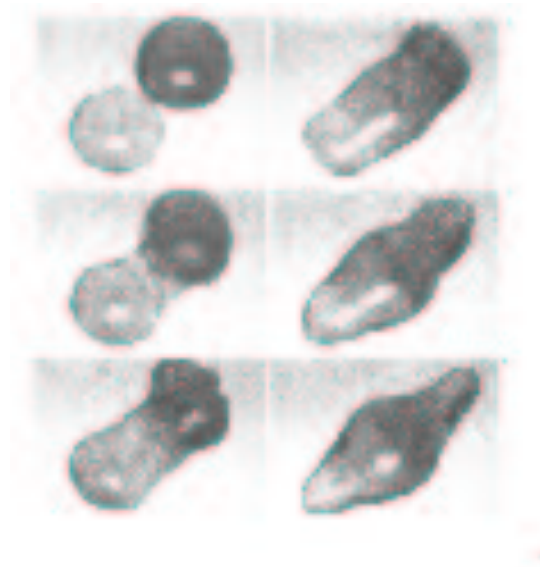


Figure 1.1: The three-dimensional coalescence process

gin by investigating the basics of two-dimensional potential flow in Chapter 2. In Chapters 3 and 4 we formulate an integral equation for the normal velocity of a drop boundary in terms of the local boundary curvature. We then convert our integral equations into a corresponding matrix equation in Chapter 5, in order to computationally model the drop's motion. Results of a numerical simulation are presented in Chapter 6.

Chapter 2

Potential Flow in a Hele-Shaw Cell

This chapter discusses certain key properties of Hele-Shaw cells and the fluid mechanical equations which govern them. Section 2.1 explains the basic definition of a Hele-Shaw cell, and Section 2.2 shows how the Hele-Shaw equation is formulated, leading into the idea of potential flow.

2.1 Hele-Shaw Cells

A Hele-Shaw cell is comprised of two parallel plates of rigid, clear material with spacers placed between them to create a thin gap. The gap is constant in height throughout the cell and is generally much smaller than the dimensions of the plates. If the edges of the plates are sealed, the interior of cell can be partially evacuated to establish a non-atmospheric base pressure. In this situation, a fluid reservoir can be connected to a gap in the edge of the cell and will drain into the gap, filling the volume between the plates. At atmospheric or greater pressures, a fluid is generally introduced to the cell by means of injection through the cell wall. Often multiple fluids are injected into a cell to allow an experimenter to observe the physical or chemical interactions they display [15].

The Hele-Shaw cell offers two primary forms of assistance to experimental researchers. The first is visibility. The indicators of many quantifiable dynamic processes are evident primarily within the interior of a region of reacting fluid, preventing easy observation. By minimizing the volume of such a region, the processes take place closer to the visible surface, allowing access to the information

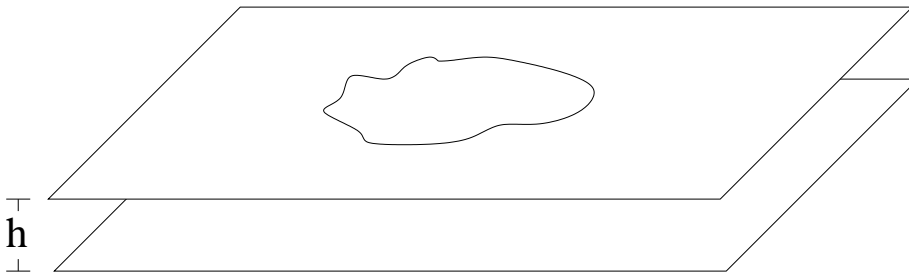


Figure 2.1: A drop within a Hele-Shaw cell

contained within.

The second general function of the Hele-Shaw cell is to create an environment for approximating the two-dimensional behavior of fluids. Many thin films are well-modeled by the two-dimensional equations of fluid mechanics, as are many fluid bodies with two exceptionally large orthogonal length scales¹. When the planar interactions of fluids in a Hele-Shaw cell take place on a significantly larger scale than the gap spacing, h , the effect from motion perpendicular to the cell becomes negligible in effect, and the material properties can often be averaged over the perpendicular (denoted as \hat{z}) direction.

2.2 Potential Flow

In the specific case of an incompressible fluid with no outside rotational forcing, we can find an equation describing \mathbf{z} -averaged velocity in a Hele Shaw cell. For a collection of fluid “particles”² with no sinks or sources, the continuity equation requires that

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{v}) \quad (2.1)$$

¹The Earth’s oceans (although up to 11,000 m deep) are perhaps the most commonly modeled case of two-dimensional potential flow in all of mathematics!

²Although we will treat the fluid as a continuous medium, we wish to account for all infinitesimal regions within some larger collection.

If the fluid is incompressible, ρ is spatially invariant and can be removed from the divergence term. Conserving mass then sets $\frac{\partial \rho}{\partial t}$ to zero, and so

$$\nabla \cdot \mathbf{v} = -\frac{1}{\rho} \frac{\partial \rho}{\partial t} = 0 \quad (2.2)$$

The restriction of Equation 2.2 onto Cauchy's equation of motion yields the prevalent Navier-Stokes Equations [1]:

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p + \rho \mathbf{g} + \mu \nabla^2 \mathbf{v} \quad (2.3)$$

$$\nabla \cdot \mathbf{v} = 0 \quad (2.4)$$

By averaging over one dimension with fixed boundary conditions, neglecting inertia, and requiring that $h \ll \sqrt{Area}$, we obtain Darcy's law³:

$$\nabla p = -\frac{12\mu}{h^2} \mathbf{v}, \quad (2.5)$$

where μ is the fluid's viscosity and h is again the plate spacing of the Hele-Shaw cell. The constant $\frac{h^2}{12}$ is known as the *permeability* of the Hele-Shaw cell⁴ [19]. Combining the continuity result, Equation 2.2, with Equation 2.5 shows that the pressure in a region of the Hele-Shaw cell must obey Laplace's equation,

$$\nabla^2 p = -\frac{\mu}{k} \nabla \cdot \mathbf{v} = 0, \quad (2.6)$$

and so the fluid behaves according to the properties of potential flow.

2.3 Flow in Bounded Regions

We have not yet discussed the idea of boundaries for our fluid. In fact, since the previous result is derived without any reference to the form of the region, it holds

³Darcy's law is often referred to as the *Hele-Shaw equation*.

⁴The permeability derives its name from the study of porous media and is equal to the permeability of a two-dimensional porous lattice with the same bulk properties as the Hele-Shaw cell.

true for all points in any continuous fluid in the Hele-Shaw cell. This means that for a fluid with defined boundary curve enveloped by a second fluid (with potentially different properties), each will evolve according to potential flow. This provides the necessary framework in which to analyze the behavior of a single drop in the Hele-Shaw cell.

In the study of two-dimensional fluid evolution and coalescence, the drop property of greatest qualitative interest is the shape of the drop boundary. Because of this, our approach will focus on the boundary elements of drops and their local properties. As we explore these properties, we will try to maintain a standard notation. The plane of the Hele-Shaw cell will in all cases be the (x, y) Cartesian plane or the (r, θ) polar plane with normal vector $\hat{\mathbf{z}}$. Unit normal vectors, $\hat{\mathbf{n}}$, of the boundary contour in the plane will be outward-pointing, and the normal velocity, \mathcal{V} in the direction of $\hat{\mathbf{n}}$, will have magnitude

$$\|\mathcal{V}\| = \hat{\mathbf{n}} \cdot \mathbf{v} \equiv -\frac{h^2}{12\mu} \frac{\partial p}{\partial \hat{\mathbf{n}}}. \quad (2.7)$$

Chapter 3

Curvature

We are interested in properties of the surface of the drop, and so it will be useful to define certain geometric values associated with curves and surfaces.

3.1 Curvature in the Plane of the Cell

For a contour in \mathbb{R}^3 , a *Frenet frame* consists of the following three vector quantities:

$$\hat{\mathbf{t}}(\mathbf{r}) = \frac{\mathbf{r}'}{\|\mathbf{r}'\|} \quad (3.1)$$

$$\hat{\mathbf{n}}(\mathbf{r}) = \frac{\hat{\mathbf{t}}'}{\|\hat{\mathbf{t}}'\|} = \frac{\mathbf{r}''}{\|\mathbf{r}''\|} \quad (3.2)$$

$$\hat{\mathbf{b}}(\mathbf{r}) = \hat{\mathbf{t}} \times \hat{\mathbf{n}} \quad (3.3)$$

These unit vectors are called the *tangent*, *normal*, and *binormal* vectors respectively for the position vector, \mathbf{r} , along the boundary curve. Since we are studying the properties of a planar curve, notice that \mathbf{b} never changes (except in sign) and points always in the direction perpendicular to the plane of the Hele-Shaw cell.

It will sometimes be convenient to deal with Cartesian coordinates, while at other times cylindrical polar coordinates may prove more useful. In either of these systems, we will define the positive $\hat{\mathbf{z}}$ -direction as the upward or out-of-the-page direction perpendicular to the plane of the plates of the cell.

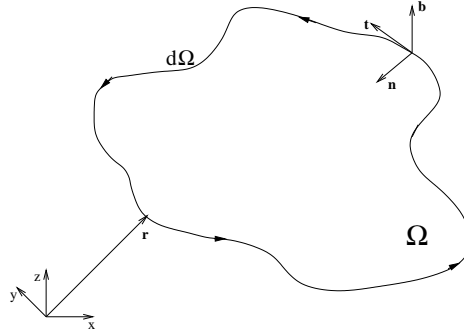


Figure 3.1: The vectors of the Frenet frame at a point along a contour

3.1.1 Parameterization by Arclength

For our tangent vector to be consistent, we need to select a direction along the curve $d\Omega$ to parameterize by. We choose the counter-clockwise direction so that $\hat{\mathbf{t}} \times \hat{\mathbf{z}}$ points along the outwards normal vector. Now, since we wish to confine ourselves to $d\Omega$, we can parameterize by arclength, s , so that s is the distance along the curve counter-clockwise from some arbitrary $s_0 = 0$, and $\mathbf{r} = \mathbf{r}(s)$. If our curve has length \mathcal{L} , then $\mathbf{r}(s) = \mathbf{r}(s + \mathcal{L})$ and $\oint_{d\Omega} ds = \mathcal{L}$.

We can also write $\mathbf{r}(s) = x(s)\mathbf{i} + y(s)\mathbf{j}$ and $\mathbf{r}'(s) = x'(s)\mathbf{i} + y'(s)\mathbf{j} = \hat{\mathbf{t}}(s)$. Since we defined the normal vector as $\hat{\mathbf{n}}(s) = \frac{\hat{\mathbf{t}}'(s)}{\|\hat{\mathbf{t}}'(s)\|}$, we may be interested in the quantity $\|\hat{\mathbf{t}}'(s)\|$, called the *curvature* of $d\Omega$. We will denote it as $\mathcal{C}_r(s)$. We can calculate the value of the curvature from the definitions of our Frenet frame vectors.

$$x''(s)\mathbf{i} + y''(s)\mathbf{j} = \hat{\mathbf{t}}'(s) = \mathcal{C}_r(s)\hat{\mathbf{n}}(s) \quad (3.4)$$

$$= \mathcal{C}_r(s) [\hat{\mathbf{b}}(s) \times \hat{\mathbf{t}}(s)] \quad (3.5)$$

$$= \mathcal{C}_r(s) [x'\mathbf{i} - y'\mathbf{j}] \quad (3.6)$$

Since $\hat{\mathbf{n}}$ is a unit vector, taking the dot product with $\hat{\mathbf{n}}$ of both sides yields

$$\mathcal{C}_r(s) [x'\mathbf{i} - y'\mathbf{j}] \cdot [x'\mathbf{i} - y'\mathbf{j}] = [x''\mathbf{i} + y''\mathbf{j}] \cdot [x'\mathbf{i} - y'\mathbf{j}] \quad (3.7)$$



Figure 3.2: The profile of a drop between the plates of a Hele-Shaw cell

$$\boxed{\mathcal{C}_r(s) = x'y'' - x''y'} \quad (3.8)$$

3.2 Curvature in the z-direction

Although we approximate the boundary in the \mathbf{x} - \mathbf{y} plane as cylindrical about the \mathbf{z} -axis, there is in fact a nonzero curvature between the plates of the Hele-Shaw cell, shown in Figure 3.2.

If the capillary number, $C_a = \frac{\mu V}{\sigma}$ is small, then surface tension properties dominate, and so the boundary will minimize its surface area by forming circular curves. The radius of curvature, \mathcal{R} , of a contour is the inverse of its curvature, \mathcal{C} . Thus, by solving for the radius of the circular arc in Figure 3.3 we can find the *axial* curvature, \mathcal{C}_z (as opposed to the *radial* curvature, \mathcal{C}_r , defined previously).

The contact angle, α , in Figure 3.3 is the angle formed between the solid and fluid interfaces. If we draw a line from the center of the circle to the point of contact, the angle formed, β will just be $\alpha - \frac{\pi}{2}$.

$$\mathcal{R} = \frac{h}{2 \sin \beta} = \frac{h}{2 \sin \left(\alpha - \frac{\pi}{2} \right)} = -\frac{h}{2 \cos \alpha}, \quad (3.9)$$

so the curvature is just

$$\boxed{\mathcal{C}_z = -\frac{2 \cos \alpha}{h}} \quad (3.10)$$

If $\mathcal{C}_z < 0$, then $\alpha < \frac{\pi}{2}$. This is known as a *wetting* condition. Likewise, if $\mathcal{C}_z > 0$, the drop is not wetting.

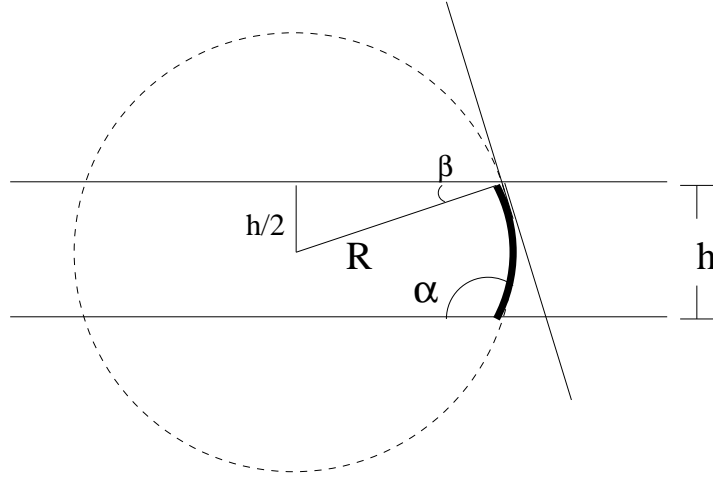


Figure 3.3: Radius of curvature of a drop in the plane perpendicular to the cell

The total curvature of the drop is just $\mathcal{C} = \mathcal{C}_r + \mathcal{C}_z$, so from Equations 3.8 and 3.10,

$$\boxed{\mathcal{C} = x'y'' - x''y' - \frac{2 \cos \alpha}{h}} \quad (3.11)$$

3.3 Curvature-Driven Motion

The curvature plays an important role in determining the time-evolution of a drop. \mathcal{C} is directly proportional to the pressure difference across the drop's boundary, and the ratio $\frac{\Delta p}{\mathcal{C}}$ is the surface tension, σ , of the boundary interface. Thus, in regions of high curvature, the surface tension-induced forces on the drop are particularly high relative to the forces in regions of low curvature. This leads to potential problems in modeling drop evolution near the point of coalescence, since not only does \mathbf{t} become singular at the cusp, but the surface tension force is locally infinite.

For drops with *constant* curvature in both dimensions (planar circles with constant contact angle), we expect constant surface tension forces, and thus radially-symmetric motion. For incompressible fluids, however, the area of a drop must

remain constant, and so the boundary curve should remain static.

3.3.1 *Electrowetting*

The contact angle for a fluid/fluid/solid interface is an empirically-determinable constant under equilibrium conditions¹. However, since surface tension is very dependent upon induced electric polarity, a voltage across the medium can significantly change the contact angle [6]. This can be easily taken advantage of to drive the motion of drop by applying differing potentials across different regions of the Hele-Shaw cell in a process called *electrowetting*. Electrowetting is becoming a common practice in the field of micro-fluidics and is easily implemented in a model of drop behavior by simply allowing variation in the contact angle.

¹The contact angle for the water-air boundary on a Teflon sheet is 110° , (non-wetting for water) due to the relatively high surface tension, $\sigma = 7.2 \times 10^{-2} \frac{\text{N}}{\text{m}}$ [11].

Chapter 4

Formulating an Integral Equation

In this chapter we create integral equations for the normal velocity of a drop in terms of pressure and the Green's function for Laplace's equation. In Section 4.2 we look at the case where the pressure outside the drop is a constant. We extend this in Section 4.3 to potential flow outside the drop as well.

4.1 Preliminaries

We begin with Green's second identity for two differentiable functions, ϕ and ψ , in a region Ω representing our drop [see Appendix A for a more thorough discussion]:

$$\oint_{d\Omega} \left(\psi \frac{\partial \phi}{\partial \hat{\mathbf{n}}} - \phi \frac{\partial \psi}{\partial \hat{\mathbf{n}}} \right) d\ell = \iint_{\Omega} (\psi \nabla^2 \phi - \phi \nabla^2 \psi) dA. \quad (4.1)$$

This will prove very useful in determining the motion of boundary elements, given the proper functions ϕ and ψ . Since the impetus for the drop's motion is the pressure difference across the boundary, it is reasonable to think that one of these functions may be $p_{in} - p_{out}$, where p_{in} is the pressure inside the drop and p_{out} is the constant pressure outside the drop.

4.2 Equation for constant external pressure

Since we hope that Equation 4.1 will be strictly in terms of this pressure difference, one might recognize the benefit of defining the other function as the Green's function for the pressure [see Appendix B]. The pressure satisfies Laplace's equation,

and so the corresponding Green's function is

$$G(\mathbf{r}, \mathbf{r}_0) = \frac{1}{2\pi} \ln \|\mathbf{r} - \mathbf{r}_0\|, \quad (4.2)$$

$$\text{with } \nabla^2 G(\mathbf{r}, \mathbf{r}_0) = \delta(\mathbf{r}, \mathbf{r}_0) = \delta(x, x_0) \delta(y, y_0). \quad (4.3)$$

In Cartesian coordinates,

$$\boxed{G = \frac{1}{2\pi} \ln [(x - x_0)^2 + (y - y_0)^2]} \quad (4.4)$$

$$\boxed{\frac{\partial G}{\partial \hat{\mathbf{n}}} = \hat{\mathbf{n}} \cdot \nabla G = \frac{x'(y - y_0) - y'(x - x_0)}{2\pi [(x - x_0)^2 + (y - y_0)^2]}} \quad (4.5)$$

$$\text{where } x = x(s), \quad x_0 = x(s_0), \quad y = y(s), \quad y_0 = y(s_0)$$

If we now set $\phi = p_{in} - p_{out} \equiv p(\mathbf{r})$ and $\psi = G(\mathbf{r}, \mathbf{r}_0)$, we notice that Equation 4.1 becomes

$$\oint_{d\Omega} \left[G(\mathbf{r}, \mathbf{r}_0) \frac{\partial p}{\partial \hat{\mathbf{n}}} - p \frac{\partial G}{\partial \hat{\mathbf{n}}} \right] d\ell = \iint_{\Omega} [G(\mathbf{r}, \mathbf{r}_0) \nabla^2 p - p \nabla^2 G(\mathbf{r}, \mathbf{r}_0)] dA \quad (4.6)$$

$$= 0 - \iint_{\Omega} p \delta(\mathbf{r} - \mathbf{r}_0) dA \quad (4.7)$$

$$= \frac{1}{2} p(\mathbf{r}_0), \quad (4.8)$$

for our point \mathbf{r}_0 on the contour $d\Omega$. If we parameterize by arclength as in Section 3.1.1, Equation 4.8 becomes

$$\frac{1}{2} p(s_0) = \int_0^{\mathcal{L}} \left[p(s) \frac{\partial G}{\partial \hat{\mathbf{n}}} - G(s, s_0) \frac{\partial p}{\partial \hat{\mathbf{n}}} \right] ds. \quad (4.9)$$

We recall from Section 3.3 that the difference in pressure across the boundary of the drop is $p(s) = p_{in}(s) - p_{out} = \sigma \mathcal{C}(s)$. Making use of our notation for the normal velocity, $\mathcal{V}(s) = -\frac{h^2}{12\mu} \frac{\partial p}{\partial \hat{\mathbf{n}}}$, we find

$$\frac{\sigma}{2} \mathcal{C}(s_0) = \int_0^{\mathcal{L}} \left[\sigma \mathcal{C}(s) \frac{\partial G}{\partial \hat{\mathbf{n}}} + \frac{12\mu}{h^2} G(s, s_0) \mathcal{V}(s) \right] ds, \quad (4.10)$$

which can be rearranged into

$$\boxed{\frac{12\mu}{h^2 \sigma} \int_0^{\mathcal{L}} G(s, s_0) \mathcal{V}(s) ds = \frac{1}{2} \mathcal{C}(s_0) - \int_0^{\mathcal{L}} \mathcal{C}(s) \frac{\partial G}{\partial \hat{\mathbf{n}}} ds.} \quad (4.11)$$

This is an integral equation of the first kind for $\mathcal{V}(s)$.

4.3 Equations for non-constant pressure

In a similar fashion, if the pressure outside the drop is not a constant, but rather a function satisfying Laplace's Equation, we can formulate an integral equation for both the interior and the exterior of the drop. The two equations are coupled through the normal velocity and the pressure at the boundary. If the barred ($\bar{}$) demarcation corresponds to properties inside the drop, then

$$\frac{1}{2}\bar{P}(s_0) = \int_0^{\mathcal{L}} \left[G(s, s_0) \frac{12\bar{\mu}}{h^2} \mathcal{V}(s) + \bar{P}(s) \frac{\partial G}{\partial \hat{\mathbf{n}}} \right] ds \quad (4.12)$$

$$\text{and} \quad \frac{1}{2}P(s_0) = -\int_0^{\mathcal{L}} \left[G(s, s_0) \frac{12\mu}{h^2} \mathcal{V}(s) + P(s) \frac{\partial G}{\partial \hat{\mathbf{n}}} \right] ds \quad (4.13)$$

By adding these equations together and subtracting the second from the first we get a new pair of integral equations,

$$\frac{1}{2}S(s_0) = \int_0^{\mathcal{L}} \left[\frac{12(\bar{\mu} - \mu)}{h^2} G(s, s_0) \mathcal{V}(s) + \sigma \mathcal{C}(s) \frac{\partial G}{\partial \hat{\mathbf{n}}} \right] ds \quad (4.14)$$

$$\frac{\sigma}{2}\mathcal{C}(s_0) = \int_0^{\mathcal{L}} \left[\frac{12(\bar{\mu} + \mu)}{h^2} G(s, s_0) \mathcal{V}(s) + S(s) \frac{\partial G}{\partial \hat{\mathbf{n}}} \right] ds, \quad (4.15)$$

where $\sigma \mathcal{C}(s)$ has been substituted for $\bar{P}(s) - P(s)$, and $S(s)$ has been substituted for $\bar{P}(s) + P(s)$. The simultaneous solution of Equations 4.14 and 4.15 allows us to solve for the drop boundary motion for an incompressible two-fluid system.

Chapter 5

Discretization

Both of the integrals in Equation 4.11 become weakly singular at the point $s = s_0$. To avoid infinite-valued intervals, we can regularize the two terms and remove singular points. Upon regularization [see Appendix C], Equation 4.11 becomes

$$\frac{\mathcal{L}}{2\pi} \ln \left(\frac{\mathcal{L}}{2\pi} \right) \mathcal{V}(s_0) + \int_0^{\mathcal{L}} [(G(s, s_0) - F(s, s_0)) \mathcal{V}(s)] + [F(s, s_0) (\mathcal{V}(s) - \mathcal{V}(s_0))] + \frac{\sigma h^2}{12\mu} \left[\frac{\partial G}{\partial \hat{\mathbf{n}}} (\mathcal{C}(s) - \mathcal{C}(s_0)) \right] ds = 0, \quad (C.8)$$

where the function $F(s, s_0)$ is the Green's function for a circular region of circumference \mathcal{L} .

To discretize this equation in its simplest form, we view the integral as a sum of trapezoidal areas. If the boundary of the drop is broken into N segments by N nodes spaced evenly by arclength, we get a "summation equation":

$$\sum_{m=1}^N \left[\lambda \mathcal{V}_n + (G_{m,n} - F_{m,n}) \mathcal{V}_m + F_{m,n} (\mathcal{V}_m - \mathcal{V}_n) + \frac{\sigma h^2}{12\mu} \frac{\partial G}{\partial \hat{\mathbf{n}}}{}_{m,n} (\mathcal{C}_m - \mathcal{C}_n) \right] = 0 \quad (5.1)$$

where the subscript notation represents nodal values, the constant λ is equal to $\frac{1}{2\pi} \ln \left(\frac{\mathcal{L}}{2\pi} \right)$, and the arbitrary source point, s_0 , is now denoted by the node n .

Since the singularities at $m = n$ are removed in the regularized equation, we can make use of new functions, $\tilde{G}_{m,n}$, $\tilde{F}_{m,n}$, and $\frac{\partial \tilde{G}}{\partial \hat{\mathbf{n}}}{}_{m,n}$ that are identical to $G_{m,n}$, $F_{m,n}$, and $\frac{\partial G}{\partial \hat{\mathbf{n}}}{}_{m,n}$, respectively, except that they are equal to zero when $m = n$.

$$\sum_{m=1}^N \left[\tilde{G}_{m,n} \mathcal{V}_m + \left(\lambda - \tilde{F}_{m,n} \right) \mathcal{V}_n + \frac{\sigma h^2}{12\mu} \frac{\partial \tilde{G}}{\partial \hat{\mathbf{n}}}{}_{m,n} (\mathcal{C}_m - \mathcal{C}_n) \right] = 0 \quad (5.2)$$

To solve for the normal velocity at any node, we want to collect all \mathcal{V} terms under a single index. \mathcal{V}_n can be rewritten as

$$\mathcal{V}_n = \sum_{i=1}^N \delta_{ni} \mathcal{V}_i, \quad (5.3)$$

where δ is Kronecker's delta, so that

$$\sum_{m=1}^N \left[(\lambda - \tilde{F}_{m,n}) \mathcal{V}_n \right] = \sum_{m=1}^N \left[(\lambda - \tilde{F}_{m,n}) \left(\sum_{i=1}^N \delta_{ni} \mathcal{V}_i \right) \right] \quad (5.4)$$

$$= \sum_{i=1}^N \left[\delta_{ni} \left(\lambda - \sum_{m=1}^N \tilde{F}_{m,n} \right) \mathcal{V}_i \right] \quad (5.5)$$

A similar treatment of the \mathcal{C}_n term allows us to write

$$\sum_{m=1}^N \left[\frac{\partial \tilde{G}}{\partial \hat{\mathbf{n}}}{}^{m,n} \mathcal{C}_n \right] = \sum_{i=1}^N \left[\delta_{ni} \left(\sum_{m=1}^N \frac{\partial \tilde{G}}{\partial \hat{\mathbf{n}}}{}^{m,n} \right) \mathcal{C}_i \right] \quad (5.6)$$

In Equations 5.5 and 5.6, m is now clearly a dummy index, as it is in the remaining two terms of 5.2, $\tilde{G} \cdot \mathcal{V}_m$ and $\frac{\partial \tilde{G}}{\partial \hat{\mathbf{n}}} \cdot \mathcal{C}_m$. Summing over m makes both \tilde{F} and $\frac{\partial \tilde{G}}{\partial \hat{\mathbf{n}}}$ dependent only on n , so we denote

$$\Lambda_n = \sum_{m=1}^N \tilde{F}_{m,n} \quad \text{and} \quad \Gamma_n = \sum_{m=1}^N \frac{\partial \tilde{G}}{\partial \hat{\mathbf{n}}}{}^{m,n}$$

We can use Einstein summation notation (summing over repeated indices) to rephrase this in a suggestive manner. We change the remaining m indices to i 's and the dummy n 's to j 's and combine Equations 5.2, 5.5, and 5.6 to find

$$\tilde{G}_{ji} \mathcal{V}_i + \delta_{ji} (\lambda - \Lambda_j) \mathcal{V}_i + \frac{\sigma h^2}{12\mu} \left[\frac{\partial \tilde{G}}{\partial \hat{\mathbf{n}}}{}^{ji} \mathcal{C}_i - \delta_{ji} \Gamma_j \mathcal{C}_i \right] = 0, \quad (5.7)$$

or
$$\boxed{\left[\tilde{G}_{ji} + \delta_{ji} (\lambda - \Lambda_j) \right] \mathcal{V}_i = \frac{\sigma h^2}{12\mu} \left[\delta_{ji} \Gamma_j - \frac{\partial \tilde{G}}{\partial \hat{\mathbf{n}}}{}^{ji} \right] \mathcal{C}_i} \quad (5.8)$$

Since Equation 5.2 should hold true for any arrangement of equally-spaced nodes, the terms of the summation must *each* be identically zero for every node, $i \in [1, N]$. Furthermore, since node j is arbitrary, each of the N nodal equations must hold independently, and thus we have a linear matrix equation relating the nodal velocities and curvatures.

Chapter 6

Simulated Results

This chapter discusses the implementation of a routine to model a drop's evolution in time using the governing equations of Chapter 5. Section 6.1 discusses the basic structure of the program used, and Section 6.2 presents the numerical results of the simulation.

6.1 Computational Model

A numerical routine written in Octave/Matlab was used to model a drop's evolution in time [see Appendix D]. Under the routine "evolution", a drop was initialized as a set of (x, y) coordinate pairs. The points were then redistributed equally by arclength, both to better model a continuous boundary curve and to meet the restrictions required in Chapter 5. The redistribution process simulated the physical drop boundary through a twice-differentiable cubic spline with periodic boundary conditions (1st and 2nd derivative matching).

During each iteration of the drop's evolution, matrix equation 5.8 was solved for the normal velocity, \mathcal{V}_i , the nodes were advanced by a distance $\mathcal{V}_i \cdot \Delta t$ in the normal direction, and the points were redistributed by arclength along a new spline representing the drop boundary.

The contact angle, α , could be assigned different values for different regions of the x - y plane, corresponding to voltages applied across the plane of the Hele-Shaw cell. The initial conditions of the drop could also be easily changed, including the area of the drop, the number of boundary nodes, the shape of the initial drop

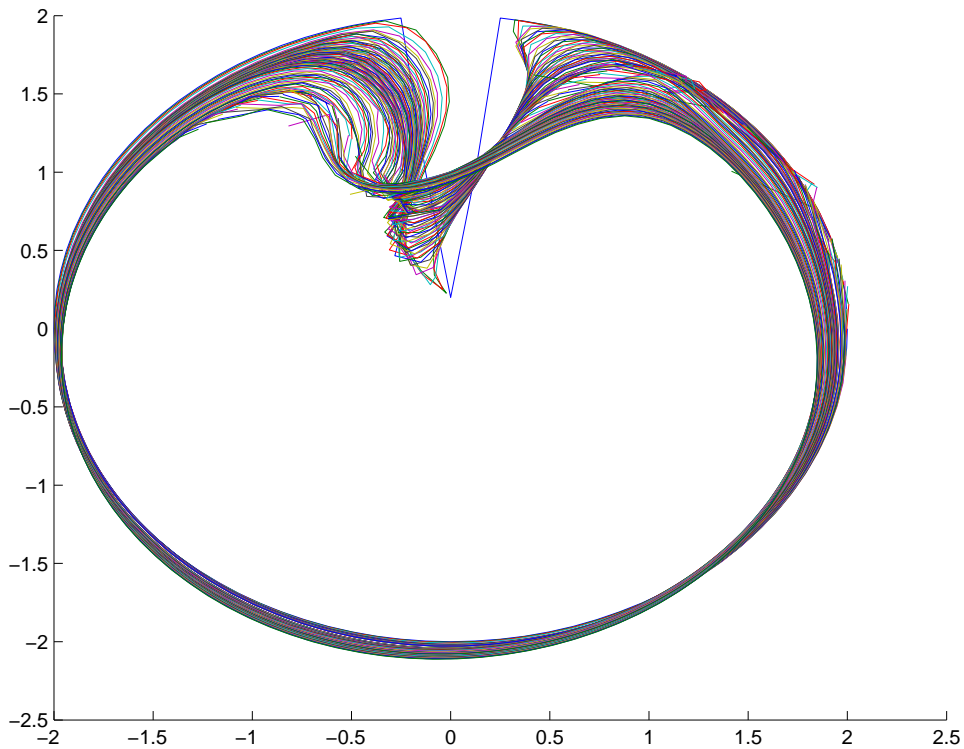


Figure 6.1: 100 timesteps of a drop's evolution from initial conditions with a near-singularity

boundary, and the placement of the drop in the plane.

6.2 Numerical Results

For suitable timestep values, a drop could easily be observed to evolve in time under the summed actions of the local curvature. Drops with constant curvature remained stable with slight changes in relative node positions due to the node redistribution process, and drops with initially elliptical boundaries would compress along the major axis and expand along the minor axis, as desired.

Figure 6.2 shows the unforced progression in time of a drop from a nearly singular initial state to an increasingly smooth contour as it returns to a circular state. The timestep was constant throughout the 100 iterations and was constructed so

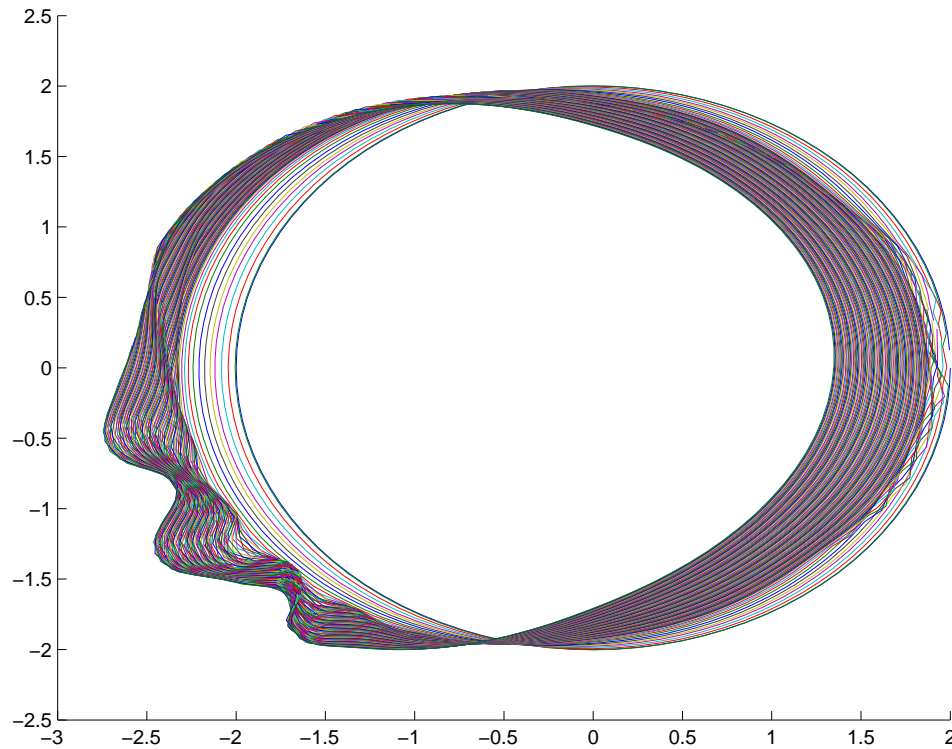


Figure 6.2: A drop subject to x^3 voltage in the $x < 0$ half-plane

that the maximum nodal perturbation on the first iteration was $\frac{1}{1000} \sqrt{\frac{A}{\pi}}$ for drop area A . This condition was commonly used so that after the first timestep, no node would have moved more than $1/1000^{\text{th}}$ the radius of the drop, were the drop a perfect circle.

Figure 6.2 shows an initially circular drop moving in the $-x$ direction under the influence of a voltage across the left Cartesian half-plane. The effective voltage, and thus the change in the contact angle, α increased with the cube of x and led to more volatile boundary behavior than linear or quadratic voltages.

Figure 6.2 shows three plots of a single drop's progression from an initial state representing a close approximation of two-drop coalescence. The initial drop is simply a circular drop with the points at $\theta = \frac{\pi}{2}$ and $\frac{3\pi}{2}$ moved very near to the

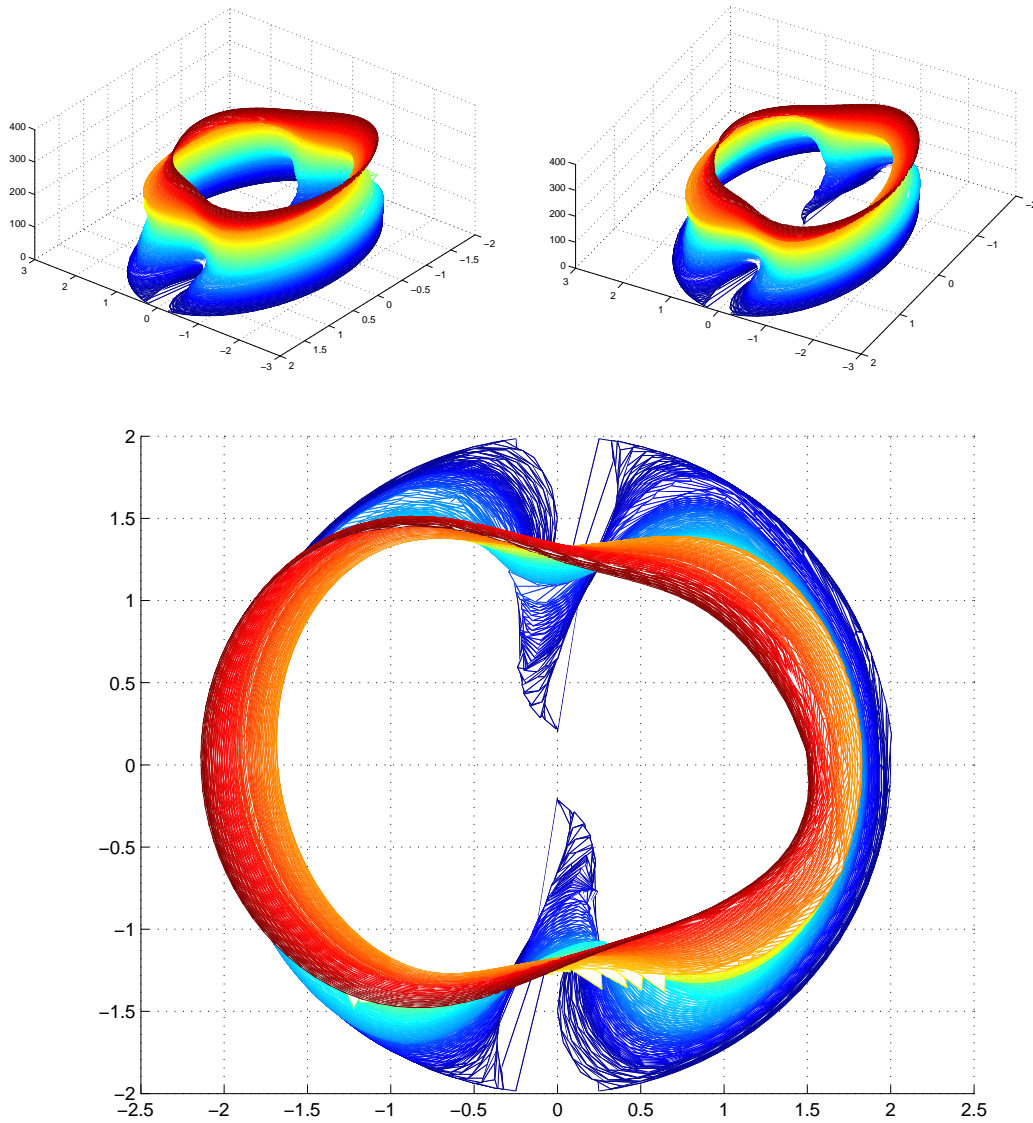


Figure 6.3: A drop evolving shortly after simulated unification of two semicircular drops (vertical axis represents time)

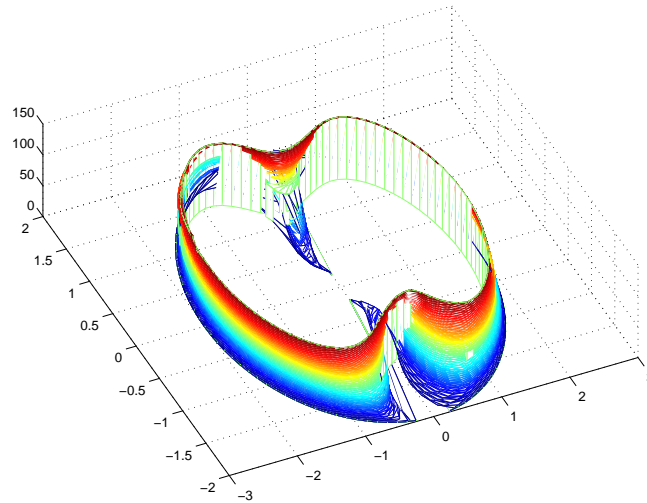


Figure 6.4: Drop evolution with variable-length time-stepping immediately following simulated drop unification (Vertical axis represents time)

drop's center, and the two adjacent points for each moved very slightly towards the center of the circle. This is a simple approximation to the boundary curve shape that would be found immediately following the thin-film rupture of two identical drops with high surface tension or low collision velocity. The vertical axes of Fig. 6.2a and Fig. 6.2b represent time or iteration number.

Figure 6.2 depicts the time-evolution of the same initial conditions as in Figure 6.2, but with a variable time step which ensures that with each iteration, one node will move exactly $1/1000^{\text{th}}$ the radius of the equivalent circular drop. This fixed-distance stepping generally had fewer associated undesirable discontinuities than the constant timestep method.

6.3 Associated Problems and Solutions

There appear to be errors in the implemented method for calculating the second derivatives of the x and y coordinates from the boundary spline. Even for constant

curvature circular drops, x'' and y'' fluctuate in value for nodes near the endpoints. A second routine¹ for calculating periodic splines and derivatives taken from the octave-forge distribution yields similar results.

The irregularity caused by the inaccuracies in knowledge of the second derivatives leads to occasional problems in the simulated behavior of the drop. The ripples which are visible in Figure 6.2 in the third quadrant are the direct product of boundary oscillations near nodes 1 and n . For large timestep values or particularly dynamic drop conditions, nodes sometimes appear to cross paths, leading to negative or complex values for the area inside the boundary. When this happens, the tridiagonal matrix solver which calculates the second derivatives generally fails, but, occasionally, the nodes are instead propelled dramatically away from their original position, leading to significant changes in the drop's apparent behavior.

The most prevalent problem in the routine, however, is the loss of area of a drop. This seems to stem primarily from the same issue regarding second derivatives. The ideal solution to this problem would be to scale the nodal distances from the centroid of the drop during each timestep. Unfortunately, since most calculations of the centroid depend on either first or second derivative values, the centroid tends to magnify the errors in the simulation rather than dampen them.

The most beneficial solution to any current errors in "evolution" seems to be the constant-distance timestep. Although program errors are currently the source of most of the inaccuracy in the model, the precision errors and discretization problems which are the ultimate hurdle for numerical simulations manifest themselves in regions of high curvature and large velocity as well. By maintaining the same maximum normal velocity value at each step, we are more likely to overcome these obstacles as they arise.

¹csape.m,©Kai Habel,2001

Chapter 7

Summary and Conclusions

Apart from the numeric implementation, the aspects of this research which presently need the most development are the pre-coalescence processes. Modeling the boundary evolution of a drop from initial conditions seems to be quite feasible on its own. Determining exactly how the coalescence process is begun is far more challenging. At present, this model is not far from being able to properly simulate the physical interactions between two non-rupturing drops upon collision. Researching thin film rupture or other surface interactions would likely prove useful in more properly modeling the entire coalescence process.

To further explore the post-coalescence behavior of drops, an investigation into curvature-dependent node distributions might be beneficial. Since the normal velocity is lower in areas of low curvature, fewer nodes are needed to maintain the same accuracy as compared to areas of high curvature. Also, since electrowetting forces are bounded and can be easily dwarfed by the planar curvature forces, a model does not have to trade accuracy in \mathcal{C}_z -driven behavior directly for its gains in \mathcal{C}_r -driven behavior.

The other interesting off-shoot of this project is, of course, electrowetting research. Very few of the simulated test cases thus far have been purely electrically driven since the curvature-based effects often dominate the forcing. By altering the larger length-scales of the drops under investigation, the relative influence of an applied voltage on the fluid will change.

Many of the experimental projects currently underway in two-dimensional electrowetting deal with mixing and other “internal” aspects of a drop of fluid which



Figure 7.1: Electrowetting research from Nanolytics Inc. [13] (© 2001, Nanolytics)

are difficult to probe using only boundary elements. The nanolaboratory research currently in progress at Nanolytics, Inc., for example, seeks to replace micropipette-based chemical assays with microchip-based mixing procedures. The relationship between an electrically-driven drop and its inner mixing behavior will likely become applicably relevant in the near future as nanolaboratories become more logistically viable.

One final area with a lot of room for both theoretical and applied research in electrowetting is control theory. Creating either physical or mathematical fluid feedback systems would be relatively simple and could easily have an important impact on developing microtechnology. For more information on developing technologies in electrowetting, see [13, 8, 22].

Appendix A

Potential Functions and Green's Second Identity

Consider two twice-differentiable functions, ϕ and ψ , in a region Ω bounded by a contour $d\Omega$. The chain rule allows us to write

$$\nabla \cdot (\psi \nabla \phi) = \nabla \psi \cdot \nabla \phi + \psi \nabla^2 \phi \quad (\text{A.1})$$

$$\nabla \cdot (\phi \nabla \psi) = \nabla \phi \cdot \nabla \psi + \phi \nabla^2 \psi \quad (\text{A.2})$$

Subtraction of Equation A.2 from Equation A.1 yields

$$\nabla \cdot [\psi \nabla \phi - \phi \nabla \psi] = \psi \nabla^2 \phi - \phi \nabla^2 \psi \quad (\text{A.3})$$

By taking advantage of this formulation and applying the divergence (Gauss') theorem, we can create an integral equation for ϕ and ψ [18]:

$$\oint_{d\Omega} \hat{\mathbf{n}} \cdot (\psi \nabla \phi - \phi \nabla \psi) d\ell = \iint_{\Omega} (\psi \nabla^2 \phi - \phi \nabla^2 \psi) dA, \quad (\text{A.4})$$

Using our earlier notation for the directional derivative, $\frac{\partial x}{\partial \hat{\mathbf{n}}} \equiv \hat{\mathbf{n}} \cdot \nabla x$,

$$\boxed{\oint_{d\Omega} \left(\psi \frac{\partial \phi}{\partial \hat{\mathbf{n}}} - \phi \frac{\partial \psi}{\partial \hat{\mathbf{n}}} \right) d\ell = \iint_{\Omega} (\psi \nabla^2 \phi - \phi \nabla^2 \psi) dA} \quad (\text{A.5})$$

Equation A.5 is commonly known as *Green's second identity*. Applying the divergence theorem to Equation A.1 or A.2 directly yields the *first* identity which is often used as an intermediate to the second¹.

¹For an alternate derivation of Green's second identity, see Power & Wrobel

Appendix B

Formulation of Green's Function

We will define the time-independent Green's function, $G(s, s_0)$, as it is most often thought of – as the “response at s due to a concentrated source” at s_0 , subject to boundary conditions a and b [7].

$$\mathcal{L}[G(s, s_0)] = \delta(s - s_0) \quad (\text{B.1})$$

Now, consider a general linear partial differential equation with an arbitrary linear differential operator, \mathcal{L} :

$$\mathcal{L}(u) = g(s), \quad (\text{B.2})$$

The general form of \mathcal{L} of order p in n variables is [20]

$$\mathcal{L} = \sum_{\{k\} \leq p} f_k(x) D^k = \sum_{\{k\} \leq p} f_k(x) \frac{\partial^{k_1 + \dots + k_n}}{\partial x_1^{k_1} \dots \partial x_n^{k_n}}, \quad (\text{B.3})$$

where the $f_k(x)$ are arbitrary k -valued functions of n variables. The traditional Green's formula,

$$\int_{\Omega} (v \nabla^2 u - u \nabla^2 v) dx = \int_{\Gamma} \hat{\mathbf{n}} \cdot (v \nabla u - u \nabla v) dS, \quad (\text{B.4})$$

generalizes for arbitrary \mathcal{L} to the *full* Green's formula [20]:

$$\int_{\Omega} (v \mathcal{L} u - u \mathcal{L}^* v) dx = \int_{\Gamma} \hat{\mathbf{n}} \cdot \mathcal{J}(u, v) dS. \quad (\text{B.5})$$

With one-dimensional boundary conditions,

$$\int_a^b (v \mathcal{L} u - u \mathcal{L}^* v) ds = k \left(v \frac{du}{dv} - u \frac{dv}{du} \right) \Bigg|_a^b, \quad (\text{B.6})$$

If we set $v = G(s, s_0)$, then

$$\left(G(s, s_0) \frac{du}{dG(s, s_0)} - u \frac{dG(s, s_0)}{du} \right) \Big|_a^b = 0 \quad (\text{B.7})$$

since both $G(s, s_0)$ and u satisfy the same boundary conditions. Thus,

$$0 = \int_a^b (G(s, s_0)g(s) - u(s)\delta(s - s_0)) ds = \int_a^b G(s, s_0)g(s)ds - u(s_0). \quad (\text{B.8})$$

Exchanging s and s_0 yields

$$\boxed{u(s) = \int_a^b G(s_0, s)g(s_0)ds_0,} \quad (\text{B.9})$$

the solution to equation B.2. This result is what leads us to be able to reformulate a partial differential equation (Eq. 2.5) into a more manageable boundary integral equation (Eq. 4.11).

B.0.1 Symmetry of the Green's Function

¹ Making use of the one-dimensional Green's formula again, Equation B.6, this time letting $u = G(s, s_1)$ and $v = G(s, s_2)$, the right hand side again vanishes since both $G(s, s_1)$ and $G(s, s_2)$ satisfy the same homogeneous boundary conditions.

$$0 = \int_a^b (G(s, s_1)\mathcal{L}G(s, s_2) - G(s, s_2)\mathcal{L}^*G(s, s_1)) ds \quad (\text{B.10})$$

$$= \int_a^b (G(s, s_1)\delta(s - s_2) - G(s, s_2)\delta(s - s_1)) ds \quad (\text{B.11})$$

$$= G(s_2, s_1) - G(s_1, s_2), \quad (\text{B.12})$$

and so the Green's function is symmetric under exchange of its arguments.

$$\boxed{G(s_1, s_2) = G(s_2, s_1)} \quad (\text{B.13})$$

¹This argument follows nearly directly from the discussion of Maxwell's reciprocity in Haberman [7].

This, of course, holds relevance for our interpretation of $G(s, s_0)$. By our own definition, we find that the response at s “due to a concentrated source” at s_0 *is the same* as the response at s_0 “due to a concentrated source” at s ! This symmetry proves problematic, however, as the source of the singularity in both $G(s, s_0)$ and our function $\frac{\partial G}{\partial \mathbf{\hat{n}}}$.

Appendix C

Integral Equation Regularization

The functions $G(s, s_0)$ and $\frac{\partial G}{\partial \mathbf{n}}$ are both singular at the point $s = s_0$ in Equation 4.11 below.

$$\frac{12\mu}{h^2\sigma} \int_0^{\mathcal{L}} G(s, s_0) \mathcal{V}(s) ds = \frac{1}{2} \mathcal{C}(s_0) - \int_0^{\mathcal{L}} \mathcal{C}(s) \frac{\partial G}{\partial \mathbf{n}} ds. \quad (\text{C.11})$$

To evaluate the integrals, we first need to remove any singularities from the integrals themselves. For the first integral,

$$\int_0^{\mathcal{L}} G(s, s_0) \mathcal{V}(s) ds = \int_0^{\mathcal{L}} [G(s, s_0) - F(s, s_0)] \mathcal{V}(s) ds + \int_0^{\mathcal{L}} F(s, s_0) \mathcal{V}(s) ds \quad (\text{C.1})$$

where $F(s, s_0)$ is any well-defined function over $[0, \mathcal{L}]$. If we choose F specifically so that it is singular in the same manner as G , then the first term of the right side of Equation C.1 will no longer diverge at s_0 .

We let F be the Green's function of a circular region of circumference \mathcal{L} :

$$F(s, s_0) = \frac{1}{2\pi} \ln |\mathbf{r}(s) - \mathbf{r}(s_0)| = \frac{1}{2\pi} \ln \left| \frac{\mathcal{L}}{\pi} \sin \left(\frac{\pi(s - s_0)}{\mathcal{L}} \right) \right|, \quad (\text{C.2})$$

$$\text{so that} \quad \int_0^{\mathcal{L}} F(s, s_0) ds = \frac{\mathcal{L}}{2\pi} \ln \left(\frac{\mathcal{L}}{2\pi} \right) \quad (\text{C.3})$$

Since as $s \rightarrow s_0$, $\sin \left(\frac{\pi(s - s_0)}{\mathcal{L}} \right) \rightarrow \frac{\pi}{\mathcal{L}}(s - s_0)$, and $|\mathbf{r}(s) - \mathbf{r}(s_0)| \rightarrow d\mathbf{r} \cdot d\mathbf{r} = ds^2$, the asymptotic behavior of both F and G is

$$\frac{1}{2\pi} \ln |s - s_0| + \frac{1}{2\pi} \ln |\mathbf{r}'(s_0)| + \mathcal{O}(s - s_0) \quad (\text{C.4})$$

and so $G(s, s_0) - F(s, s_0) \rightarrow 0$.

Of course, in removing the singularity from Equation C.1 we have simply created another singular integral, but of F rather than G . However, since F is analytically integrable, we can write

$$\int_0^{\mathcal{L}} F(s, s_0) \mathcal{V}(s) ds = \int_0^{\mathcal{L}} F(s, s_0) [\mathcal{V}(s) - \mathcal{V}(s_0)] ds + \mathcal{V}(s_0) \int_0^{\mathcal{L}} F(s, s_0) ds \quad (\text{C.5})$$

$$= \int_0^{\mathcal{L}} F(s, s_0) [\mathcal{V}(s) - \mathcal{V}(s_0)] ds + \frac{\mathcal{L}}{2\pi} \ln \left(\frac{\mathcal{L}}{2\pi} \right) \mathcal{V}(s_0) \quad (\text{C.6})$$

For the second integral of Equation 4.11, we recall that

$$\oint_{d\Omega} \frac{\partial G}{\partial \hat{\mathbf{n}}} d\ell = \frac{1}{2}$$

and so the right side of 4.11 becomes

$$\mathcal{C}(s_0) \int_0^{\mathcal{L}} \frac{\partial G}{\partial \hat{\mathbf{n}}} ds - \int_0^{\mathcal{L}} \mathcal{C}(s) \frac{\partial G}{\partial \hat{\mathbf{n}}} ds = \int_0^{\mathcal{L}} \frac{\partial G}{\partial \hat{\mathbf{n}}} [\mathcal{C}(s_0) - \mathcal{C}(s)] ds \quad (\text{C.7})$$

By substituting these results into Equation 4.11, we obtain a regularized integral equation,

$$\begin{aligned} \frac{\mathcal{L}}{2\pi} \ln \left(\frac{\mathcal{L}}{2\pi} \right) \mathcal{V}(s_0) + \int_0^{\mathcal{L}} [(G(s, s_0) - F(s, s_0)) \mathcal{V}(s)] + [F(s, s_0) (\mathcal{V}(s) - \mathcal{V}(s_0))] \\ + \frac{\sigma k}{\mu} \left[\frac{\partial G}{\partial \hat{\mathbf{n}}} (\mathcal{C}(s) - \mathcal{C}(s_0)) \right] ds = 0, \end{aligned} \quad (\text{C.8})$$

which we can discretize without creating elements with infinite values.

Appendix D

Simulation Source Files

All files were written by Daniel Gianotti and last modified between 1/29/03 and 5/6/03. The simulation was performed in Octave version 2.1.47 with installed packages octave-ci and octave-forge, and the image files were created in Matlab6 using output Octave datafiles.

evolution.m

```
% Written by Daniel Gianotti for Octave. Commenting would need
% to be changed for Matlab functionality, as would some non-user-defined
% m-files (most notably the octave-forge splines package). All are
% freely available and used under the terms of the GNU Public License.

global dT;
    dT=0;
global h_scale;                %spacing between plates
    h_scale=.01;
global contactang;
    contactang=110*pi/180;
global new_xy;
global iteration;
    iteration=0;
global N;
global dds_xy;
global Area;
global rad_scale;
global dxy;

%%%
%Some diagnostic variables...
%%%
global curvature;
```

```

global dGdn;
global input_xy;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The initialization process (pre-evolution)%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
input_xy = build_xy; % vector of (x,y) pairs
                                % asks for user input to create
                                % initial drop shape (currently ellipse)

N=length(input_xy);           % N is the number of nodes we have

%debug="xy has been built for the first time"
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The drop evolution process%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function evolve(old_xy)

    global dT;
    global h_scale;
    global contactang;
    global N;
    global input_xy;
    global iteration;
    global new_xy;
    global dds_xy;
    global Area;
    global rad_scale;
    global dxy;

    iteration++

    old_s = build_s_vector(old_xy); % s_vector is the vector of arclengths to
        % each x,y point

    [xy, s] = redistribute(old_xy, old_s); % redistribute places N nodes
        % at equal s positions along
        % cubic splines representing
#xy=old_xy;           % the perimeter of the drop. It calls the
#s=old_s;             % functions build_K, build_b, arclength, and
                    % extend_nodes

```



```

solution = build_solution(xy,dds_xy,d2ds2_xy,contactang,(rad_scale/h_scale));

matA = build_matA(xy,s);

nVel = inverse(matA)*solution;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%the new positions are just the old ones plus nVel * normal vectors%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
normal = [dds_xy(:,2), -dds_xy(:,1)]./ (sqrt( sum( (dds_xy.*dds_xy)' )' ) *ones(1,2)));

#centroid=(L/(2*N))*sum(xy.*xy.*normal)

dxy = (nVel*ones(1,2)).*normal;

dT = rad_scale/(100* max(max(dxy)) );           %take a time-step that is
                                                %not constant in time, but
%rather changes to make the largest step 1/100th the radius of the drop

new_xy = dT*dxy + xy;

axis([-4,4,-3,3]);
if (mod(iteration,10)==0) %plot every 10 times
    plot(input_xy(:,1),input_xy(:,2),new_xy(:,1),new_xy(:,2))
%plot(1:N, dxy, "+")
%    plot(new_xy(:,1),new_xy(:,2))
end

endfunction

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%End of the evolve function%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

evolve(input_xy);
Areal=Area;
%dT=.1;
%dT = rad_scale/(100* max(max(dxy)) );

#DATA=[input_xy, zeros(N,1);
#      new_xy, ones(N,1)];

```

```

while (((isreal(Area)==1) & (iteration<301)) )
    evolve(new_xy)
# DATA=[DATA;new_xy, iteration*ones(N,1)];
end

```

```
%save 2stabcircle3.dat DATA #data collection
```

build_xy.m

```
% This function creates and initializes the points of the boundary
```

```
function xy = build_xy
```

```
% This build function creates an ellipse with number N points determined
% by the user
```

```
%N = input('\nWe are going to look at points along the boundary of an ellipse. \nPoints are originally defi
```

```
N=100
```

```
%theta is vector of angles, evenly-spaced up to 2pi
theta = 2*pi*(1/N:1/N:1)';
```

```
%define axis lengths for ellipse
a=2;
b=2;
```

```
xy = [a*cos(theta), b*sin(theta)];
```

```
xy(24,2)=1;
xy(25,:)=[0, .2];
xy(26,2)=1;
```

```
xy(74,2)=-1;
xy(75,:)=[0,-.2];
xy(76,2)=-1;
```

```
#
#theta1=2*pi*[2/(N+2):2/(N+2):1-2/(N+2)]';
```

```
#theta2=2*pi*[4/(N+2):2/(N+2):1]';

#xy = [cos((2*pi/N)*[1:N]') .^3, (cos((2*pi/N)*[1:N]')) .^2 .* sin((2*pi/N)*[1:N]')];
```

build_K

```
function K = build_K(s, N)

% the new, improved K is an Nx2 matrix with the diagonal elements in the
% first column and the off-diagonal elements in the second column
% (the corner elements are K(N,2) )

K = [(shift(s,-1)-shift(s,1))/3, (shift(s,-1)-s)/6];

K(1,1)=s(2)/3;           %need to properly deal with node
K(N,1)=( s(1)+s(N)-s(N-1) )/3;   %zero (not == node N!!)
K(N,2)=s(1)/6;         %and nodes > N

%OLD K FUNCTION(SLOOOOOOOW)

% s is the s_vector of arclengths
% K is the tridiagonal (with corners) matrix for gettin' the 2nd derivatives
% Let's define K now...
%
%K=zeros(N,N);
%for i=2:(N-1)
% K(i,i)=(s(i+1)-s(i-1))/3;
% K(i-1,i)=(s(i)-s(i-1))/6;
% K(i,i-1)=(s(i)-s(i-1))/6;
%end
%
%
%K(1,1)=s(2)/3;
%K(N,N)=(s(1)+s(N)-s(N-1))/3;
%
%K(N,1)=(s(1))/6;
%K(N-1,N)=(s(N)-s(N-1))/6;
%
%K(1,N)=(s(1))/6;
```

```
%K(N,N-1)=(s(N)-s(N-1))/6;
%
```

build_b.m

```
function b = build_b(x, s, N)

%nonhomogeneous vector for spline 2nd derivative calculation
%used for function redistribute.m

b = (shift(x,-1)-x)./(shift(s,-1)-s) - (x-shift(x,1))./(s-shift(s,1));
```

arclength.m

```
% returns arclengths for values other than [1,N]
% the input s is the "s_vector" of arclengths for
% nodes 1 through N, and the input "i" is the index
% for which you want the arclength
```

```
function q = arclength(s, i)

N = length(s);

if (i < 1)
    q = s(N+i)-s(N);
elseif (i > N)
    q = s(N)+s(i-N);
else
    q = s(i);
end
```

build_matA.m

```
function G = build_matA(xy, s)

%this function builds the matrix which operates on the velocity
%vector in our matrix equation. We call it "matA" for no good reason

global N;

s_cols=s*ones(1,N);           %matrix of N identical column vectors
```

```

F = (1.0/(2.0*pi))*log(abs( eye(N) + (s(N)/pi)*sin( pi*(s_cols-s_cols')/s(N)) ));

Gdiags=sum(F')';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%first make the G' function, ie the matrix w/o diagonals%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x_cols=(xy(:,1)*ones(1,N));
y_cols=(xy(:,2)*ones(1,N));

G = ( (1.0/(2.0*pi)) *log(sqrt( eye(N) + (x_cols-x_cols').^2 + (y_cols-y_cols').^2 )) ) + diag(Gdiags);

```

build_s_vector.m

```

% at the moment, the arclengths are calculated by using linear
% approximations... kinda crappy, eh?

function s= build_s_vector(xy)

    global N;

    %s = sum(tril( ones(N,1)*sqrt(sum( (diff([xy(N,:);xy]).^2)' )) )')';

    %this is faster!

    s = zeros(N,1);
    s(1) = sqrt((xy(1,1)-xy(N,1))^2+(xy(1,2)-xy(N,2))^2);

    for i=2:N
        s(i) = sqrt((xy(i,1)-xy(i-1,1))^2+(xy(i,2)-xy(i-1,2))^2) + s(i-1);
    end

```

build_solution.m

```

function b = build_solution(xy,der1,der2,angl,a_over_h)

%Handy diagnostics
    global curvature;
    global dGdn;

```

```

N=length(xy(:,2));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%define the curvature vector (nondimensionalized)%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%find where xy is in the left half-plane
left_xy = (xy(:,1) < 0);

curvature=der1(:,1).*der2(:,2)-der1(:,2).*der2(:,1);# - 2*a_over_h*cos(angl);

#curvature=der1(:,1).*der2(:,2)-der1(:,2).*der2(:,1) - 2*a_over_h*cos(angl + (50*pi/180)*xy(:,1).*left_xy);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%define the dG'dn matrix (remember, no diagonal components!)%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xcols=(xy(:,1)*ones(1,N));
ycols=(xy(:,2)*ones(1,N));
xrows=xcols';
yrows=ycols';

dGdn_nodiag = \
    ((ones(N,1)*der1(:,1)')*(yrows-ycols) -
     (ones(N,1)*der1(:,2)')*(xrows-xcols))./( 2*pi*((xrows-xcols).^2 \
    + \
     (yrows-ycols).^2) \
    + realmin);

%by adding in the value "realmin" to the denominator, you no longer
%run into divide-by-zero problems. As a matter of fact, the numerator
%is zero on the diagonal, so dividing by realmin automatically sets
%those entries to zero... ready and awaiting addition of dGdn_diag
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%define the diagonals of our matrix: I*Gamma, first as a vector%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dGdn_diag=sum(dGdn_nodiag,2); %the 2 argument of sum means to
                             %sum along the rows rather than columns

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%and now the real matrix dGdn, with the diagonals...%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dGdn = dGdn_nodiag + diag(dGdn_diag);

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%and the "solution vector" is the product of dGdn and %
%the curvature with our length scale parameter in front%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
b = (1/a_over_h) * (dGdn*curvature);

```

mod.m

```

function m = mod(a,b)

% What?! Octave doesn't have a modulus function?!
% Fuck this open-source shit...

m = a;
c = abs(b);

if (a >= 0)
    while (m >= c)
        m=m-c;
    end

else
n=m;
    while (n < -c)
        n=n+c;
    end
    m=c-n;
end

```

redistribute.m

```

function [newxy, news] = redistribute(oldxy, olds)

my_spline = csape(olds, oldxy, 'periodic'); %this should be
    %in the "pp", or piece-wise polynomial structure format.

N=length(olds);

```

```

news=[olds(N)/N:olds(N)/N:olds(N)]';

%hopefully no reason for this, since I assume it wouldn't work
%my_spline=csape(new_s, ppval(my_spline, new_s), 'not-a-knot');

newxy=ppval(my_spline, news);

#der1=[ppval(splder(csape(olds,oldxy(:,1), 'periodic'),1), news), ppval(splder(csape(olds,oldxy(:,2), 'per
#der2=[ppval(splder(csape(olds,oldxy(:,1), 'periodic'),2), news), ppval(splder(csape(olds,oldxy(:,2), 'per

```

oldredistribute.m

```

function [new_points, new_s_vector] = redistribute(old_points, old_s, N)

K = build_K(old_s, N);      % Banded matrix K (w/corners)

old_x = old_points(:,1);
old_y = old_points(:,2);

b_x = build_b(old_x, old_s, N);
b_y = build_b(old_y, old_s, N);

%der2old_y is the vector of second derivatives of y at the nodes

der2old_x = trisolve(K(1:(N-1),2),K(:,1),K(1:(N-1),2),b_x,K(N,2),K(N,2));
der2old_y = trisolve(K(1:(N-1),2),K(:,1),K(1:(N-1),2),b_y,K(N,2),K(N,2));

%der2old_x=inverse(K)*b_x;
%der2old_y=inverse(K)*b_y;

% Now, to redistribute the nodes so that they are equidistant in arclength
% new_s is an Nx2 matrix where the first column is the values
% {L/N, 2L/N, 3L/N, ... L} and the second column shows where between the old
% nodes the equidistant values lie.  If new_s(i,2) = j, that means that all of
% the old nodes up to (and including) node j have arclength less than iL/N,
% so the new node i lies between old nodes j and j+1

L=old_s(N);
avgL = L/N;

```



```

%%new_s = [[1:N]', [1:N]-ceil([1:N]-old_s/avgL)];    hmm... that's not right...
%%try floor(old_s)-shift(floor(old_s),-1) and then get a vector that
%%records jumps

new_s=zeros(N,2);
new_s(:,1)= [avgL:avgL:N*avgL]';

j=0;                % if j = 0, then
for i=1:N           % we must be sure to define s(0) as 0
    while ( arclength(old_s,j+1) <= new_s(i,1) )
        j++;
    end
    new_s(i,2)=j;
    j--;
end

new_s_vector=new_s(:,1);

nnz_s = find( new_s(:,2) );                %vector of indices of non-zero components of new_s
s_zrs = length(nnz_s);                    %number of non-zero elements of new_s(:,2)

nnz_s_noN = nnz_s(1:(s_zrs-1));

if (s_zrs != N)
    xsi_zros = new_s(1:(N-s_zrs) ,1)/old_s(1);
else
    xsi_zros = [];
end

xsi = [xsi_zros; (new_s(nnz_s_noN, 1) - (old_s( new_s(nnz_s_noN,2) ) ) ) ./ \
        (old_s( new_s(nnz_s_noN,2) + 1) - old_s( new_s(nnz_s_noN,2) ) ) ); 0 ];

%xsi = (new_s(:,1) - (old_s(new_s(:,2) ) ) ) ./ (shift(old_s,-1) - old_s);
xsi(N) = (new_s(N,1) - old_s(new_s(N,2)))/ old_s(1);

tmp_x=[old_x(N)*ones((N-s_zrs),1); old_x(new_s(nnz_s,2))];    %this would just be
                                                                %tmpx=oldx(news(:,2))
tmp_y=[old_y(N)*ones((N-s_zrs),1); old_y(new_s(nnz_s,2))];    %except for those pesky zeros

new_points = \
[tmp_x.*(1-xsi)+shift(tmp_x,-1).*xsi+((shift(old_s,-1)-old_s).^2).*(der2old_x.*(1-xsi).*(xsi.^2-2*xsi)/6+shift(de
tmp_y.*(1-xsi)+shift(tmp_y,-1).*xsi+((shift(old_s,-1)-old_s).^2).*(der2old_y.*(1-xsi).*((xsi.^2)-2*xsi)/6+shift(d

```

```

new_points(N,:) =[old_x(N), old_y(N)];

%
%new_points = zeros(N,2);
%
% These new points are made so as to lie along a cubic spline between old
% nodes k and k+1
%
% % % % % % % % % Time to play hardball, eh?
%
%for i=1:N
% k=new_s(i,2);
%
% xsi = \
%      (new_s(i,1)-arclength(old_s,k))/(arclength(old_s,k+1)-arclength(old_s,k));
%
% new_points(i,1) = extend_nodes(old_x,k)*(1-xsi) + extend_nodes(old_x,k+1)*xsi + (arclength(old_s,k+1)-ar
%
% new_points(i,2) = extend_nodes(old_y,k)*(1-xsi) + extend_nodes(old_y,k+1)*xsi + (arclength(old_s,k+1)-ar
%end
%
%new_s_vector = new_s(:,1);

```

extend_nodes.m

```

function q = extend_nodes(x,i)

% returns node values other than [1,N]
% the input x is the "x_vector" of nodal x-values for
% nodes 1 through N, and the input "i" is the index
% for which you want the x-value

N = length(x);

q = x(xtnd_mod(i,N));

```

take_dds.m

```

function dxds_dyds = take_dds(xy, s)

```

```

N = length(s);

dxds_dyds = zeros(N,2);

dxds_dyds(1,1) = (xy(2,1)-xy(N,1))/s(2);
dxds_dyds(1,2) = (xy(2,2)-xy(N,2))/s(2);

dxds_dyds(N,1) = (xy(1,1)-xy(N-1,1))/(s(N)-s(N-1)+s(1));
dxds_dyds(N,2) = (xy(1,2)-xy(N-1,2))/(s(N)-s(N-1)+s(1));

for i=2:(N-1)
dxds_dyds(i,1) = (xy(i+1,1)-xy(i-1,1))/(s(i+1)-s(i-1));
dxds_dyds(i,2) = (xy(i+1,2)-xy(i-1,2))/(s(i+1)-s(i-1));
end

```

take_der2.m

```

function c = take_der2 (x, y)

n = length(x);

a = y;
b = c = zeros (size (y));
h = diff (x);
idx = ones (columns(y),1);

h = [h; h(1)];

## XXX FIXME XXX --- the following gives a smoother periodic
##transition:
a(n,:) = a(1,:) = ( a(n,:) + a(1,:) ) / 2;
##a(n,:) = a(1,:);

tmp = diff (shift ([a; a(2,:)], -1));
g = 3 * tmp(1:n - 1,:) ./ h(2:n,idx)\
    - 3 * diff (a) ./ h(1:n - 1,idx);

#   if (n > 3)
#       dg = 2 * (h(1:n - 1) .+ h(2:n));

```

```

#     e = h(2:n - 1);
#     c(2:n,idx) = trisolve(dg,e,g,h(1),h(1));
#     elseif (n == 3)
#         A = [2 * (h(1) + h(2)), (h(1) + h(2));
#             (h(1) + h(2)), 2 * (h(1) + h(2))];
#         c(2:n,idx) = inverse(A) * g;
#     endif

#     c(1,:) = c(n,:);
#     b = diff (a) ./ h(1:n - 1,idx)\
#         - h(1:n - 1,idx) / 3 .* (c(2:n,:) + 2 * c(1:n - 1,:));
#     b(n,:) = b(1,:);
#     d = diff (c) ./ (3 * h(1:n - 1, idx));
#     d(n,:) = d(1,:);

#     d = d(1:n-1,:); c=c(1:n-1,:); b=b(1:n-1,:); a=a(1:n-1,:);
#     coeffs = [d(:), c(:), b(:), a(:)];
#     pp = mkpp (x, coeffs);

#der2=6*c./h;
endfunction

```

xtn_d_mod.m

```

function m = mod(a,b)

% What?! Octave doesn't have a modulus function?!
% Fuck this open-source shit...

m = a;
c = abs(b);

if (a > 0)
    while (m > c)
        m=m-c;
    end
elseif (a = 0)
    m = c;
else

```

```

n=m;
    while (n < -c)
        n=n+c;
    end
    m=c-n;
end

```

octsplines.m

```

%try csape(X, Y, 'not-a-knot', 'periodic')
%it should create the pp-form of the cubic spline

## x: sample points
## P: polynomial coefficients for points in sample interval
## P(i,:) contains the coefficients for the polynomial over
interval i
## ordered from highest to lowest. If d > 1, P(r,i,:) contains the
## coefficients for the r-th polynomial defined on interval i.
However
## octave does not support 3-dimensional matrices, so define:
## c = reshape (P(:, j), n, d);
## Now c(i,r) is j-th coefficient of the r-th polynomial over the
## i-th interval.
## n: number of polynomial pieces
## k: order of the polynomial + 1
## d: number of polynomials defined for each interval
function [x, P, n, k, d] = unmkpp(pp)

#csape body.....

function pp = csape (x, y, cond, valc)

x = x(:);
n = length(x);

transpose = (columns(y) == n);
if (transpose) y = y'; endif

a = y;
b = c = zeros (size (y));

```

```

h = diff (x);
idx = ones (columns(y),1);

if (nargin < 3 || strcmp(cond,"complete"))
# specified first derivative at end
    point
    if (nargin < 4)
        valc = [0, 0];
    endif

    dg = 2 * (h(1:n - 2) .+ h(2:n - 1));
    dg(1) = dg(1) - 0.5 * h(1);
    dg(n - 2) = dg(n-2) - 0.5 * h(n - 1);

    e = h(2:n - 2);

    g = 3 * diff (a(2:n,:)) ./ h(2:n - 1,idx)\
        - 3 * diff (a(1:n - 1,:)) ./ h(1:n - 2,idx);
    g(1,:) = 3 * (a(3,:) - a(2,:)) / h(2) \
        - 3 / 2 * (3 * (a(2,:) - a(1,:)) / h(1) - valc(1));
    g(n - 2,:) = 3 / 2 * (3 * (a(n,:) - a(n - 1,:)) / h(n - 1) -
    valc(2))\
        - 3 * (a(n - 1,:) - a(n - 2,:)) / h(n - 2);

    c(2:n - 1,:) = trisolve(dg,e,g);
    c(1,:) = (3 / h(1) * (a(2,:) - a(1,:)) - 3 * valc(1)
        - c(2,:) * h(1)) / (2 * h(1));
    c(n,:) = - (3 / h(n - 1) * (a(n,:) - a(n - 1,:)) - 3 * valc(2)
        + c(n - 1,:) * h(n - 1)) / (2 * h(n - 1));
    b(1:n - 1,:) = diff (a) ./ h(1:n - 1, idx)\
        - h(1:n - 1,idx) / 3 .* (c(2:n,:) + 2 * c(1:n - 1,:));
    d = diff (c) ./ (3 * h(1:n - 1, idx));

elseif (strcmp(cond,"variational") || strcmp(cond,"second"))

    if ((nargin < 4) || strcmp(cond,"variational"))
        ## set second derivatives at end points to zero
        valc = [0, 0];
    endif

    c(1,:) = valc(1) / 2;
    c(n,:) = valc(2) / 2;

```

```

g = 3 * diff (a(2:n,:)) ./ h(2:n - 1, idx)\
  - 3 * diff (a(1:n - 1,:)) ./ h(1:n - 2, idx);

g(1,:) = g(1,:) - h(1) * c(1,:);
g(n - 2,:) = g(n-2,:) - h(n - 1) * c(n,:);

dg = 2 * (h(1:n - 2) .+ h(2:n - 1));
e = h(2:n - 2);

c(2:n - 1,:) = trisolve (dg,e,g);
b(1:n - 1,:) = diff (a) ./ h(1:n - 1,idx)\
  - h(1:n - 1,idx) / 3 .* (c(2:n,:) + 2 * c(1:n - 1,:));
d = diff (c) ./ (3 * h(1:n - 1, idx));

elseif (strcmp(cond,"periodic"))

h = [h; h(1)];

## XXX FIXME XXX --- the following gives a smoother periodic
transition:
##   a(n,:) = a(1,:) = ( a(n,:) + a(1,:) ) / 2;
a(n,:) = a(1,:);

tmp = diff (shift ([a; a(2,:)], -1));
g = 3 * tmp(1:n - 1,:) ./ h(2:n,idx)\
  - 3 * diff (a) ./ h(1:n - 1,idx);

if (n > 3)
  dg = 2 * (h(1:n - 1) .+ h(2:n));
  e = h(2:n - 1);
  c(2:n,idx) = trisolve(dg,e,g,h(1),h(1));
elseif (n == 3)
  A = [2 * (h(1) + h(2)), (h(1) + h(2));
       (h(1) + h(2)), 2 * (h(1) + h(2))];
  c(2:n,idx) = A \ g;
endif

c(1,:) = c(n,:);
b = diff (a) ./ h(1:n - 1,idx)\
  - h(1:n - 1,idx) / 3 .* (c(2:n,:) + 2 * c(1:n - 1,:));
b(n,:) = b(1,:);

```

```

d = diff (c) ./ (3 * h(1:n - 1, idx));
d(n,:) = d(1,:);

elseif (strcmp(cond,"not-a-knot"))

if (n > 4)

    dg = 2 * (h(1:n - 2) .+ h(2:n - 1));
    dg(1) = dg(1) - h(1);
    dg(n - 2) = dg(n-2) - h(n - 1);

    ldg = udg = h(2:n - 2);
    udg(1) = udg(1) - h(1);
    ldg(n - 3) = ldg(n-3) - h(n - 1);

elseif (n == 4)

    dg = [h(1) + 2 * h(2), 2 * h(2) + h(3)];
    ldg = h(2) - h(3);
    udg = h(2) - h(1);

endif
g = zeros(n - 2,columns(y));
g(1,:) = 3 / (h(1) + h(2)) * (a(3,:) - a(2,)\
    - h(2) / h(1) * (a(2,:) - a(1,)));
if (n > 4)
    g(2:n - 3,:) = 3 * diff (a(3:n - 1,:)) ./ h(3:n - 2,idx)\
        - 3 * diff (a(2:n - 2,:)) ./ h(2:n - 3,idx);
endif
g(n - 2,:) = 3 / (h(n - 1) + h(n - 2)) *\
    (h(n - 2) / h(n - 1) * (a(n,:) - a(n - 1,)) -\
    (a(n - 1,)) - a(n - 2,));
c(2:n - 1,:) = trisolve(ldg,dg,udg,g);
c(1,:) = c(2,:) + h(1) / h(2) * (c(2,:) - c(3,:));
c(n,:) = c(n - 1,:) + h(n - 1) / h(n - 2) * (c(n - 1,)) - c(n - \
2,:));
b = diff (a) ./ h(1:n - 1, idx)\
    - h(1:n - 1, idx) / 3 .* (c(2:n,)) + 2 * c(1:n - 1,));
d = diff (c) ./ (3 * h(1:n - 1, idx));

else
msg = sprintf("unknown end condition: %s",cond);

```



```

    error (msg);
endif

d = d(1:n-1,:); c=c(1:n-1,:); b=b(1:n-1,:); a=a(1:n-1,:);
coeffs = [d(:), c(:), b(:), a(:)];
pp = mkpp (x, coeffs);

endfunction

%!shared x,y,cond
%! x = linspace(0,2*pi,15)'; y = sin(x);

%!assert (ppval(csape(x,y),x), y, 10*eps);
%!assert (ppval(csape(x,y),x'), y', 10*eps);
%!assert (ppval(csape(x',y'),x'), y', 10*eps);
%!assert (ppval(csape(x',y'),x), y, 10*eps);
%!assert (ppval(csape(x,[y,y]),x), \
%!      [ppval(csape(x,y),x),ppval(csape(x,y),x)], 10*eps)

%!test cond='complete';
%!assert (ppval(csape(x,y,cond),x), y, 10*eps);
%!assert (ppval(csape(x,y,cond),x'), y', 10*eps);
%!assert (ppval(csape(x',y',cond),x'), y', 10*eps);
%!assert (ppval(csape(x',y',cond),x), y, 10*eps);
%!assert (ppval(csape(x,[y,y],cond),x), \
%!      [ppval(csape(x,y,cond),x),ppval(csape(x,y,cond),x)], 10*eps)

%!test cond='variational';
%!assert (ppval(csape(x,y,cond),x), y, 10*eps);
%!assert (ppval(csape(x,y,cond),x'), y', 10*eps);
%!assert (ppval(csape(x',y',cond),x'), y', 10*eps);
%!assert (ppval(csape(x',y',cond),x), y, 10*eps);
%!assert (ppval(csape(x,[y,y],cond),x), \
%!      [ppval(csape(x,y,cond),x),ppval(csape(x,y,cond),x)], 10*eps)

%!test cond='second';
%!assert (ppval(csape(x,y,cond),x), y, 10*eps);
%!assert (ppval(csape(x,y,cond),x'), y', 10*eps);
%!assert (ppval(csape(x',y',cond),x'), y', 10*eps);
%!assert (ppval(csape(x',y',cond),x), y, 10*eps);

```

```
%!assert (ppval(csape(x,[y,y],cond),x), \
%!      [ppval(csape(x,y,cond),x),ppval(csape(x,y,cond),x)], 10*eps)

%!test cond='periodic';
%!assert (ppval(csape(x,y,cond),x), y, 10*eps);
%!assert (ppval(csape(x,y,cond),x'), y', 10*eps);
%!assert (ppval(csape(x',y',cond),x'), y', 10*eps);
%!assert (ppval(csape(x',y',cond),x), y, 10*eps);
%!assert (ppval(csape(x,[y,y],cond),x), \
%!      [ppval(csape(x,y,cond),x),ppval(csape(x,y,cond),x)], 10*eps)

%!test cond='not-a-knot';
%!assert (ppval(csape(x,y,cond),x), y, 10*eps);
%!assert (ppval(csape(x,y,cond),x'), y', 10*eps);
%!assert (ppval(csape(x',y',cond),x'), y', 10*eps);
%!assert (ppval(csape(x',y',cond),x), y, 10*eps);
%!assert (ppval(csape(x,[y,y],cond),x), \
%!      [ppval(csape(x,y,cond),x),ppval(csape(x,y,cond),x)], 10*eps)
```

Bibliography

- [1] D. J. Acheson. *Elementary Fluid Dynamics*. Oxford University Press, 1990.
- [2] Robert Almgren, Andrea Bertozzi, and Michael Brenner. Stable and unstable singularities in the unforced hele-shaw cell. *Phys. Fluids*, 8(6):1356–1370, 1996.
- [3] C. Andrieu, D. A. Beysens, V. S. Nikolayev, and Y. Pomeau. Coalescence of sessile drops. *J. Fluid Mech.*, 453:427–438, 2002.
- [4] Wei-Shen Dai and Michael J. Shelley. A numerical study of the effect of surface tension and noise on an expanding hele-shaw bubble. *Phys. Fluids A*, 9:2131–2146, 1993.
- [5] P. G. de Gennes. Wetting: statics and dynamics. *Rev. Mod. Phys.*, 57:827–863, 1985.
- [6] David J. Griffiths. *Introduction to Electrodynamics*. Prentice-Hall, 1999.
- [7] Richard Haberman. *Elementary Applied Partial Differential Equations*. Prentice-Hall, 1998.
- [8] Keck Graduate Institute. Microfluidics research laboratory. <http://faculty.kgi.edu/sterling/Microfluidics>
- [9] Hyeong-Gi Lee, J. S. Lowengrub, and J. Goodman. Modelling pinchoff and reconnection in a hele-shaw cell part i: The models and their calibration. Unpublished, 2001.

- [10] Hyeong-Gi Lee, J. S. Lowengrub, and J. Goodman. Modelling pinchoff and reconnection in a hele-shaw cell part ii: Analysis and simulation in the non-linear regime. Unpublished, 2001.
- [11] David R. Lide, editor. *Handbook of Chemistry and Physics*. CRC Press, 1997.
- [12] A. Nadim, A Borhan, and H Haj-Hariri. Simulation of the bouyancy-driven motion of a drop in a hele-shaw cell by the boundary integral method. *Boundary Elements XVII*, pages 613–620, 1995.
- [13] Nanolytics. Making microfluidics just a tool. www.nanolytics.com.
- [14] V. S. Nikolayev and D. A. Beysens. Relaxation of nonspherical sessile drops towards equilibrium. *Phys. Rev., E* in press, 2001.
- [15] M. R. H. Nobari and G. Tryggvason. Numerical simulations of drop collisions. In *NASA Technical Memorandum*, volume 94-0835. American Institute of Aerodnautics and Astronautics, 1994.
- [16] H. Power and L. C. Wrobel. *Boundary Integral Methods in Fluid Mechanics*. Computational Mechanics Publications, 1995.
- [17] C. Pozrikidis. *Boundary integral and singularity methods for linearized viscous flow*. Cambridge University Press, 1992.
- [18] B. M. Rush and A. Nadim. The shape oscillations of a two-dimensional drop including viscous effects. *Engineering Analysis with Boundary Elements*, 24:43–51, 2000.
- [19] P. G. Saffman. A note on the motion of bubbles in a hele-shaw cell and porous medium. *Quarterly Journal of Mechanics and Applied Mathematics*, 7:265–279, 1959.

- [20] Ivar Stakgold. *Green's Functions and Boundary Value Problems*. John Wiley & Sons, Inc., 1979.
- [21] Ivar Stakgold. *Green's Functions and Boundary Value Problems*. John Wiley & Sons, 1979.
- [22] Duke University. Digital microfluidics by electrowetting. <http://www.ece.duke.edu/Research/microfluidics/>.
- [23] S. G. Yiantsios and R. H. Davis. Close approach and deformation of two viscous drops due to gravity and van der waals forces. *J. Colloid Interface Sci.*, 144:412–433, 1991.
- [24] Alexander Z. Zinchenko, Michael A. Rother, and Robert H. Davis. Cusping, curvature, and breakup of interacting drops by a cureveless boundary integral algorithm. *J. of Fluid Mech.*, 391:242–292, 1999.