Math 451H
Fluid Flow – Final Report

Jeremy P. Carlo
Pritam O. Dodeja
Jeffrey Fernandez
Rupen Patel
Mark Timonera
Rafal Turek

Instructor: Prof. Lou Kondic

Abstract

This paper discusses the problem of a viscous liquid flowing down an inclined plane. Such fluids are known to exhibit a fingering-type instability, in which the contact line of the fluid becomes distorted in a wavy pattern. This effect is familiar to anyone who has seen the flow of fluid down a wall, in which a uniform contact line quickly turns into a wavy pattern, with very rapidly moving "fingers" and slowly moving spaces between them. A theoretical treatment of viscous fluid flow will be developed, starting from the Navier-Stokes equations to derive an approximation for thin fluid films (the lubrication approximation), and an analysis of flow stability to perturbations will be carried out. Comparisons of the theoretical results with numerical simulations and experimental trials will be made. In particular, the growth rate in length and the transverse width of the fingers will be analyzed using theoretical, numerical, and experimental techniques, and the values derived from each compared.

Introduction to the Theory

The behavior of a viscous fluid may be analyzed theoretically by means of the Navier-Stokes equations:

$$D\mathbf{u}/Dt = (\mathbf{u}\bullet\nabla)u + \partial\mathbf{u}/\partial t = -\nabla P/\rho + (\mu/\rho)\nabla^2\mathbf{u} + \mathbf{g} \qquad \textbf{(1)}$$

Here $\mathbf{u} = (u_x\ ,\ u_{y,},\ u_z) = (\mathbf{v}\ ,\ w)$ is the fluid velocity, P the fluid pressure, μ the viscosity, ρ the density, and g the gravitational acceleration. The term on the left represents the fluid's inertia, while the terms on the right represent the pressure gradient, viscosity, and gravity, respectively. This equation will be simplified to develop the thin film lubrication approximation for our theoretical analysis.

Our coordinate system will be set up as shown in *Figure 1* below. The plane is

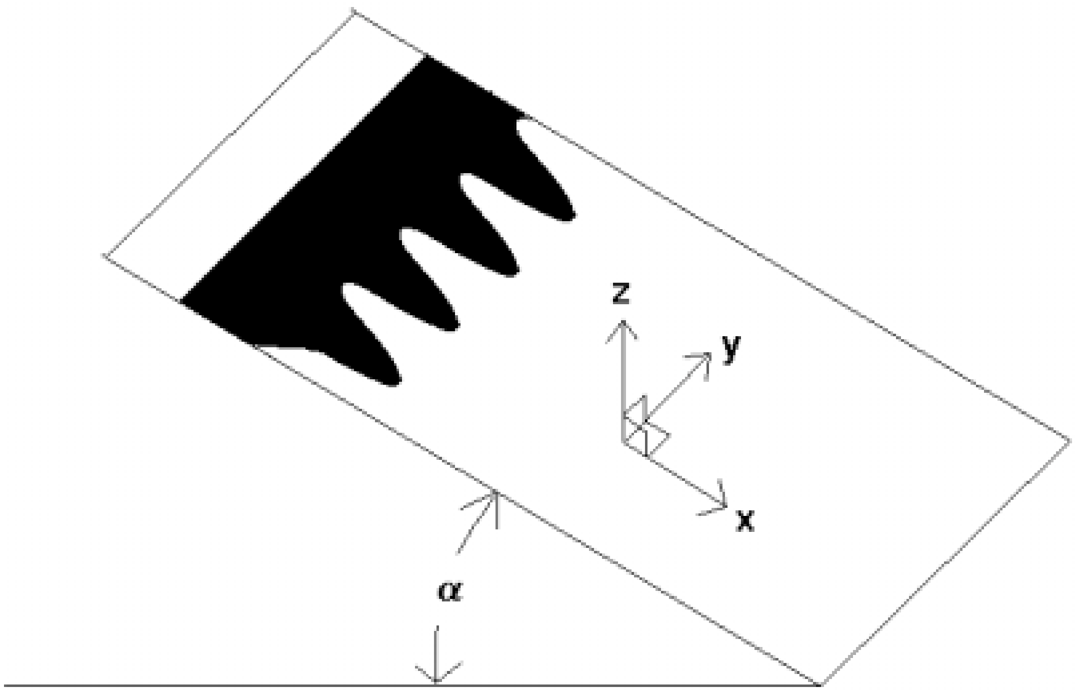


Figure 1:
Coordinate System and Basic Setup

inclined to the horizontal at an angle α, with flow in the *x* direction. The *y* axis is in the plane but perpendicular to the flow direction, and the *z* axis is normal to the plane. Thus the gravitational term may be written as $\mathbf{g} = g^*[\mathbf{i}\sin(\alpha) - \mathbf{k}\cos(\alpha)]$, where **i** and **k** are the unit normal vectors in the *x* and *z* direction, respectively.

Derivation of the Lubrication Approximation

First we will assume that the flow has a small Reynolds number (i.e., very viscous and/or slow, $Re = UL\rho/\mu << 1$, where *U* is a typical velocity and *L* a typical length scale), so that the inertia term may be discarded, and the right side of *(1)* set equal to zero. In addition, we assume that the in-plane derivatives of the in-plane velocity of the thin film are smaller than the derivative normal to the plane: $\partial^2|\mathbf{v}|/\partial x^2$,

$\partial^2|\mathbf{v}|/\partial y^2 << \partial^2|\mathbf{v}|/\partial z^2$. This directly implies that the normal component of velocity is smaller than the in-plane component of velocity, i.e. $w << |\mathbf{v}|$. Therefore, we may split *(1)* componentwise:

$$\mu\, \partial^2\mathbf{v}/\partial z^2 + \rho\, g \sin(\alpha)\mathbf{i} - \nabla_2 P = 0 \quad (x \text{ and } y \text{ components})$$

$$\partial P/\partial z + \rho\, g \cos(\alpha) = 0 \qquad (z \text{ component}) \tag{2}$$

where ∇_2 represents the gradient taken only in the x and y directions. From integration of the z component we find that $P = C(x,y) - \rho\, g\, z \cos(\alpha)$.

We denote the height of the film by $h(x,y)$, and we note that the pressure at the surface of the fluid must satisfy the Laplace-Young boundary condition: $P(h) = P_0 - \gamma\,\kappa$, where κ represents the curvature of the surface at the point (x,y,h), and γ is the surface tension of the fluid. With this boundary condition, we can solve for the constant of integration $C(x,y)$ above, and we find that

$$P(z) = P_0 - \gamma\,\kappa - \rho\, g\, (z-h)\cos(\alpha). \tag{3}$$

If we integrate the *x-y* component of *Equation (2)* twice, using the expression just found for P, we find $\mathbf{v} = {}^{1}/_{\mu}\nabla_2 P(z^2/2) + Az + B - \rho\, g\, (z^2/2\mu) \sin(\alpha)\mathbf{i}$. If we assume a no-slip boundary condition at the fluid-surface interface ($z = 0$), we find $A = -(h/\mu)\nabla_2 P + \rho\, g\, (z^2/2h\mu) \sin(\alpha)\mathbf{i}$. Further, we may average the in-plane velocity $\mathbf{v}$ over the thickness of the film (by integrating in the z direction and dividing by h), and find

$$\langle v \rangle = -(h^2/3\mu)\, [\nabla P - \rho\, g \sin(\alpha)\mathbf{i}] \tag{4}$$

Next, using the incompressibility of the fluid (i.e. $\nabla\bullet(h\mathbf{v}) + \partial h/\partial t = \mathrm{D}h/\mathrm{D}t = 0$), and making the approximation $\kappa \approx \nabla^2 h$, and we find that

$$3\mu\, \partial h/\partial t + \nabla\bullet[\gamma\, h^3\, \nabla\nabla^2 h] - \rho\, g \cos(\alpha)\nabla\bullet[h^3\nabla h] + \rho\, g \sin(\alpha)\partial h^3/\partial x = 0 \tag{5}$$

Equation (5) is known as the *thin film equation* and will be the differential equation used for the rest of this analysis. The second term represents capillarity of the fluid, and the last two terms represent gravity. Again, h is the fluid thickness at point (*x,y*), with α the inclination angle of the plane, g the gravitational acceleration, and γ, μ, and ρ the fluid surface tension, viscosity and density, respectively.

As a final step, we will nondimensionalize this equation to simplify the theoretical analysis.

1. Far from the contact line, the film has a constant thickness of h_c, so we will normalize the film thickness by this amount: $z = z/h_c$. In reality, the film will gradually thin far behind the contact line, but we assume that the thickness far behind the contact line will be a constant. As further justification for this assumption, the linear analysis we will perform will be used only for very short flow times and small finger amplitudes, where the thinning effect will be less pronounced.
2. We will use a "natural" length scale x_c in the plane, and normalize x and y by this amount: $x = x/x_c$, $y = y/x_c$.
3. We define U to be a typical in-plane velocity, and normalize time by $t = t/t_c$, *with* $t_c = x_c/U$.
4. Further, we define the capillary number: $Ca = \mu U/\gamma$.
5. Finally, we incorporate all the fluid constants and the plane inclination into a function $D(\alpha) = (3Ca)^{1/3} * \cot(\alpha)$.

With these normalizations, *Equation (5)* is placed in the much simpler nondimensional form

$$\partial h/\partial t + \nabla \bullet [h^3 \nabla \nabla^2 h] - D \nabla \bullet [h^3 \nabla h] + \partial h^3/\partial x = 0 \qquad \textbf{(6)}$$

At this point, we are ready to begin our analysis of the lubrication approximation. However, it should be noted that the no-slip boundary condition at the fluid-surface interface is rather strict, and in fact allows for no fluid flow at all. In reality, there is some amount of slip, since we know that fluids do in fact flow. For our theoretical development, to get around the no-slip condition we assume there is a very thin film in front of the contact line. This precursor film, of normalized thickness *b* << 1, allows fluid to flow without losing the important no-slip boundary condition.

Solution in One Dimension

Next we wish to try and solve *Equation (6).* First, we will make a simplifying assumption, and make the fluid thickness uniform across the plane, so that *h* depends only on the *x* coordinate. In this manner, *Equation (6)* simplifies to

$$h_t + [h^3 h_{xxx}]_x - D^* [h^3 h_x]_x + h^3{}_{xxx} = 0 \qquad \textbf{(7)}$$

$h(0) = 1$ (far behind contact line), $h(L_x) = b$ (at contact line)

$h_{xxx}(0) = h_{xxx}(L_x) = 0$

The above boundary-value problem has a solution which looks roughly like *Figure 2* below:

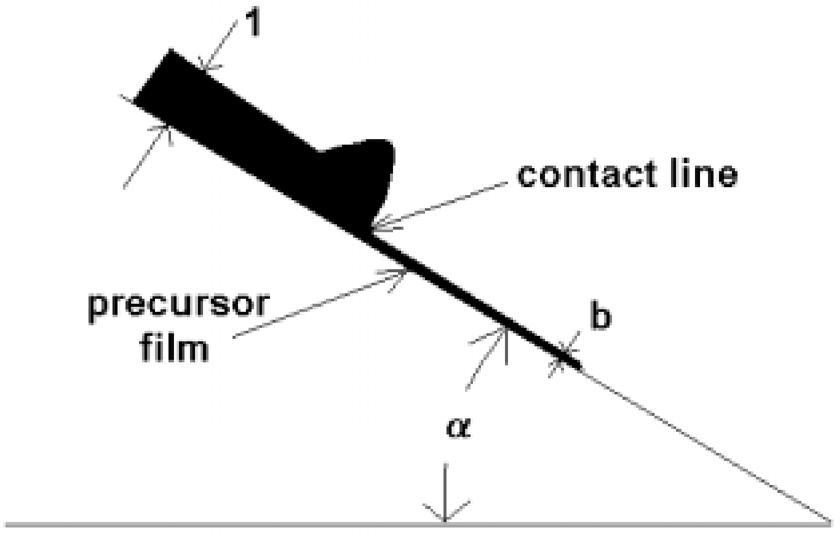


Figure 2: Profile $h_o(x)$

The size of the bump near the contact line will be a function of the plane's inclination angle. Note that far behind the contact line the profile is flat, with a normalized height of 1, and the thickness at and beyond the contact line is a flat value of *b*.

This solution looks something like a traveling wave. Assigning velocity *U* to the wavefront, and replacing *x* with *x* = *x –Ut*, we can integrate *Equation (7)* once to obtain a solution of the form

$$h^3*[1 + h_{xxx}] – U*h – D*h^3*h_x = d \qquad \textbf{(8)}$$

where V and d are constants of integration. Plugging in the boundary conditions in *(7)*, we find $U = 1 + b + b^2 = (1-b^3)/(1-b)$, and $d = -b/(1+b)$.

Extending to Two Dimensions

The above is a solution *(8)* for *h*(x), where the film thickness depends only on *x*. Of course, we are interested in fingering instabilities in the *y* direction, so this solution is of limited interest.

To include *y* behavior in the solution, we let

$$h(x,y,t) = h_0(x) + \varepsilon*\exp(iqy)*G(x,t) \qquad \textbf{(9)}$$

Here, h_0 is the solution depending only on x found in *Equation (8),* and we are writing the *y* dependence of *h* as a Fourier sum of oscillations of spatial frequency *q,* where the wavelength of the disturbances is given by $\lambda = 2\pi/q$. The value ε is assumed to be small and represents the amplitude of the oscillation. In addition, there is a function *G(x,t),* which is a function of both time and the *x* coordinate; what G looks like determines the stability of the profile to small perturbations of amplitude $\varepsilon*G$ and *y* spatial frequency *q*. In particular, we will assume *G* has the form $G(x,t) = G(x)*\exp(\beta*t).$ We will study β as a function of disturbance frequency *q*, to see which wavelengths of fingers are stable or unstable, and how quickly they can be expected to grow or decay.

The details of the stability analysis are presented in *Appendix A*; what will follow is a brief treatment of the analysis, with details available in the *Appendix*.

Linear Stability Analysis

The first step in the linear stability analysis is to expand *Equation (5)* fully, with only terms of up to O(ε) kept (higher orders are discarded since ε is assumed small).

$$(g)_t - U\cdot(g)_x + 3\cdot\left(h_o^2\cdot g\right)_x + \nabla\left[3\cdot g\cdot h_o^2\cdot\nabla\cdot\nabla^2\cdot\left(h_o\right) + h_o^3\cdot\nabla\cdot\nabla^2\cdot g - D(\alpha)\cdot\left(h_o^3\cdot\nabla\cdot g + 3\cdot g\cdot h_o^2\cdot\nabla\cdot h_o\right)\right] = 0$$

The terms of order ε are shown in *Equation 14* in the *Appendix*, reproduced below:

The next step is to take the Fourier transform in *y* of *Equation 14*. The result is given as *Equation 23* in the Appendix. This result holds for various inclination angles

$$\begin{aligned}&(g(x))_t + \left[3\cdot g(x)\cdot h_o(x)^2\cdot\left(h_o(x)\right)_{xxx}\right]_x + \left[h_o(x)^3\cdot(g(x))_{xxx}\right]_x - q^2\cdot h_o(x)^3\cdot(g(x))_{xx} - \\ &-\left[q^2\cdot\left[3\cdot h_o(x)^2\cdot\left(h_o(x)\right)_x\cdot(g(x))_x + h_o(x)^3\cdot(g(x))_{xx}\right]\right] + q^4\cdot h_o(x)^3\cdot g(x) + 3\cdot\left(h_o(x)^2\cdot g(x)\right)_x - (g(x))_x + \\ &+\left[3\cdot g(x)\cdot h_o(x)^2\cdot\left[-D(\alpha)\cdot\left(h_o(x)\right)_x\right]\right]_x - \left[D(\alpha)\cdot(g(x))_x\cdot h_o(x)^3\right]_x + D(\alpha)\cdot q^2\cdot h_o(x)^3\cdot g(x) = 0\end{aligned}$$

(varying values of *D(α)*), and with the value of *U* set to 1:

Next, we wish to expand the growth rate β in powers of *q*. Noting that *G* is an even function of *q*, we can expand β and *G* in even powers of *q*:

$$\beta(q) = \beta_0 + \beta_1 q^2 + \beta_2 q^4 + \ldots$$
$$G(x) = B_0 * (g_0(x) + q^2 g_1(x) + q^4 g_2(x) + \ldots) \qquad \textbf{(10)}$$

After doing this, we substitute the expanded *G(x,t)* into *Equation 23* in the

$$\begin{aligned}&\left(g(x)_0 + q^2\cdot g(x)_1\right)\cdot\left(\beta_0 + \beta_1\cdot q^2\right) + h_o(x)^3\cdot\left[q^4\cdot\left(g(x)_0 + q^2\cdot g(x)_1\right) - q^2\cdot\left[\left(g(x)_0\right)_{xx} + q^2\cdot\left(g(x)_1\right)_{xx}\right]\right] + 2\cdot\left[\left(g(x)_0\right)_x + q^2\cdot\left(g(x)_1\right)_x\right] + \\ &+ 3\cdot h_o(x)^2\cdot h_o(x)_x\cdot\left[-\left(g(x)_0\right)_{xxx} - \left(q^2\cdot g(x)_1\right)_{xxx} + q^2\cdot\left[\left(g(x)_0\right)_x + q^2\cdot\left(g(x)_1\right)_x\right]\right] + \\ &h_o(x)^3\cdot\left[\left(g(x)_0\right)_{xxxx} + q^2\cdot\left(g(x)_1\right)_{xxxx} - q^2\cdot\left[\left(g(x)_0\right)_{xx} + q^2\cdot\left(g(x)_1\right)_{xx}\right]\right] + \\ &+ -D(\alpha)\cdot\left[\left[\left(g(x)_0\right)_{xx} + q^2\cdot\left(g(x)_1\right)_{xx}\right]\cdot h_o(x)^3 + 3\left[\left(g(x)_0\right)_x + q^2\cdot\left(g(x)_1\right)_x\right]\cdot h_o(x)^2\cdot h_o(x)_x - q^2\cdot h_o(x)^3\cdot\left(g(x)_0 + q^2\cdot g(x)_1\right)\right] = 0\end{aligned}$$

Appendix, and we derive *Equation 33* (using terms of up to order 4 in *q*):

We note that *Equation 33* has many terms of varying even powers of *q* set equal to zero; in order for equality to hold for all *q*, the terms of each power in *q* must individually add up to zero. Thus, we can separate terms of varying orders in *q*; the zeroth order terms are given in *Equation 34*, and second order terms in *Equation 37*.

Looking at the zeroth order terms in *Equation 34:*

$$g(x)_0 \cdot \beta_0 + 2\cdot\left(g(x)_0\right)_x + 3\cdot h_0(x)^2 \cdot h_0(x)_x \cdot \left(g(x)_0\right)_{xxx} + h_0(x)^3 \cdot \left(g(x)_0\right)_{xxxx} - D(\alpha)\cdot\left[h_0(x)^3\cdot\left(g(x)_0\right)_{xx} + 3\, h_0(x)^2 \cdot h_0(x)_x \cdot \left(g(x)_0\right)_x\right] = 0$$

we can find that $\beta_0 = 0$. From *Equation 37:*

$$g(x)_0 \cdot \beta_1 - h_0(x)^3 \cdot \left(g(x)_0\right)_{xx} + 2\cdot\left(g(x)_1\right)_x + 3\cdot h_0(x)^2 \cdot h_0(x)_x \cdot\left[\left(g(x)_1\right)_{xxx} - \left(g(x)_0\right)_x\right] + h_0(x)^3 \cdot\left[\left(g(x)_1\right)_{xxxx} - \left(g(x)_0\right)_{xx}\right] + \blacksquare$$
$$\blacksquare + -D(\alpha)\cdot\left[h_0(x)^3\cdot\left(g(x)_1\right)_{xx} + 3\cdot\left(g(x)_1\right)_x \cdot h_0(x)^2 \cdot h_0(x)_x - g(x)_0 \cdot h_0(x)^3\right] = 0$$

we derive an integral representation for β_1 shown in *Equation 43:*

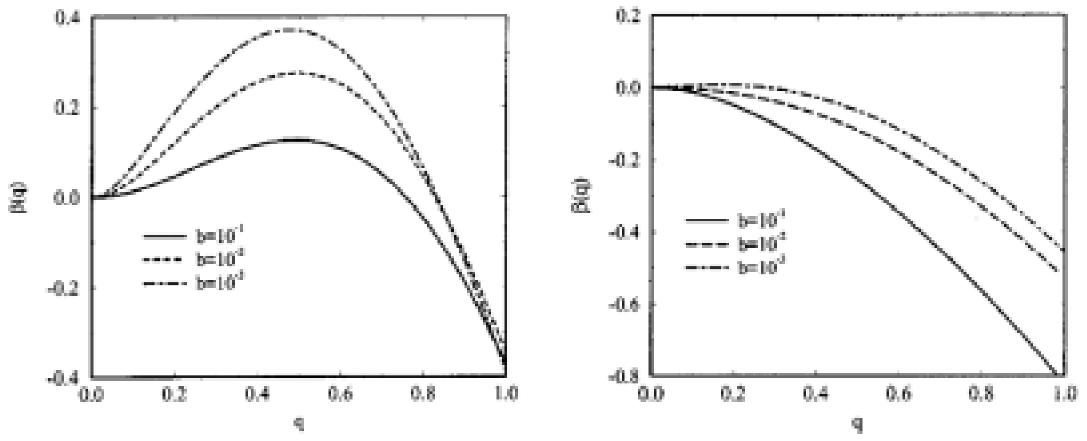
$$\beta_1 = \int_0^\infty h_0(x)_{xxx} \cdot h_0(x)^3 \, dx$$

This type of analysis can be carried to higher orders to find β_2 and higher powers, although it gets enormously complicated.

Instead of seeking analytic results for these higher powers of β, we will try to find numerical results for β, using our earlier result for h_0. When we do this, we find that $\beta = 0$ at q = 0, increases to some maximum value (where the disturbance grows most quickly and is thus the most unstable mode), then decreases to zero and negative values. The decrease in β at high q is due to the effects of surface tension, which tends to wipe out very small oscillations of high spatial frequency (and thus large surface curvatures). The most unstable wavelength *q* and the value of β at this *q* is a function of the artificial precursor film thickness *b*, although the dependence on *b* of $q_{unstable}$ is rather weak. In addition, the most unstable mode is an implicit function of the fluid parameters and plane inclination (all contained in the function $D(\alpha)$), affecting the solution by changing the shape of the 1-dimensional profile $h_0(x)$; all terms explicitly containing $D(\alpha)$ cancel one another out, as shown in the *Appendix*.

A typical numerical result for $\beta(q)$ is shown below in *Figure 3*. It was derived numerically by summing the first three terms in the expansion for β, and presented in Ref. 1 by Bertozzi and Brenner. While the results depend on the assumed value of b, and depend on $D(\alpha)$, the general result is that there is some q for which the instability is maximized. For large q, surface tension represented in the quartic term, with $\beta_2 < 0$ becomes dominant. In addition, for large D (small inclination angles), there is no q for maximum instability; for these D values the flow will be stable with respect to oscillations in the y-direction (at least to this first linear approximation).

In practice it will turn out that observed fingering instabilities will be most prominent for $q \approx q_{unstable}$, exhibiting exponential growth with growth rate β, at least when



The growth rate $\beta(q)$ computed from the long time behavior of solutions of the linear PDE (12) with $D(\alpha)=0$ (top) and $D(\alpha)=5$ (bottom).

Figure 3: Numerical results for β(q)*, from Bertozzi and Brenner [3].*

the fingers are small, so that the above linear analysis (including a dropping of all terms of higher orders in ε) is valid. The following sections present numerical and experimental results concerning these fingering instabilities.

Numerical Analysis

$$\delta h/\delta t + [h^3 h_{xxx}]_x - D[h^3 h_x]_x + (h^3)_x = 0$$

To solve this partial differential equation the following methods were used:

- Finite differences used to solve in space
- Explicit method: Forward Euler used to solve in time

The simplified algorithm used to solve this equation is as follows:

```
For t = 0, to T
        {
            For i = 1, to I = N -1
                 h[i] = h[i] + delta_t * G(i)
             t = t + delta_t
    }
```

Where G(i) is equal to: -A(i) + D*B(i) – C(i):

Where the following terms are equal to:

A(i) = [**a**(h[i], h[i+1])*(h[i+2] – 3* h[i+1]+ 3*h[i] – h[i-1]) - **a**(h[i-1], h[i])*(h[i+1] – 3*h[i]+ 3* h[i-1] - h[i-2])] / (delta_x$^{4)}$

a(h[a], h[b]) = (h[a]^3 + h[b]^3) / 2

B(i) = [**a**(h[i], h[i+1])*(h[i+2] - h[i]) - **a**(h[i-2], h[i-1])*(h[i] - h[i-2])] / (4*delta_x$^{2)}$

$C(i) = [\ (h[i+1])^{3} - (h[i-1])^{3}\ ]\ /\ (2*delta\_x)$

$D = (3*Ca)^{1/3}\cot(\alpha)$

Program Listing:

```
#include <condefs.h>
#include <iostream.h>
#include <fstream.h>

#pragma hdrstop


const int N= 200;        // Division of space into 50 equally sized blocks
const long double N_d = 200;

// Time Step
const long double delta_t = 0.000005;//*( (long double)( 1.00 /16.00 ));

const int N_time_steps = 1000000;        // number of time steps

const int output = 10000;   // interval for every output

const long double delta_x = 20/(N_d - 1);  // space between points

const long double precursor = 0.1;

long double H[N],H_TEMP[N],K1[N],K2[N];              // each index is a position;

bool Log = false;


//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

// Forward Reference for functions


// Set boundary conditions
void boundary();


// Set initial conditoins
void initial_condition();


long double G( const long double H[], int i);

// first term:  4'th order term
long double A( const long double H[], int i);

// part of the second term: 2'nd order term
long double B( const long double H[], int i);


// third term:  1's order term
long double C( const long double H[], int i);


// part of the second term: coefficient
long double D();


// part of the second term:
long double I( const long double H[], int i);


// part of the first term: 3'rd order term
long double L( const long double H[], int i);
```

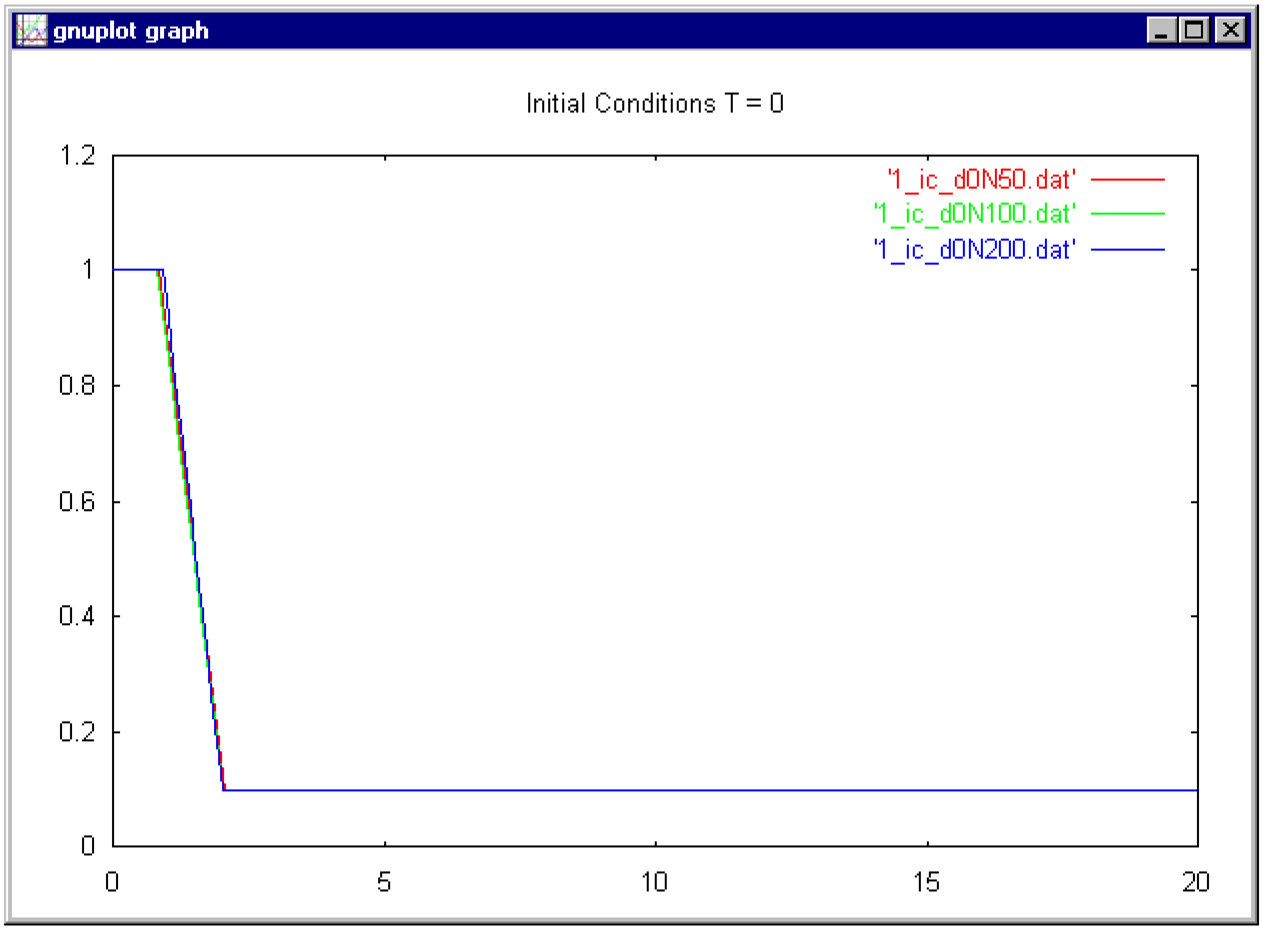
```
// h^3
long double f( const long double H[], int a, int b);





//---------------------------------------------------------------------------
ofstream fout;


#pragma argsused
int main(int argc, char **argv)
{

        int time_lengH;
        char file_name[20];

        cout<<" \n\n\t Beginning Forward Euler.........\n\n";
        cout<<" \t  Press enter to begin. ";
        cin.get();

        cout<<"\n\n\t Please enter the file name for the output: ";
        cin>>file_name;

       cout<<"\n\n\t Please enter the number of seconds to simulate = ";
       cin>>time_lengH;
       cin.get();


        // create output file
        fout.open(file_name);

        //output formatting
                                fout.setf(ios::fixed);
                                fout.setf(ios::showpoint);
                                fout.setf(ios::right);
        fout.precision(14);

                                cout.setf(ios::fixed);
                                cout.setf(ios::showpoint);
                                cout.setf(ios::right);
        cout.precision(14);


        // set Boundary values
        boundary();

        // set initial conditions
        initial_condition();


       fout<<"#\n#\t Number of points ="<<N<<endl;
       fout<<"#\n#\t delta_t = "<<delta_t<<endl;
       fout<<"#\n#\t D = "<<D()<<endl;
       fout<<"#\n#\t delta_x = "<<delta_x<<endl;
       fout<<"#\n#\t Domain = "<<delta_x*(N -1)<<endl;
       fout<<"#\n#\t precursor ="<<precursor<<"\n#\n#"<<endl;


       fout<<"#\tInitial conditions:  "<<endl;
        for( int j = 0; j < N ; j++)
            {
                fout<<"#\t H["<<j<<"] = "<<H[j]<<"."<<endl;
                //cout<<"\t H["<<j<<"] = "<<H[j]<<"."<<endl;
            }


// set initializatoin data
```

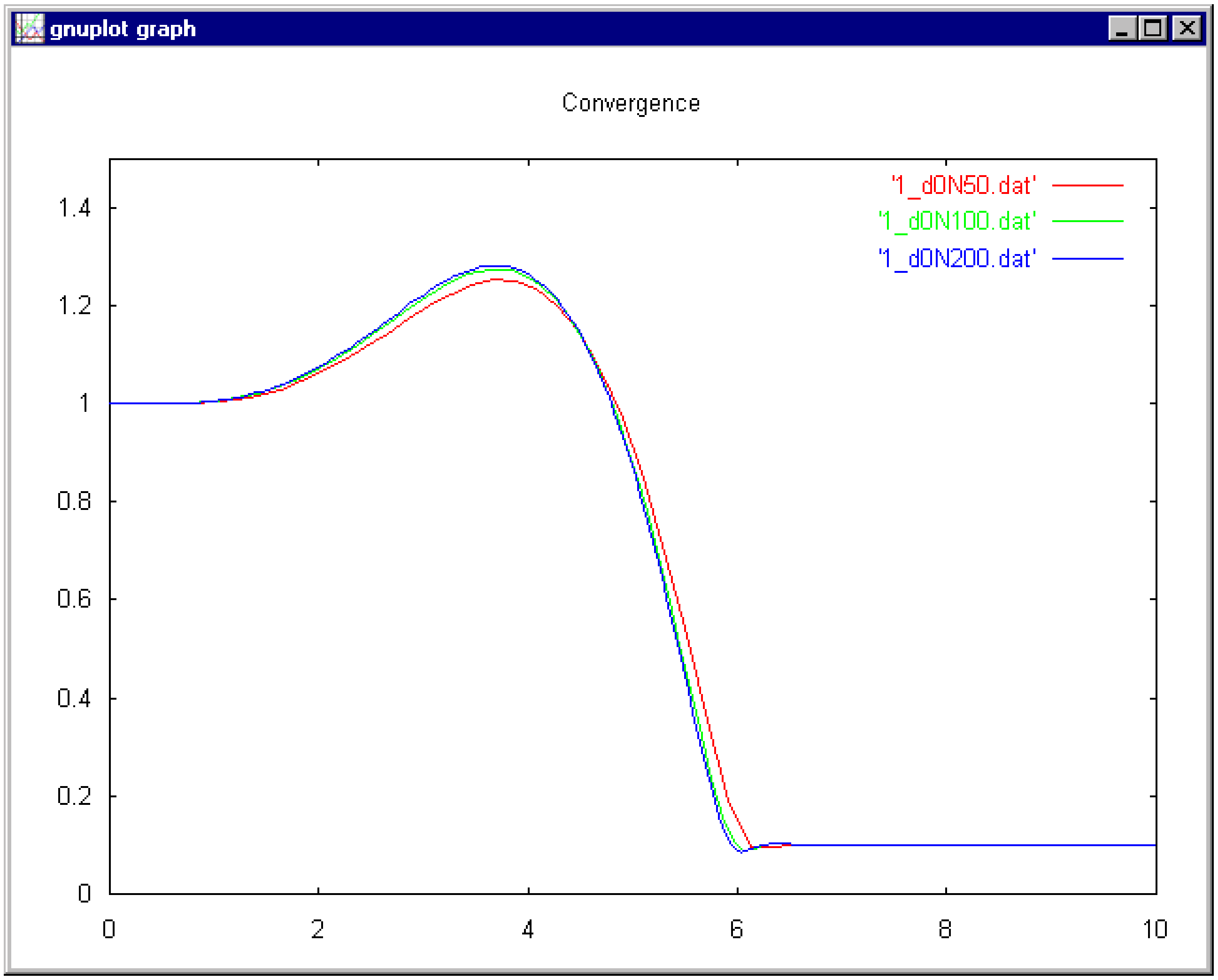
```
long double t = 0;

long double wall = H[N-2];

long double time_stamp =0;

int temp = 0;

for (  t = delta_t;  /*t <= time_lengH*/ ; t = (t + delta_t) )
{
  // cout<<t<<endl;


    for( int i = 1; i < N-1; i++)
    {

        //fout<<endl;
        //fout<<"\tI = "<<i<<endl;
        H_TEMP[i] = H[i] + (delta_t * G( H,i) ) ;

        if( H_TEMP[i] < 0 )
           {
            fout<<"\n\tError: H is less than 0"<<endl;
            cout<<"\n\tError: H is less than 0"<<endl;
            cout<<"\n\tT = "<<t<<endl;
            fout<<"\n\tT = "<<t<<endl;

            fout<<"\n\n\tTime = "<<t<<endl;
            for( int w = 1; w <=i ; w++)
            {
             fout<<"\tH["<<w<<"] =\t"<<H_TEMP[w]<<"\n";
            }


            cin.get();
            cin.get();
            return 0;
           }

    }   // end space for


    // set H( t+delta_t )  to new value
    for( int r = 1; r < N-1; r++)
        {
             H[r] = H_TEMP[r];
            // fout<<"\t"<<i<<"\t"<<H[i]<<"\n";
        }


    // print every second
    if (  ( (time_stamp + 1 - delta_t) < t) &&  ( (time_stamp + 1) >= t )  )
        {
            time_stamp = time_stamp + 1;
            cout<<" \tTime ="<<t<<endl;
            //cin.get();
            fout<<"\n\n# Time = "<<t<<endl;
            for ( int u = 0; u < N ; u++)
                fout<<"\t"<<u*delta_x<<"\t"<<H[u]<<endl;
        }

       // once we hit the wall we stop
       if( H[(N-2)] >= (wall*2) )
        break;



       temp++;
       if( temp == 10000 )
        {
          cout<<t<<endl;
          temp = 0;
         }

}   // end time for



     fout<<"#\n#\n#\tTime = "<<t<<endl;
     for( int w = 0; w < N; w++)
       {
         fout<<"\t"<<delta_x*w<<"\t"<<H[w]<<"\n";
       }


        fout.close();

        cout<<"\n\n\t!!!!!! Program Terminated !!!!!!!";
        cin.get();


        return 0;

}   // end main



//---------------------------------------------------------------------------
//---------------------------------------------------------------------------



// Set boundary conditions
void boundary()
    {

        H[0] = 1.0;
        H_TEMP[0] = 1.0;


        H[N-1] = precursor;
        H_TEMP[N-1] = precursor;

        return;

    }   // end boundary



//---------------------------------------------------------------------------
//---------------------------------------------------------------------------



// Set initial conditoins
void initial_condition()
    {

       int x, y, w;

       bool set_y = false;
       bool set_w = false;

       for ( x = 0; x < N ; x++)
        {

            if(   x*delta_x < 1)
                H_TEMP[x] = H[x] = 1.0;

            else if( x*delta_x >2 )
            {
                H_TEMP[x] = H[x] = precursor;
                if( !set_w )
                    {
                     w = x; // w is set to index where x*delta_x < 2
                     set_w = true;
                    }
            }
            else
                if( !set_y)
                {
                  y = x; // y is the first index that x*delta_x  > 1
                  set_y = true;
                }

        }   // end for


       long double delta = (1 - precursor) / ( w - y + 1);

       // this creates the linear initial condition from 1 to .1
       for ( int t = 1; y <=w ; y++, t++ )
           H[y] = 1 - delta*t;


       return;

    }   // end initial_condition



//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------



long double G( const long double H[],  int i)
    {
        long double X;

        X = -A(H,i);
        X = X + D()*B(H, i);
        X = X - C(H, i);

        if ( Log )
            fout<<"\t G["<<i<<"] = "<<X<<endl;
        return X;

    }   // end G



//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------



// 4'th order term
long double A( const long double H[], int i)
    {

        long double X, Y;
        long double denominator = powl( delta_x, 4);

        X = f(H, i, i+1)* L(H, i+1);
        Y = f(H, i-1, i)* L(H, i);

        if ( Log )
        {
            fout<<"\t A["<<i<<"] : X = "<<X<<endl;
            fout<<"\t A["<<i<<"] : Y = "<<Y<<endl;
            fout<<"\t Delta_X ^ 4: = "<<denominator<<endl;
        }



        X = X - Y;

        if ( Log )
            fout<<"\t A["<<i<<"]: X - Y = "<<X<<endl;

        X = X / denominator;

        if ( Log )
            fout<<"\t A["<<i<<"] = "<<X<<endl;

        return X;
    }   // end A 4'th order term



//----------------------------------------------------------------------------
//----------------------------------------------------------------------------



long double B( const long double H[], int i)
    {

        // Use central differences

        long double X;

//          General formula
//              X =  f(H, i, i+1) * ( H[i+2] - H[i] )
//                 - f(H, i-2, i-1) * ( H[i] - H[i-2]);

        // ghost point (i - 2 = -1) is equal to ( i + 2) which would be i
        if ( (i != 1) && ( i != (N-2) )  )
        {
            X =  f(H, i, i+1) * ( H[i+2] - H[i] );
            X =  X - f(H, i-2, i-1) * ( H[i] - H[i-2]);
        }
        // ghost point ( i + 2 = N) is equal to ( i - 2) which would be i
        else if ( i == (N-2) )
        {
            X =  f(H, i, i+1) * ( H[i] - H[i] );
            X =  X - f(H, i-2, i-1) * ( H[i] - H[i-2]);
         }
        else if ( i == 1)
        {
            X =  f(H, i, i+1) * ( H[i+2] - H[i] );
            X =  X - f(H, i, i-1) * ( H[i] - H[i]);
         }

        X = X / (4 * delta_x * delta_x);

        if ( Log )
            fout<<"\t B["<<i<<"] = "<<X<<endl;

        return X;
    }   // end B



//----------------------------------------------------------------------------
//----------------------------------------------------------------------------
```

```
long double C( const long double H[], int i)
    {

        long double X;


        X = powl(H[ i + 1], 3);
        X = X - powl( H[ i - 1 ], 3);

        X = X / ( 2*delta_x) ;

        if ( Log )
            fout<<"\t C["<<i<<"] = "<<X<<endl;

        return X;
    }   // end C




//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------



long double D()
    {
        return (long double)0.00;
    }   // end D




//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------



// 3'rd order term
long double L( const long double H[], int i)
    {

        // the possible values for i are  1 to N
        // conditions to test for are N, N-1, 1.

        long double result = 0;
        //long double temp = 0;

        if  ( (i > 1) && ( i < (N-1) ) )
         result = H[i+1] + 3*H[i-1] - 3*H[i] - H[i-2];

        // ghost point( i -2 = -1) is equal to i
        else if (i == 1 )
         result = H[i+1] + 3*H[i-1] - 3*H[i] - H[i];

        // ghost point ( i + 2 = N) is equal to N-2
        else if ( i == (N-1) )
            {
             result = H[N-2] + 3*H[i-1] - 3*H[i] - H[i-2];
            }

     //   temp = delta_x*delta_x*delta_x;

        result = result; // temp;

        if ( Log )
            fout<<"\t L["<<i<<"] = "<<result<<endl;

        return result;
    }   // end L, 'third order term



//----------------------------------------------------------------------------
//----------------------------------------------------------------------------



// h^3
long double f( const long double H[], int a, int b)
    {
        long double X;

        // boundary condiions
        if( a == -1 )
           a = 1;

        else if (  a == N)
           a = N-2;

        else if ( ( a < -1) || ( a > N) )
            {
             cout<<"\n\t Error:  F[a] = "<<a;
             cin.get();
            }


         // boundary condiions
        if( b == -1 )
           b = 1;

        else if (  b == N )
           b = N-2;

        else if ( ( b < -1) || ( b > N) )
            {
             cout<<"\n\t Error:  F[b] = "<<b;
             cin.get();
            }

        X =  powl( H[a],3 ) + powl( H[b],3 );
        X = X / 2 ;

        if ( Log )
            fout<<"\t f["<<a<<"]["<<b<<"] = "<<X<<endl;

        return X;

    }   // end h


//----------------------------------------------------------------------------
//----------------------------------------------------------------------------
```

delta_t = C*delta_x^4

C = 1/20

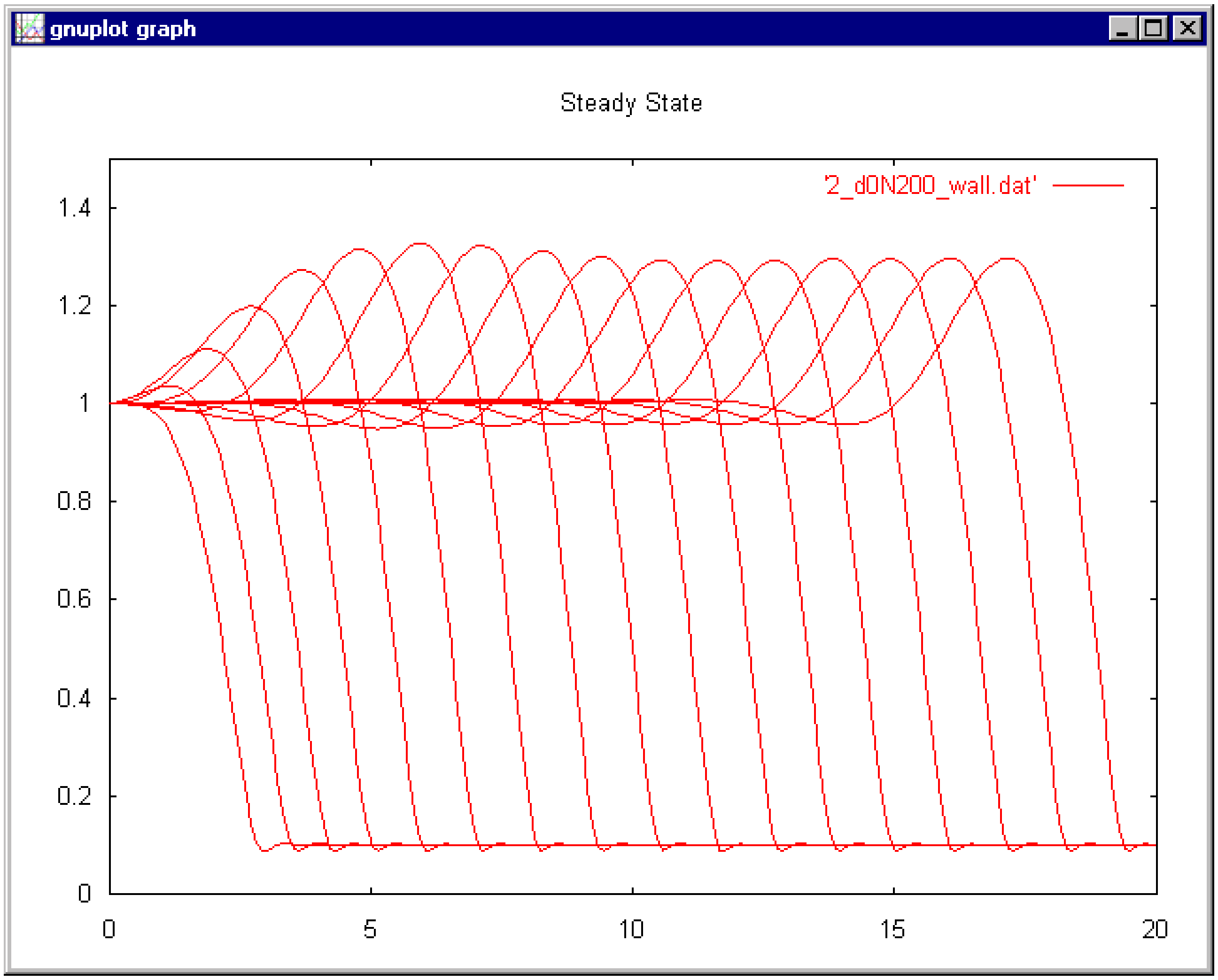

gnuplot graph
Initial Conditions T = 0
'1_ic_d0N50.dat'
'1_ic_d0N100.dat'
'1_ic_d0N200.dat'
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20

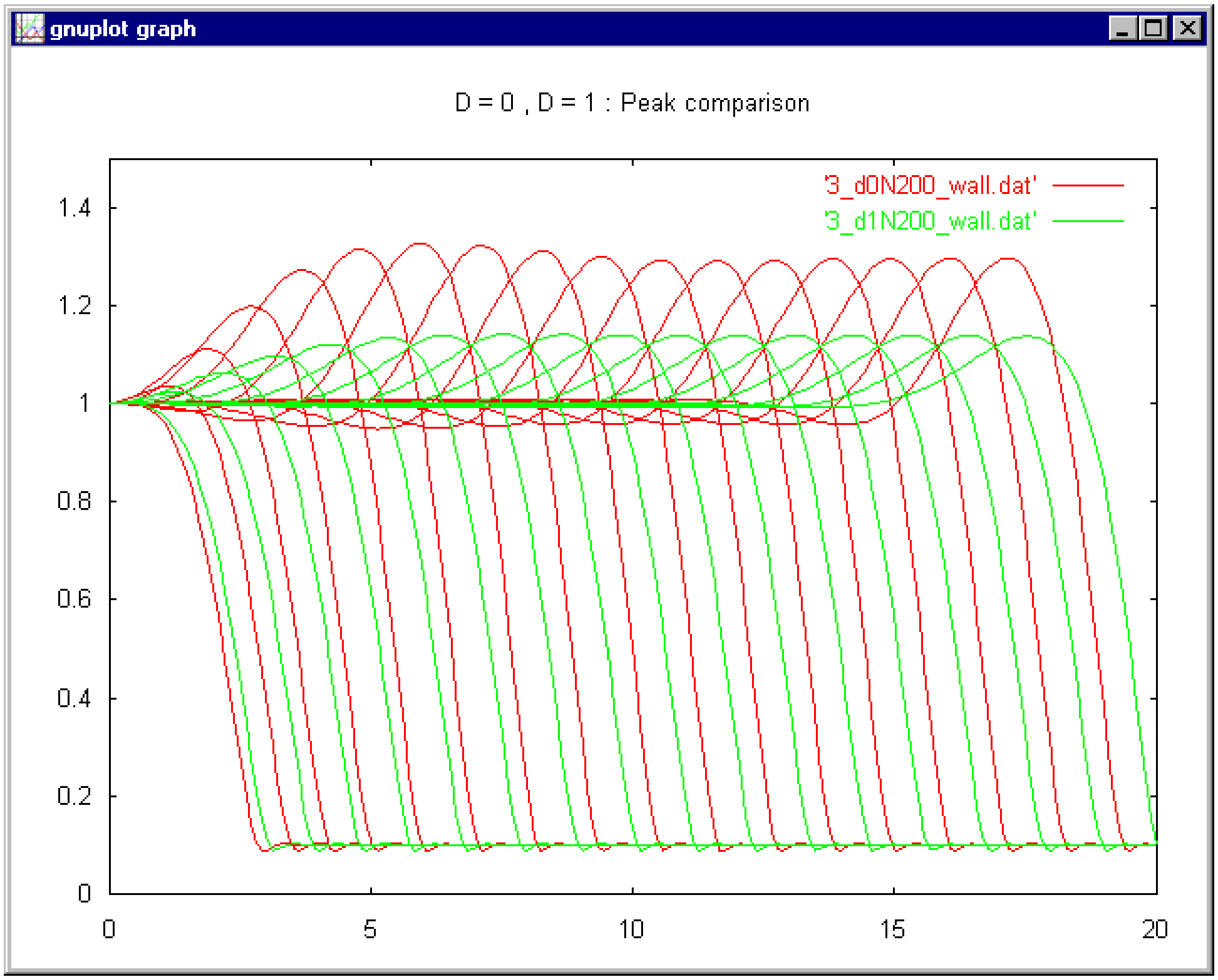


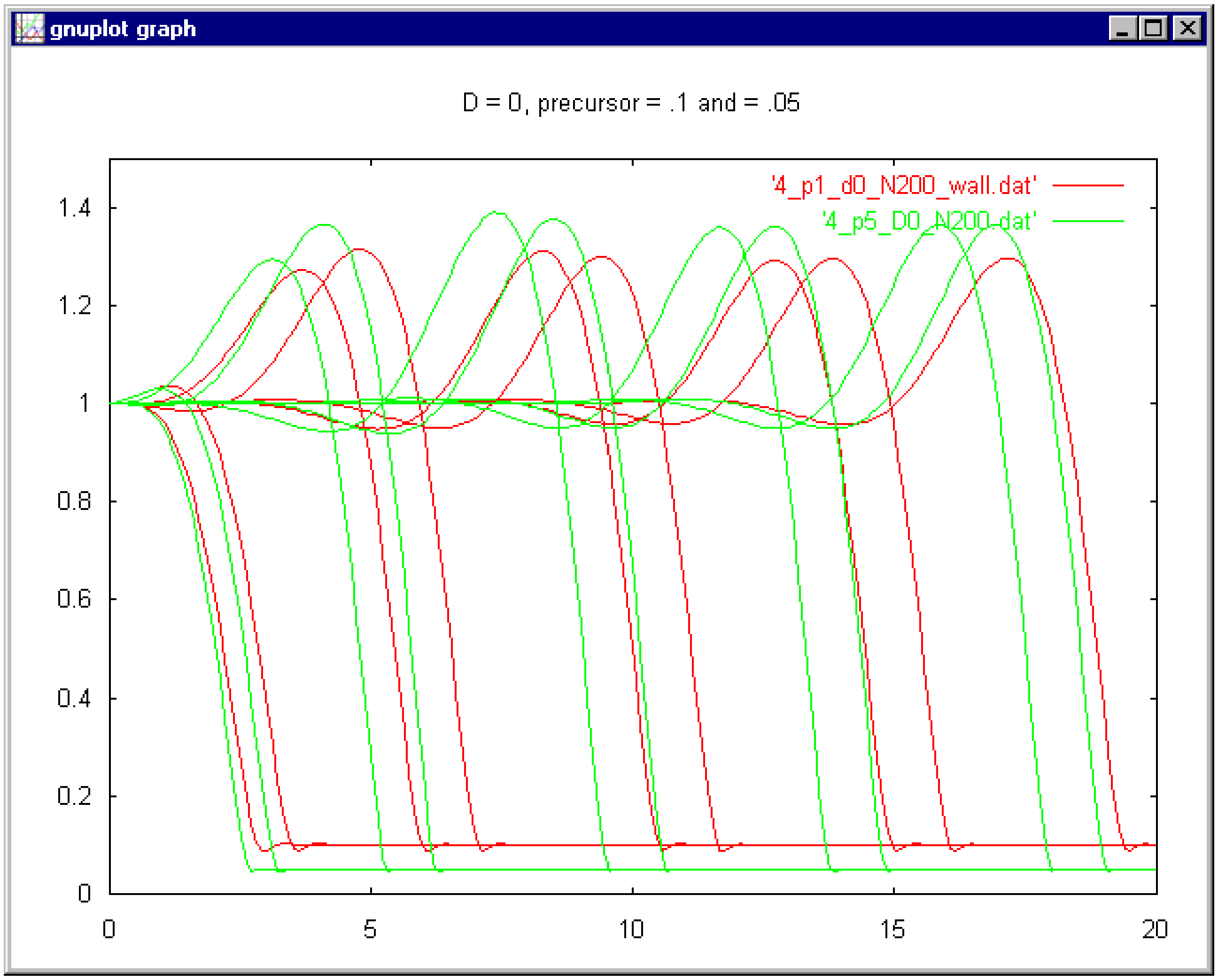
N = 50 Hmax = 1.29056720013955

N = 100 Hmax = 1.23512103035421

N = 200 Hmax = 1.20706260883653


gnuplot graph
Steady State
'2_d0N200_wall.dat'
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20


gnuplot graph
D = 0 , D = 1 : Peak comparison
'3_d0N200_wall.dat'
'3_d1N200_wall.dat'
0
0.2
0.4
0.6
0.8
1
1.2
1.4
0
5
10
15
20



Precursor = 0.1

- Velocity was found to be = 1.105521 units / s
- compared to 1.11 units / s predicted
- .4% error

Precursor = 0.05

- Velocity was found to be = 1.005025 units / s
- compared to 1.0525 units / s predicted
- 4% error

Silicone Oil Experiment:

PLATFORM

The objective of this experiment is to determine the behavior of a fluid flowing down an incline plane. In order to perform the experiment a platform needs to be built. After careful analysis of several similar experiments the size of the was determined to be 50 cm by 100 cm. These dimensions are sufficiently large to observe any triangle or finger formation. Before the platform will be built there are some requirements that need to be satisfied:

- Strong enough to sustain the weight of a glass sheet
- Ease in operation.
- Stable under all angle inclinations
- Able to level the platform anywhere

To ensure the strength of the platform a sheet of ¾ inch plywood was used with two thin sections at each end to eliminate any curvature from the sheet. Two identical sets were manufactured, one for the bottom and the other for the top where the glass will be placed. Both of these platforms are being attached with a double hinge at one end. In the lower platform in each corner a ½ inch holes were drilled and level screws mounted to ensure the vertical and horizontal stability. Having assembled the casing, the design of the rising mechanism is to follow. After several different approaches the one that best satisfies the range of inclination is the one shown in the following figure.

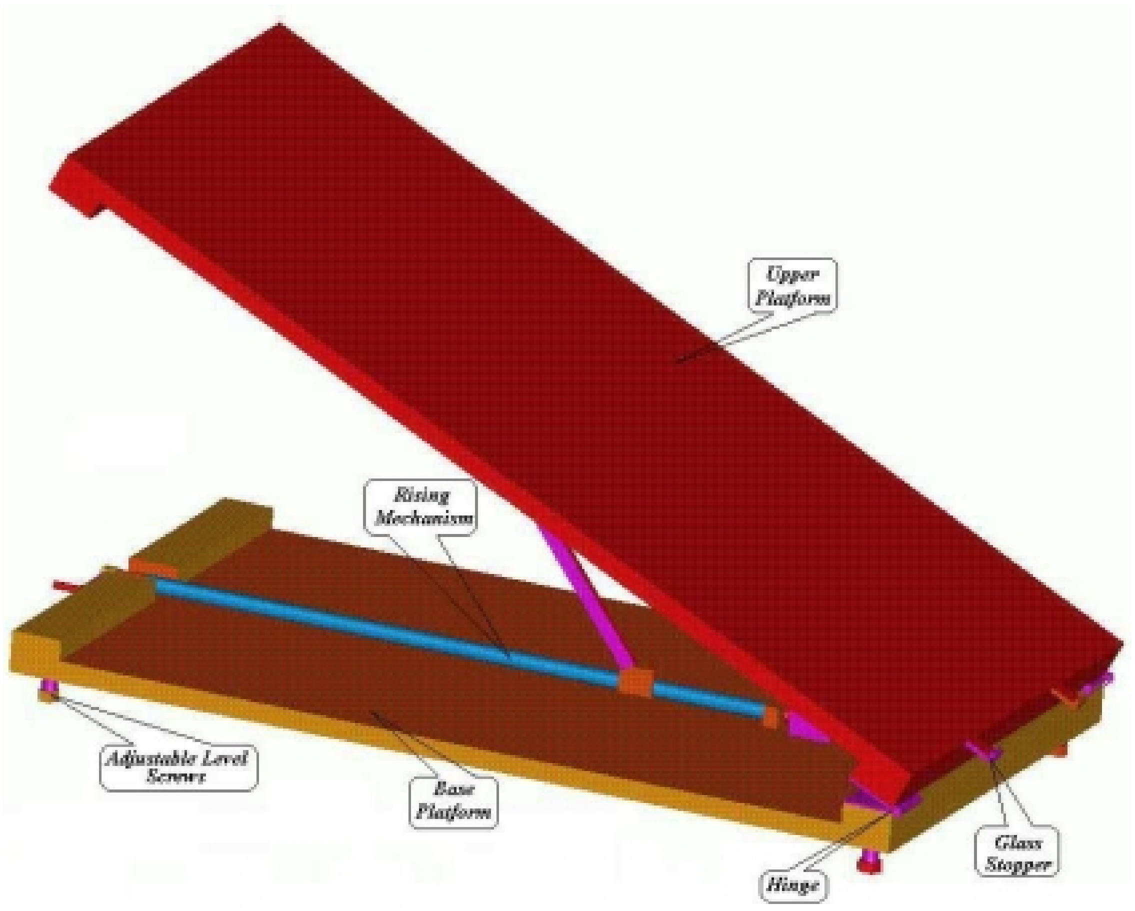


Figure #1 Solid Model of the Platform.

The threaded rod allows for fine adjustments to achieve any angle in the range of 0 to 90 degrees. This device was secured to the lower platform with several screws and a stopper at the end to counteract the thrust force. To ease the rising and lowering several wooden blocks have been attached to the back of the platform holding the glass, shown in figure #2.

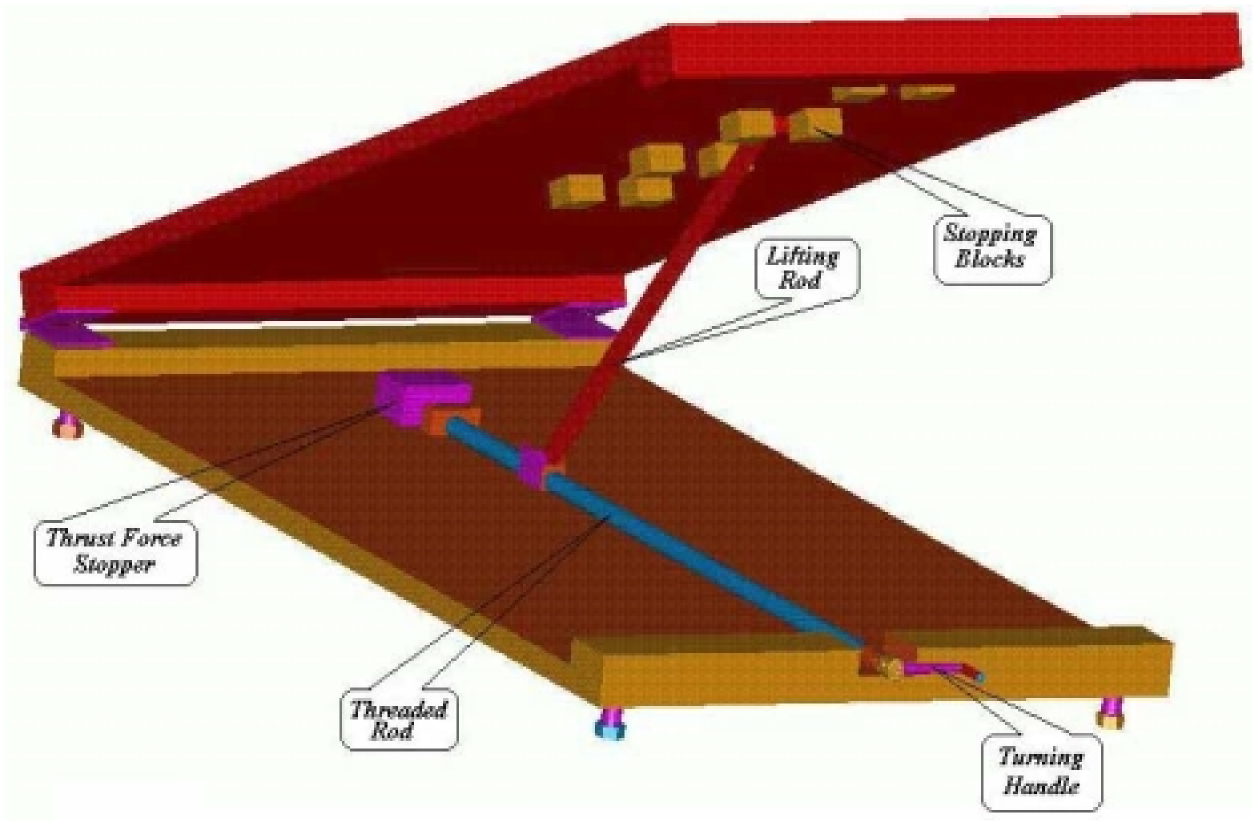


Figure #2   Model of Lifting Mechanism

For low angle a separate extension rod has been constructed to ensure stability of the platform. Also at the bottom of the inclined surface there will be attached two stoppers so that the glass will not slide during the experiment.
Finally, the platform is ready for testing. Using the adjustable screws and a level beam the apparatus will be setup in a designated area where the experiments will be performed. The last component of the apparatus is the release mechanism. It was especially difficult to arrive at the best technique of releasing the fluid properly. Fortunately, with the combined help our team the best device was produced. It was decided that a soft rubber with stiff handle would work. The only part that best fits the requirements is the doorstopper with rubber bottom. This doorstopper was then cut and bent into the following shape to ensure proper release of the fluid.

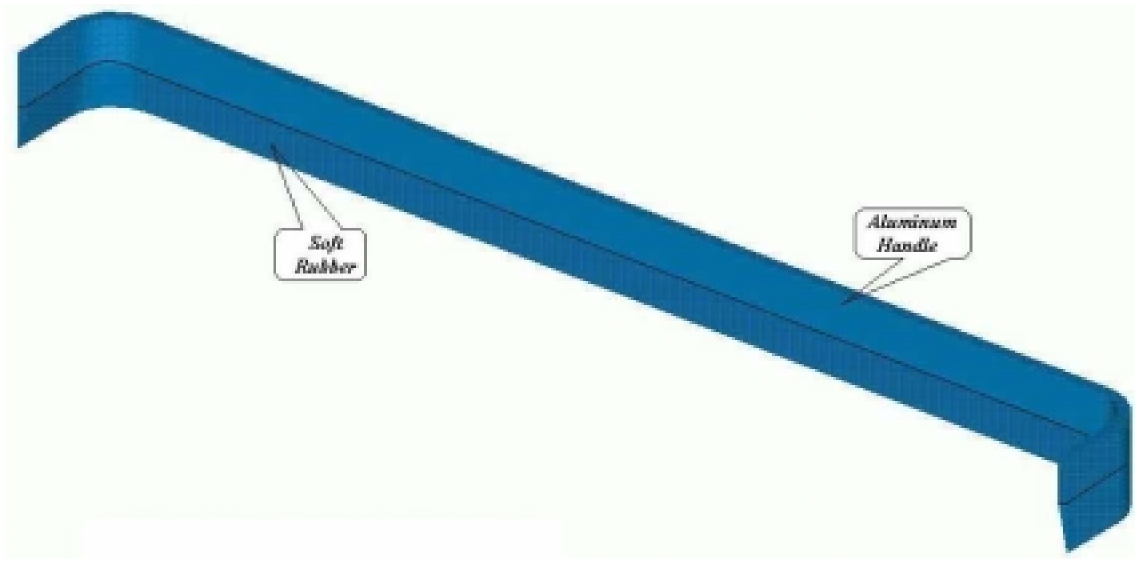


Figure #3 Release Mechanism

At this point all of the component are manufactured and several trials were conducted to ensure proper functionality of all components.

EXPERIMENTS

Before any experiment the glass was cleaned with soap to ensure proper conditions. Next a sheet of girded paper was placed underneath of the glass to allow measurement of the position of the fluid at any given time. To work efficiently a digital camera was used to record the motion of the fluid and then later data was be extracted for analysis. Several experiments were conducted at different inclinations, ~2.5, 30, 60, ~82 degrees. In our experiment the liquid used was Silicone Oil with following properties:

Property name	***Value***	***Units***
Kinematic Viscosity	$\upsilon = 0.5$	$\frac{cm^2}{sec}$
Density	$\rho = 0.96$	$\frac{gm}{cm^3}$
Surface Tension	$\gamma = 21$	$\frac{gm}{sec^2}$

The amount of oil used during experiments was 25 gm. For the small inclination 50 gm was used. To successfully perform an experiment one need top follow simple, yet critical steps:

- Clean the glass before each run with soap or cleaner
- Measure the amount of fluid to be used
- Adjust the platform to required inclination
- Position the camera so that the entire platform is viewed, for best results place the camera perpendicular to the platform to minimize distortion in extracted data
- Place the release bar at any given position and apply necessary pressure to eliminate any gaps between the rubber and the glass, critical step
- Pour the liquid behind the release bar and wait until liquid spreads uniformly
- Gently and evenly lift the bar, it is recommended to move the release bar backwards as it is lifted to eliminate any splashing. Do not allow dripping from the bar onto the glass after release, it may alter the results.

Several dozens of experiments were performed for angles ranging from 2.5 to 82 degrees. Due to limited resources and time the current release bar does not perform well for angles close to 90 degrees. To illustrate how the experiments were conducted a recording will be shown.

Experiment and Theory:

To be able to compare the experimental results to theoretical computations several transformations are needed, linearization and expansion in the limit of small 'q'. After explanation of linear stability and linearization analysis in detail in previous section, Theory, one needs to be bale to compare the theoretical results with the experimental. Since the all equations are in non-dimensional form a scale factors need to be computed. This is accomplished by using the following parameters in a relation to compute the appropriate scale factors.

distance scale

$$X_c = \frac{\sqrt{\frac{\gamma}{\rho \cdot g}}}{\sin\left(\frac{\pi}{180}\cdot\theta\right)^{\frac{1}{3}}}$$

velocity scale

$$V_c = \frac{\gamma}{3 \cdot \nu \cdot \rho}\cdot\sin\left(\frac{\pi}{180}\cdot\theta\right)$$

where:

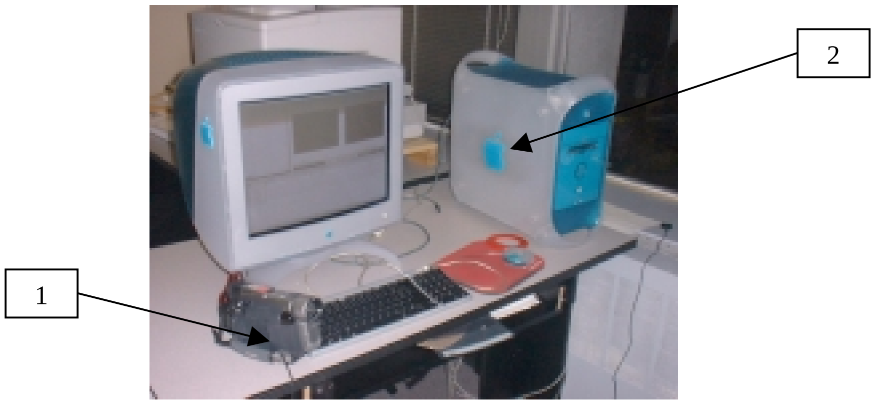
γ	*surface tension*
ρ	*density of fluid*
g	*gravitational constant*
ν	*kinematic viscosity*
θ	*inclination angle*

Data Collection and Image processing

In order to not worry about collecting data in real time we decided to video tape the experiments. The experiments were recorded on a digital video camera, which can later be connected to a computer to extract position and time information needed to compare with theoretical values derived by other group members.

Equipment used:

Hardware:



1) JVC Cybercam Digital video camera

2) Macintosh G3 with Fire Wire ports

3) Silicon Graphics Indy R5000

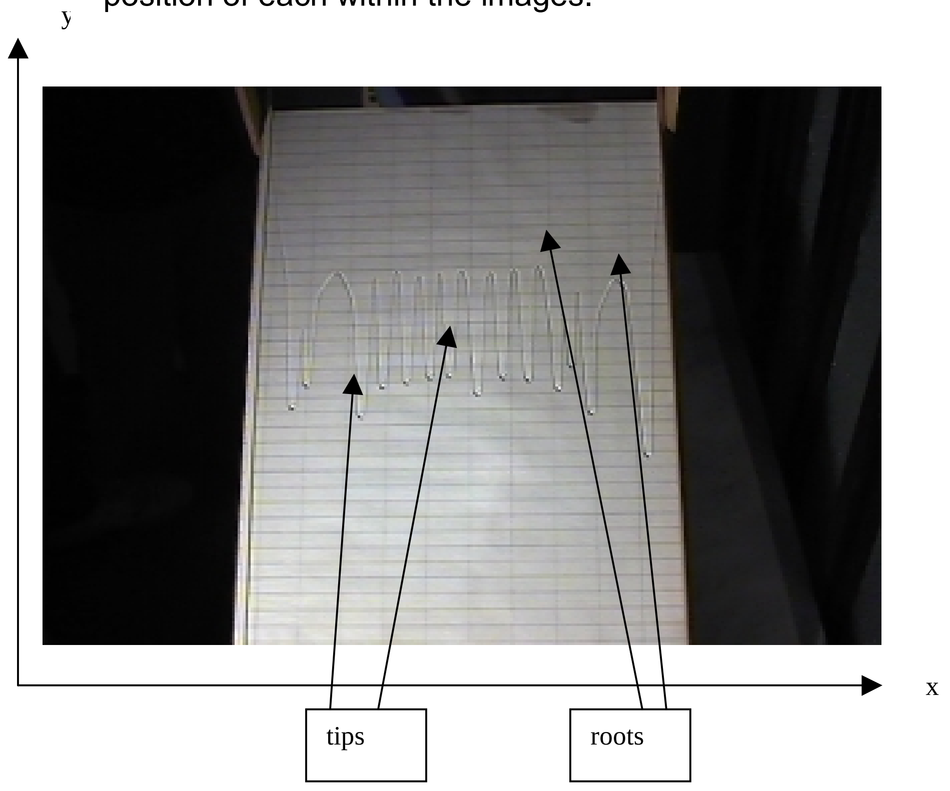
Software used:

1) Adobe Premier 5.1c

2) Matlab 5.1.0.421

Procedure:

1) Videotape the experiments and record the given parameters for the experiment (e.g. Angle of inclination).
2) Connect the Video camera to the Macintosh G3 using firewire cables and extract the video of the experiments on to the computer using the Adobe Premier software.
3) Extract individual frames from each experiment using the Premier software. Each frame was saved as a tiff image file for portability. Each file was given the time code as the name for processing it in the correct order later. Typically we extracted about 40~60 frames for each experiment.
4) Transfer all extracted frames to the Silicon Graphics stations, and use Matlab to open each frame and click on individual tips and roots to measure the x and y position of each within the images.



Matlab will then output each point as a pair of x and y coordinates into a text file also given the time code as its name. (***Note:*** *the x and y coordinate is not the same as derived in theory and numeric, x and y in this report are strictly in reference to the coordinates displayed on the figure above.*) This text file now has the raw data, which is corrected and calibrated by programs we created.

<u>Error sources:</u>

As with any experiment there are always sources of error. In data collection part of this lab there are minor details to consider. The major source of error is camera position. We created a relatively similar lighting environment for each experiment. We also placed the camera on a stable base with a level to make sure it is not slanted. In order to counter act parallax and other optical effects on the images, the camera was also kept as close to a straight view of the plane as possible. Also we created a grid system to place behind the glass as reference points. Taking no chance to believe that parallax has not affect the images we devised a scaling factor for each experiment. This factor is a correction factor for the y component, since that will have the largest variation and the motion we are interested in is the y coordinate in the image. Since we have a grid system in the image it self as reference. We know that each box is approximately 2 cm. Thus we take a y point near the top of the image and another y point at the bottom. We divide the pixel length by the number of centimeters between them and that becomes a scaling factor for the positions extracted from Matlab.

Transforming the Raw Data into Usable Data

In order to see how the data collected compared to the theory, the data had to be transformed many times. In this section, I will explain, for an experiment with 2n points (roots and tips) and m times, how raw data was arranged and how we transformed that into data we could analyze.

In a typical experiment with 2n points and m times, we had n roots and n tips. Associated with each root or tip were two pieces of data, the coordinate (x and y) and the time the data was collected. Therefore for a typical experiment, the final data would have 2n*m pieces of data.

The raw data was organized in the following way:

We had m files with 2n pieces of data. Each time t_i had associated with it a file. In other words, the data was organized by time. In order to understand the data, we had to organize the data in an understandable manner for each root and tip. That is to say that we would need n files for the roots, and n files for the tips, with each file containing m rows, each having one entry for the position, and one entry for the time.

The raw data contained m files. Each of these files was labeled with a timestamp. The timestamp is of the format %d-%d-%d, where %d is an integer. An example would help understand the naming convention for the files. For example, if a file was named 17-03-15, this means that the data contained in that file was collected on the 17 minutes, 3 seconds and 15 frames after the experiment was started. Each second is equivalent to 30 frames. Assuming 5 roots and 5 tips, the data within each file, the data was organized in the following manner:

x_{11}	x_{21}
x_{31}	x_{41}
x_{51}	x_{61}
x_{71}	x_{81}
x_{91}	x_{101}
y_{11}	y_{21}
y_{31}	y_{41}
y_{51}	y_{61}
y_{71}	y_{81}
y_{91}	y_{101}

The first thing that needed to done is to arrange the data in the following manner in each file.

x_{11}	y_{11}
x_{21}	y_{21}
x_{31}	y_{31}
x_{41}	y_{41}
x_{51}	y_{51}
x_{61}	y_{61}
x_{71}	y_{71}
x_{81}	y_{81}
x_{91}	y_{91}
x_{101}	y_{101}

This was done using rearrange.cpp (See code attached)

Once all of the files were in the format shown above, we got the time information from the data. This was done using the *ls –1* **|** *sort –n > time* command run in the directory with the raw data. What this does is it lists all of the files, sorts them, and then sends them into a file named *time*. At this point, the file time contains one column with m entries. The layout of this file is not unlike the table shown below (for m = 10):

12-02-00
12-02-10
12-02-20
12-03-00
12-03-10
12-03-20
12-04-00
12-04-10
12-04-20
12-05-00

From this, we generated a column of times (decimal values) using time.c, which is also attached. Since the time data was common to all of the files, we only needed to have one copy for the time being.

After gleaning the time information from the files, we conveniently renamed the files to make it easier to access them in an organized manner using a shell script named renamescript. The names of the files ranged from 1 to m since we had m files.

Once the files were renamed, it was time to scale the data. The data contained within the files was in the units of pixels. We converted it to standard units (cms) using scale.c which took as input the scaling factor. After the scaling factor, it was time arrange the data according to points (not times). This was done using a shell script called savescript. This script was probably the trickiest part of the data transformation process. This script opens up each file in the data after the previous transformation. It then puts each row i into the right file. The algorithm for this is the following (pseudo code)

```
for all the files in current directory
    {
      for each element in the file
            {
            do
            if(i equals 4n + 1)
            then i  = 1
            if i is even, output data element to file, then newline  //here the file is i – i/2
            if i is odd, output data element to file, then tab         //(integer division)
            }
      }
```

After this script, each file will have data associated with a particular tip or root.

x_{11}	y_{11}
x_{12}	y_{12}
x_{13}	y_{13}
x_{14}	y_{14}
x_{15}	y_{15}
x_{16}	y_{16}
x_{17}	y_{17}
x_{18}	y_{18}
x_{19}	y_{19}
x_{110}	y_{110}

The next thing that must be done to the data is combining the time values into the data. Since the y values don't really matter, since the x value is the one we care about, we can dispose of the y values. This is done by comb.c, which will strip each file of the y values. It will then add the time values into it, thus completing the transformation from the raw data into usable data.

t1	x_{11}
t_2	x_{12}
t_3	x_{13}
t_4	x_{14}
t_5	x_{15}
t_6	x_{16}
t_7	x_{17}
t_8	x_{18}
t_9	x_{19}
t_{10}	x_{110}

To summarize, the data transformation included the following:

Tranformation	Done by
Rearrange data so that each row represents a unique point.	rearrange.cpp
Get the time data from the file names	time.c
Rename the files conveniently	Renamescript
Scale the data for metric units	scale.c
Rearrange the data organized by points, not times	savescript
Combine the information about each root and tip with the time the data was collected	comb.c

I also used another script called runscript, which is also attached, to automate the process of running a particular program on a large number of files.

In conclusion the tools provided by the Unix operating system were very instrumental in making the data transformation process simple. It would be extremely hard to get the data in the right format without indispensable tools such as data redirection, pipelines, scripting and sorting the file as well as the man pages.

Flow and Instability of a Viscous Film down an Inclined Plane

Fluid flowing down an inclined plane commonly exhibits a fingering instability in which the contact line corrugates. It is believed that below a critical inclination angle the base state before the instability is linearly stable. Regardless of the long time linear stability of the front, microscopic scale perturbations at the contact line grow on a transient time scale sufficient enough to excite nonlinearities and thus initiate the finger formations. The results of a set of experiments to determine some of these features are presented in this report.

Our objective here is to study experimentally the nonlinear structures developing from the initial instability. In our experiments, all the parameters of the problem here held constant, except for the inclination angle, fluid viscosity, and fluid volumes. However, in the results provided, most of the information obtained was using the same fluid viscosity and fluid volume (unless noted otherwise).

In our experiments, a fixed volume of fluid was released at topmost portion of the plate. The fluid volume was fixed at 25 grams (unless otherwise noted), and the fluid of choice was silicone oil. The density of the silicone oil was 0.96 and the nominal kinematic viscosity was 50 cSt. The results for angles of decline of 5°, 30°, 60°, and 82° are reported here.

A glass plate that was 0.5 m wide and twice as long was mounted so that one end could be raised with respect to the other. A grid of paper was attached below the plate with marks made every 2 cm high and every 5 cm wide. These lines were viewed and photographed with a digital camcorder that was mounted approximately 90° to the plate. However, distortion can be seen when analyzing the video captures. The time stamp of each video capture labeled the name of each file taken.

Prior to starting an experiment, the glass surface was cleaned in a consistent sequence with paper toweling and a cleaning solvent made by the same manufacturer as the silicone oil. The angle of inclination was measured and set to the previously mentioned values, and checks were made with a level to ensure that the table was not tilted in the lateral direction. The oil was placed behind an unattached gating system, sealed at each end, and allowed to settle to an even film before time of release. After it was ascertained that the fluid was level, the gating system was lifted by hand, allowing the contents to spill out onto the plate toward the upper end. In this manner, the oil volume was uniformly distributed across the plate without having any initial velocity. While this method of release worked satisfactorily, it may have been less free of disturbances than those used in previous experiments. In this regard, it may have affected the fingers that were observed and the time for their initial appearance.

The location of the roots and tips were obtained from the video captures stored in a computer for later analysis using Matlab software. About 20 roots or tips were observed in each run, however only the middle 10-15 roots and tips were used to obtain the location and time histories.

I will now present the results of our fully nonlinear time-dependent simulations of a thin liquid film flowing down an inclined plane. In all of our experiments, we were considered the simplest form of fluid flow – the flow of a thin film of a completely wetting fluid down an inclined plane. By completely wetting fluid, we're assuming that the contact angle (where the front of the fluid hits the surface) is virtually non-existent or zero.

The basic picture behind the experiments is that the contact line of the fluid against the surface becomes unstable with respect to the transverse perturbations. It is believed that this instability is related to both the inclination angle and the contact angle. Since we're ignoring contact angle in our experiments, we will observe the behavior of the fluid using different angles of inclination.

Assuming complete wetting, we found that varying the inclination angle modifies the shape of the emerging patterns (i.e. fingers versus triangles). The inclination angle strongly affects the shapes of the emerging patterns: large inclination angles lead to finger-like tips while smaller angles produced the triangular shaped patterns.

First, we looked at the wavelengths between the fingers/tips. Table 1 shows that the experimental results basically agree with the theoretical results. The observed separation between the tips of the patterns is within a reasonable accuracy with the wavelength of the mode of maximum growth, λ_m.

Next, we observed the growth rates of our experiments for long times. Figures 1 and 2 show the results of our experiments for simulations done at both 30° and 60°, respectively. These figures show the "linear" relationship between the ln-ln plot of distances of the fingers/tips versus time.

Measurements of the growth rates for long times of these structures have been reported and the results were as follows:

(1) Huppert reported that the locations of the extreme positions of either type of disturbance were proportional to (time)^{q} for some exponent *q;* for tips of the fingers the exponent was 0.6, while the roots (where neighboring fingers join) was virtually stationary.

(2) Jerrett and de Bruyn observed fingers only in their experiments and reported that the average exponent for the tips of the fingers was 0.65 for glycerine (kinematic viscosity 110 cSt) and 0.52 for a mineral oil (kinematic viscosity 15 cSt)

Figures 3 and 4 show how the growth rate of the fingers/tips grow exponentially as time increases for simulations done at both 30° and 60°, respectively.

Next, we observed the growth rates of our experiments for early times. Figures 5 and 6 show the linear stability analysis of figures 3 and 4 for the overall average of distances of fingers/tips. The results shown are for simulations done at 30°, 60°, and 82°, respectively.

It is very important to note that the linear stability analysis is limited to very early times, so we expect some error in the direct comparison of experimental and theoretical results. Linear stability analysis applies only to short times and cannot predict the behavior of the system when the perturbations become large. At this point is where the nonlinear simulations are the only means of linking experiments with theory.

The straight-line graphs for all experiments show that the tip and root locations *L* and the time *t* are related by laws of the form $L \sim e^{\beta t}$, for some positive exponent $\beta * t$. The computer was used to obtain a least-squares estimate of the values of these exponents. Table 2 shows the growth rates β, defined by $L(t)/L_0=\exp(\beta t)$ [where $L(t)=x_f(t) - x_t(t)$].

For early times, L(t) increases exponentially with a growth rate close to the one given by linear stability analysis, $\beta \approx 0.24$. For later times, L increases linearly consistent with the prediction that there is a transition from exponential to linear increase of pattern length, which can be seen in figures 3 and 4.

Table 1. Wavelengths in cm

ANGLE α	VOLUME, G	KINEMATIC VISCOSITY, CC	WAVELENGTH λ, (THEORETICAL) CM	WAVELENGTH λ, (EXPERIMENTAL) CM
30°	25	50	3.69	2.74
60°	25	50	2.29	2.49
82°	25	50	1.95	1.73

Table 2. Growth Rates of Fingers/Tips

ANGLE α	VOLUME, G	KINEMATIC VISCOSITY, CC	NON-DIMENSIONAL TIME T_C	LINEAR STABILITY SLOPE	EXPONENT β (THEORY)	EXPONENT β (EXPERIMENT)
30°,tip	25	50	0.025815	6.0115	$0.2 \le \beta \le 0.6$	0.1552
60°,tip	25	50	0.012411	16.463	$0.2 \le \beta \le 0.6$	0.2043
82°,tip	25	50	0.01038	24.902	$0.2 \le \beta \le 0.6$	0.2585

Figure 1.

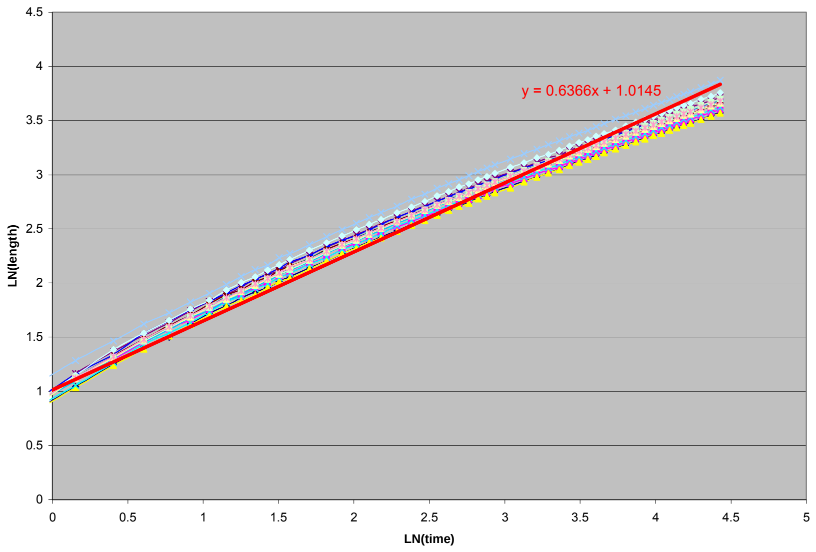


Figure 1 shows a plot of the natural logarithm of the distances of tips X, down a glass plate for disturbances produced when 50 cSt silicone oil flowed at 30° to the horizontal direction, as a function of the natural logarithm of time T.

Figure 2.

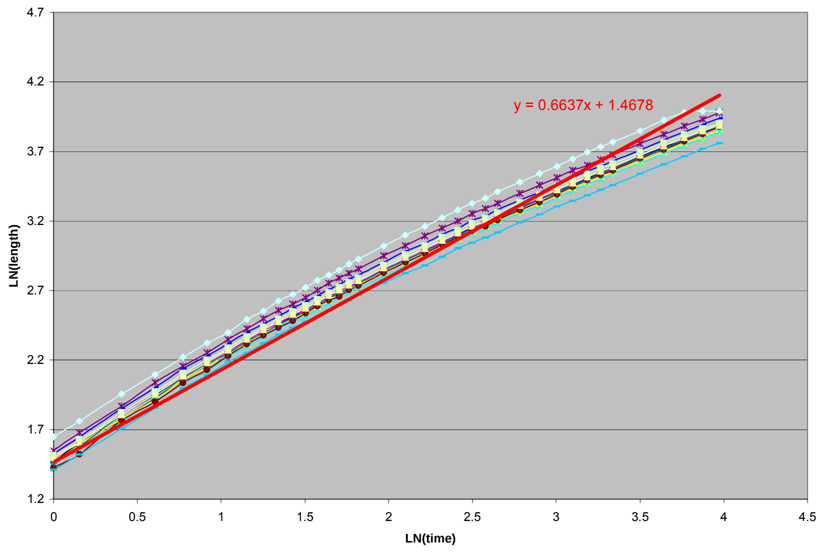


Figure 2 shows a plot of the natural logarithm of the distances of tips X, down a glass plate for disturbances produced when 50 cSt silicone oil flowed at 60° to the horizontal direction, as a function of the natural logarithm of time T.

Figure 3.

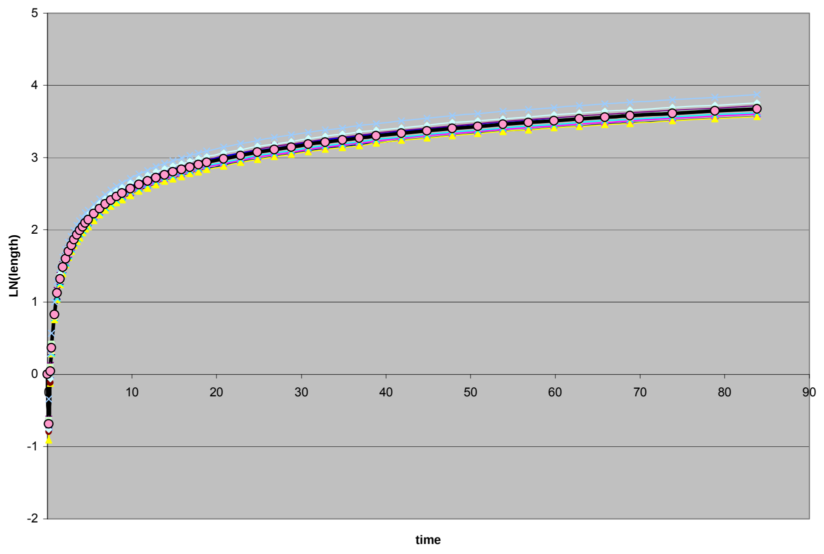


Figure 3 shows a plot of the natural logarithm of the distances of tips X, down a glass plate for disturbances produced when 50 cSt silicone oil flowed at 30° to the horizontal direction, as a function of time T.

Figure 4.

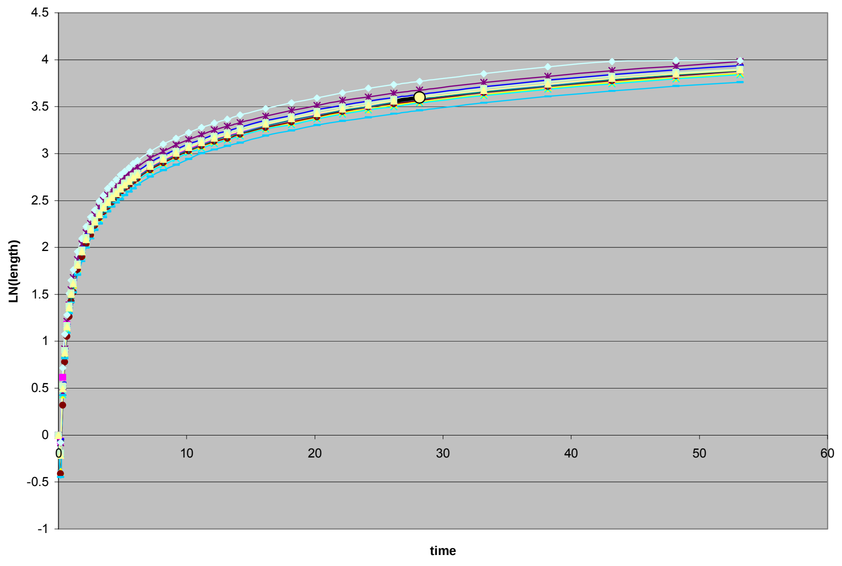


Figure 4 shows a plot of the natural logarithm of the distances of tips X, down a glass plate for disturbances produced when 50 cSt silicone oil flowed at 60° to the horizontal direction, as a function of time T.

Figure 5.

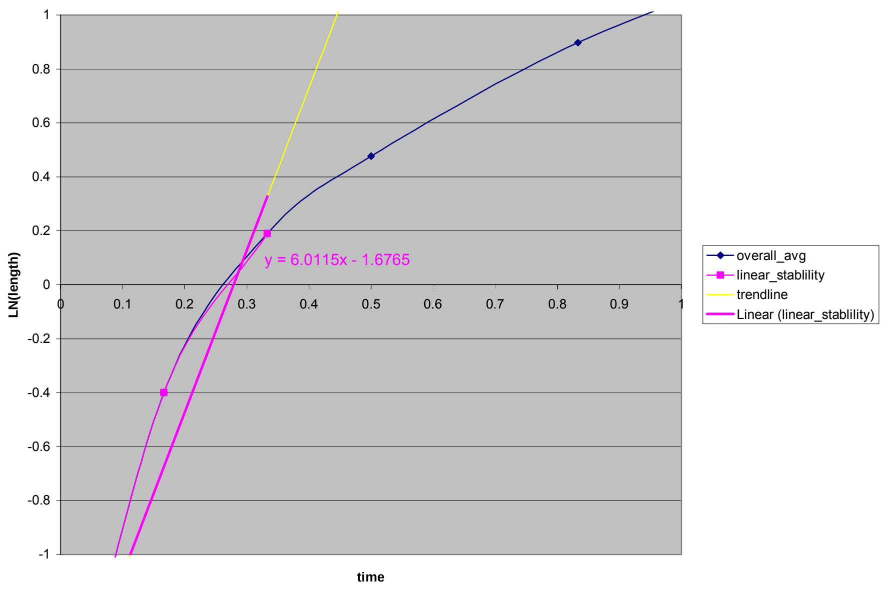


Figure 5 shows a zoom plot of the natural logarithm of the overall average of distances of tips X, down a glass plate for disturbances produced when 50 cSt silicone oil flowed at 30° to the horizontal direction, as a function of time T. Added onto the graph is a fitted trendline corresponding to the linear stability analysis with its proper equation.

Figure 6.

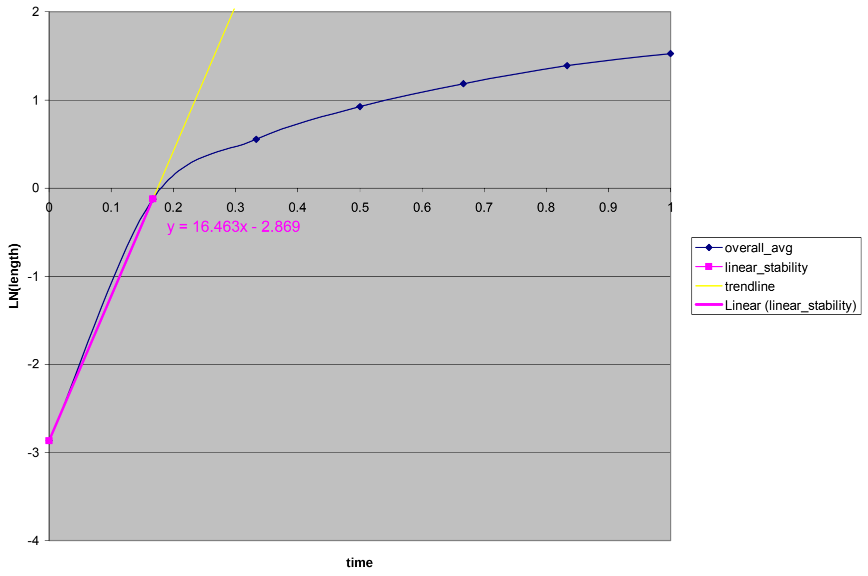


Figure 6 shows a zoom plot of the natural logarithm of the overall average of distances of tips X, down a glass plate for disturbances produced when 50 cSt silicone oil flowed at 60° to the horizontal direction, as a function of time T. Added onto the graph is a fitted trendline corresponding to the linear stability analysis with its proper equation.

Figure 7.

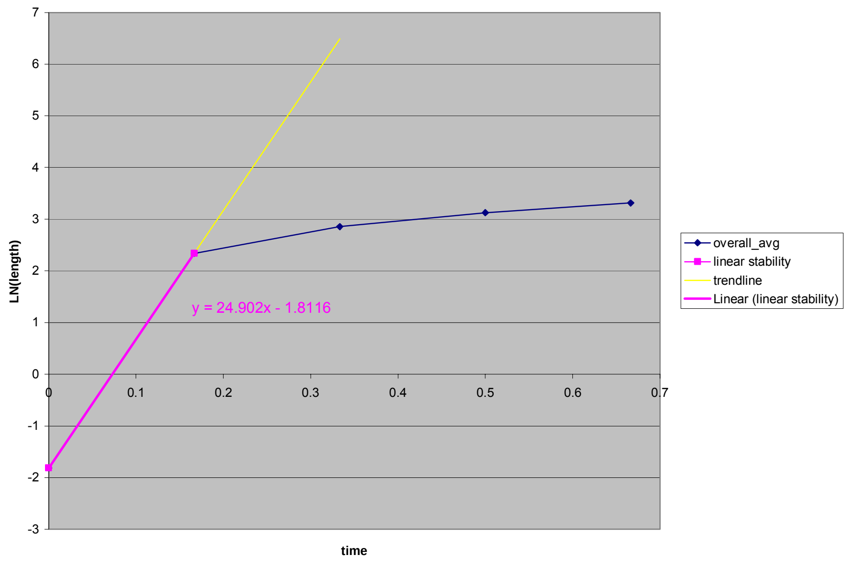


Figure 7 shows a zoom plot of the natural logarithm of the overall average of distances of tips X, down a glass plate for disturbances produced when 50 cSt silicone oil flowed at 82° to the horizontal direction, as a function of time T. Added onto the graph is a fitted trendline corresponding to the linear stability analysis with its proper equation.

Conclusions:

There is no evidence behind the understanding of the nonlinear development of the instability. However, possible effects that may cause instability include microscopic effects, such as the smoothness of the surface of the plane and the presence of contaminants.

The accuracy of our results is based solely on the consistency of our means of data collection from the video captures. Possible effects that may cause error in our data analysis include camera angle distortion and precise "clicking" in the Matlab analysis.

Time also played a significant role in the accuracy of our results since we were only able to analyze a handful of experiments. The analysis of a single experiment took between 4-6 hours on average. If we were able to analyze at least 3-4 experiments using the same parameters, our results would be a lot more concrete.

References

[1] D. J. Acheson. Elementary fluid mechanics, Clarendon Press, Oxford, 1990.

[2] S. M Troian, E. Herbolzheimer, S. A. Safran, and J. F. Joanny, *Europhys. Lett* **10** (1), 25- (1989).

[3] A. L. Bertozzi and M. P. Brenner, *Phys. Fluids* **9** (3), 530- (1997).

[4] D. E. Kataoka and S. M. Troian, *Nature* **402**, 794- (1999).

[5] H. E. Huppert, *Nature* **300**, 427- (1982).

[6] F. Melo, J. F. Joanny, and S. Fauve, *Phys. Rev. Lett.* ***63*** *(18), 1958- (1989).*