Numerical Modelling of the Dynamical Evolution of Contact Lines in Fluid Flows

Chengzhu Xu
Supervisor: Dmitry Pelinovsky

April 20, 2013

Abstract

This work is concerned with the recent model proposed by Benilov and Vynnycky in [1], who examined the behavior of contact lines with a 180° contact angle, in the context of two-dimensional Couette flows. The model is given by a fourth order linear advection-diffusion equation, with an unknown speed to be determined dynamically from an additional boundary condition at the contact line.

The main claim of [1] is that for any suitable initial condition, there is a finite positive time at which the speed of the contact line becomes infinite within this model, while the profile of the fluid flow remains regular. Additionally, it is claimed that the speed near the blow-up behaves as the logarithmic function of time.

This paper will present numerical approximations of solutions to the reduced model from [1], and simulate the blow-up behavior under different initial conditions. We will confirm the first claim of [1] but show that the blow-up is better approximated by a power function, compared with the logarithmic function of time.

1 Introduction

Contact lines are defined by the triple-point intersection of the rigid boundary, fluid flow and the vacuum state. Flows with the contact line at 180° contact angle were discussed in [2, 3], where corresponding solutions of the Navier-Stokes equations were shown to have no physical meanings. The recent approach proposed by Benilov and Vynnycky in [1] is based on the lubrication approximation and thin film equations. In particular, the authors of [1] derived a reduced model, which will be studied in this paper.

Consider the Couette flow described in figure 1, where two horizontal rigid plates are separated by a distance H, with the upper plate moving to the left with a velocity U_1, and the lower plate moving to the right with a velocity U_2. The space between the plates is filled with an incompressible liquid on the left, and vacuum (that is, gas with negligible density) on the right, separated by a free boundary. When the Couette flow is two-dimensional, the x-axis is directed along the lower plate, and the contact line is located on the upper plate. We assume that the liquid-filled region to the right of the contact line decays monotonically, and is carried away by the lower plate to some non-negative constant as $x \to \infty$.

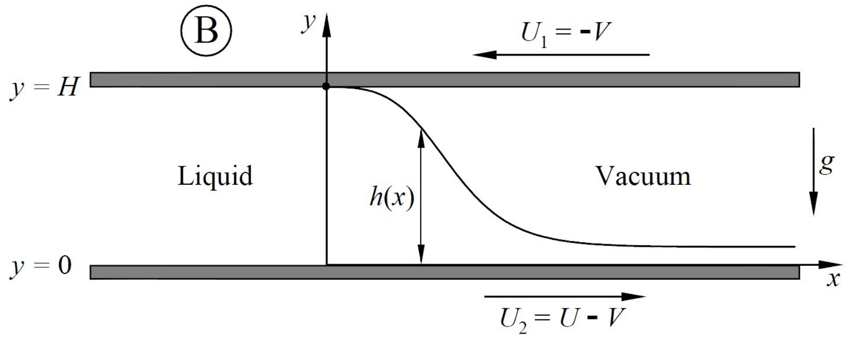

Figure 1: Couette flows with a free boundary, in the reference frame co-moving with the contact line.

In our reference frame, the position of the contact line is fixed at the point $x = 0$, by assuming that the velocity of the upper plate U_1 matches that of the contact line $V(t)$, i.e.

$$U_1 = -V(t), \qquad U_2 = U - V(t),$$

where U is a constant velocity of the lower plate relative to the upper plate. Note that U is given, whereas $V(t)$ is an unknown variable to be found for different time t. The shape of the liquid-vacuum interface at time t is described by the graph of the function $y = h(x,t)$ for $x > 0$, where h is the thickness of liquid-filled region.

1.1 The Reduced Model

The reduced model for the interface $h(x,t)$ derived by Benilov and Vynnycky in [1] is given by the linear advection-diffusion equation

$$\frac{\partial h}{\partial t} + \frac{\partial^4 h}{\partial x^4} = V(t)\frac{\partial h}{\partial x}, \quad x > 0, \quad t > 0, \tag{1}$$

subject to some suitable initial condition $h|_{t=0} = h_0(x)$ for $x \geq 0$, and the boundary conditions at $x = 0$:

$$h|_{x=0} = 1, \quad \left.\frac{\partial h}{\partial x}\right|_{x=0} = 0, \quad \left.\frac{\partial^3 h}{\partial x^3}\right|_{x=0} = -\frac{1}{2}, \quad t \geq 0. \tag{2}$$

The first boundary condition is used to determine the location of the contact line, with the assumption $H = 1$ in Figure 1 for simplicity. The second boundary condition is determined by the $180°$ contact angle at the contact line. The third boundary condition is used for normalization of the flux.

As we mentioned before, the profile of $h(x, t)$ decays monotonically to some non-negative constant as $x \to \infty$, so that $x = 0$ is a non-degenerate maximum of h, i.e. $h_{xx}|_{x=0} < 0$ for all $t \geq 0$. If the solution $h(x, t)$ loses monotonicity in x during the dynamical evolution, for instance, due to the value of $h_{xx}|_{x=0}$ crossing zero from the negative side, then the flow becomes non-physical, and we say that the reduced model blows up.

The main claim of [1] is that for any suitable initial condition, there is a finite positive time t_0 such that as $t \to t_0$, the second derivative $h_{xx}|_{x=0}$ approaches zero, and $V(t)$ diverges to negative infinity. Moreover, it is claimed that $V(t)$ behaves near the blow-up time as the logarithmic function of t:

$$V(t) \sim C_1 \log(t_0 - t) + C_2, \quad \text{as} \quad t \to t_0, \tag{3}$$

where C_1, C_2 are positive constants.

1.2 Main Results

Several attempts have been made to solve the problem analytically and prove the above facts rigorously. Explicit solutions in the case of constant velocity $V(t) = V_0$ using an application of Laplace transform were analysed in [4], and local solutions of the original problem were shown to exist by using this method as a series in powers of $t^{1/4}$. Various sufficient conditions for existence of global solutions were derived in [5], and self-similar solutions for finite-time singularities were studied. Nevertheless, results of [4, 5] did not provide a proof of the finite-time blow-up nor the construction of blow-up rates.

In this paper we will simulate numerically the behavior of the velocity $V(t)$ near the blow-up time under different initial conditions. In particular, the conjecture in [1] that all initial velocities will result in blow-up of $V(t)$ in finite time is confirmed by our numerical results. However, the power function

$$|V(t)| \sim \frac{c}{(t_0 - t)^p}, \quad \text{as} \quad t \to t_0, \tag{4}$$

with $c > 0$ and $p \approx 0.4$ is found to fit our numerical data better than the logarithmic function (3) near the blow-up time t_0.

2 Reformulation of the Model

In what follows, we shall set $u := \partial h/\partial x$, and reformulate the reduced model (1) - (2) in the equivalent form. By differentiation of Equation (1) with respect to x, we obtain

$$\frac{\partial u}{\partial t} + \frac{\partial^4 u}{\partial x^4} = V(t)\frac{\partial u}{\partial x}, \quad x > 0, \quad t > 0. \tag{5}$$

We also rewrite boundary conditions in (2) as

$$u|_{x=0} = 0, \quad \left.\frac{\partial^2 u}{\partial x^2}\right|_{x=0} = -\frac{1}{2}, \quad \left.\frac{\partial^3 u}{\partial x^3}\right|_{x=0} = 0, \quad t \geq 0. \tag{6}$$

Here the third boundary condition $u_{xxx}|_{x=0} = h_{xxxx}|_{x=0} = 0$ follows from applying the boundary conditions $h|_{x=0} = 1$ and $h_x|_{x=0} = 0$ to Equation (1) as $x \to 0$. After the reformulation, the value of $V(t)$ can be determined by the limit $x \to 0$ in (5):

$$u_{xxxx}(0,t) = V(t)u_x(0,t), \quad t \geq 0, \tag{7}$$

provided that the solution u is smooth at the boundary $x = 0$.

A suitable initial condition $u|_{t=0} = u_0(x), x \geq 0$ for the system (5) - (6) can be chosen in the form

$$u_0(x) = -\frac{1}{4}e^{-ax}x[4 + (4a+1)x + a(2a+1)x^2 + bx^3], \tag{8}$$

where parameters $a > 0$ and $b \geq 0$ are arbitrary. For simplicity, the constraint $h_{xx}|_{x=0} = u_x|_{x=0} < 0$ is standardized to $u_x|_{x=0} = -1$ at time $t = 0$. Note that initial conditions in this form also satisfy the decay constrains (that is, $u(x,t) < 0$ for all $x > 0, t \geq 0$, and $\partial^n u/\partial x^n \to 0$ as $x \to \infty$ for all $n = 0, 1, 2, \ldots$). Figure 2 shows a particular example of initial condition with $a = 0.5, b = 0$:

$$u_0(x) = -\frac{1}{4}e^{-x/2}x(4 + 3x + x^2). \tag{9}$$

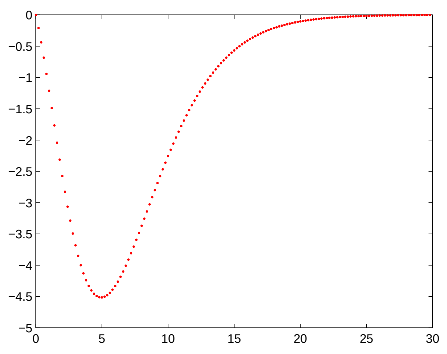

Figure 2: Plot of the initial function (9) for the reformulated model.

3 Numerical Solutions: Finite Difference Method

In this section, the solution over x-axis will be approximated by the second order central difference method. Consider a set of $N+2$ equally spaced grid points $\{x_n\}_{n=0}^{N+1}$ on the interval $[0, L]$, for L sufficiently large so that $u|_{x=L}$ is approximately zero. We assume ordering:

$$0 = x_0 < x_1 < \cdots < x_N < x_{N+1} = L.$$

For any fixed $t > 0$, let $u_n(t)$ denote the numerical approximation of $u(x,t)$ at $x = x_n$, and let Δx denote the equal step size between adjacent grid points. By applying central difference formulas (see Sections 6.1 and 6.2 in [6], for example) to Equation (5) at each $x = x_n$, we have

$$\frac{du_n}{dt} := V(t)\frac{u_{n+1} - u_{n-1}}{2(\Delta x)} - \frac{u_{n+2} - 4u_{n+1} + 6u_n - 4u_{n-1} + u_{n-2}}{(\Delta x)^4} + O(\Delta x^2). \tag{10}$$

Note that since $u_0 = u(0,t) = 0$ and $u_{N+1} = u(L,t) = 0$ for all $t \geq 0$, the above formula need only to be applied to N interior points $\{x_1, x_2, \ldots, x_N\}$, and we need to approximate u_{-1} for grid point x_1 and u_{N+2} for grid point x_N. The values of u_{-1} can be found from the boundary condition:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=0} = -\frac{1}{2} \quad \Rightarrow \quad \frac{u_{-1} - 2u_0 + u_1}{(\Delta x)^2} \approx -\frac{1}{2} \quad \Rightarrow \quad u_{-1} = -u_1 - \frac{1}{2}(\Delta x)^2,$$

and u_{N+2} can be found from the decay condition:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x\to\infty} = 0 \quad \Rightarrow \quad \frac{u_N - 2u_{N+1} + u_{N+2}}{(\Delta x)^2} \approx 0 \quad \Rightarrow \quad u_{N+2} = -u_N.$$

The system of N ODEs defined by (10) can be written in the matrix form

$$\frac{d\boldsymbol{u}}{dt} = \boldsymbol{A}\boldsymbol{u} + \boldsymbol{b}, \tag{11}$$

where $\boldsymbol{u}$ is the $N \times 1$ column vector of $\{u_1, u_2, \ldots, u_N\}$, $\boldsymbol{A}$ is the $N \times N$ matrix whose $(i,j)^{\text{th}}$ entry is given by

$$a_{i,j} = \begin{cases} -\dfrac{6}{(\Delta x)^4}, & j = i \\ \dfrac{4}{(\Delta x)^4} \pm \dfrac{V(t)}{2(\Delta x)}, & j = i \pm 1 \\ -\dfrac{1}{(\Delta x)^4}, & j = i \pm 2 \\ 0, & \text{otherwise} \end{cases}$$

except that $a_{1,1} = a_{N,N} = -\dfrac{5}{(\Delta x)^4}$, and $\boldsymbol{b}$ is the $N \times 1$ column vector with $\dfrac{1}{2(\Delta x)^2}$ in the first entry and zeros in all other entries.

The velocity $V(t)$ can be expressed by applying the same numerical method to (7):

$$\begin{aligned} V(t) &= \frac{u_{xxxx}|_{x=0}}{u_x|_{x=0}} \\ &\approx \frac{2(u_2 - 4u_1 + 6u_0 - 4u_{-1} + u_{-2})}{(\Delta x)^3(u_1 - u_{-1})}, \end{aligned} \tag{12}$$

where u_{-2} can be found from the third boundary condition in (6):

$$\left.\frac{\partial^3 u}{\partial x^3}\right|_{x=0} = 0 \quad \Rightarrow \quad \frac{u_2 - 2u_1 + 2u_{-1} - u_{-2}}{(\Delta x)^3} \approx 0 \quad \Rightarrow \quad u_{-2} = u_2 - 4u_1 - (\Delta x)^2.$$

We shall use Heun's method to evaluate the system of ODEs in (11). Let $\boldsymbol{u}_k$ denote the numerical approximation of $\boldsymbol{u}(t)$ at $t = t_k$ for $k = 0, 1, 2, \ldots$, and let Δt denote the time stepsize, then the Heun's method states that

$$\boldsymbol{u}(t_{k+1}) \approx \boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \frac{\Delta t}{2}[(\boldsymbol{A}_k \boldsymbol{u}_k + \boldsymbol{b}) + (\boldsymbol{A}_{k+1}\boldsymbol{u}_{k+1} + \boldsymbol{b})], \tag{13}$$

where the initial function $\boldsymbol{u}_0$ is given by (8). Note that the coefficient matrix $\boldsymbol{A}$ is not time independent since it is a function of $V(t)$. The global error of Heun's method is $O(\Delta t^2)$ (see Section 9.2 in [6] for a proof), so the global truncation error for the numerical approximation discussed in this section is $O(\Delta x^2 + \Delta t^2)$.

As we will show in Section 3.1, the explicit version of Heun's method is stable only when $\Delta t \leq \dfrac{1}{8}(\Delta x)^4$. Therefore, in practice we shall use the implicit Heun's method (which is stable for all $\Delta t > 0$), by solving the system of linear equations

$$(\boldsymbol{I} - \frac{\Delta t}{2}\boldsymbol{A}_{k+1})\boldsymbol{u}_{k+1} = (\boldsymbol{I} + \frac{\Delta t}{2}\boldsymbol{A}_k)\boldsymbol{u}_k + \Delta t \boldsymbol{b}, \tag{14}$$

where $\boldsymbol{I}$ is the $N \times N$ identity matrix. However, because the coefficient matrix $\boldsymbol{A}_{k+1}$ on the left hand side contains an unknown value of $V(t_{k+1})$, a prediction-correction method is necessary for solving this system of equations: $\boldsymbol{A}_{k+1}$ can be approximated using $\boldsymbol{A}_k$ at the prediction step to predict the value of $\boldsymbol{u}^*_{k+1}$, which can then be used to evaluate a prediction of $V(t^*_{k+1})$ using Formula (12); at the correction step, $\boldsymbol{A}_{k+1}$ will be updated from the prediction $V(t^*_{k+1})$ to obtain the corrected value of $\boldsymbol{u}_{k+1}$ and $V(t_{k+1})$. Since the implicit method is used in both prediction and correction steps, the unconditional stability is preserved.

3.1 Stability of Heun's Method

To study the stability of iterations, we consider the homogeneous equation associated with the evolution equation (5):

$$\frac{\partial u}{\partial t} = -\frac{\partial^4 u}{\partial x^4}, \quad x > 0, \quad t > 0.$$

Denote $u_n^k := u(x_n, t_k)$, the explicit Heun's method takes the form

$$u_n^{k+1} = u_n^k + \frac{\Delta t}{2}[f(u_n^k) + f(u_n^{k+1})] \tag{15}$$

at the point (x_n, t_k), where

$$f(u_n^k) = \frac{1}{(\Delta x)^4}[u_{n+2}^k - 4u_{n+1}^k + 6u_n^k - 4u_{n-1}^k + u_{n-2}^k], \tag{16}$$

and u_n^{k+1} on the right hand side of (15) can be approximated by Euler's method:

$$u_n^{k+1} = u_n^k + \Delta t f(u_n^k). \tag{17}$$

It can be shown (in page 466 of [6]) that the stability of explicit Heun's method is the same as Euler's method, so the stability region of the explicit method (15) and (16) can simply be determined from iterations of Euler's method (17). Denote $r = \frac{\Delta t}{(\Delta x)^4}$, Equation (17) can be rearranged as

$$u_n^{k+1} = (1 - 6r)u_n^k + 4r(u_{n+1}^k + u_{n-1}^k) - r(u_{n+2}^k + u_{n-2}^k).$$

Expanding u_n^k in the discrete Fourier transform $u_n^k := a^k e^{ipn}$, we have

$$a^{k+1} = a^k[1 - 4r(1 - \cos p)^2].$$

In order to get stable solutions, we need $|1 - 4r(1 - \cos p)^2| \leq 1$, which gives the stability constraint $r \leq \frac{1}{8}$, or $\Delta t \leq \frac{1}{8}(\Delta x)^4$.

The implicit Heun's method has the same form in (15), except that u_n^{k+1} on the right hand side is determined implicitly from a system of linear equations. Substituting f, we rewrite (15) as

$$\begin{aligned}(1 + 3r)u_n^{k+1} - 2r(u_{n+1}^{k+1} + u_{n-1}^{k+1}) + \frac{r}{2}(u_{n+2}^{k+1} + u_{n-2}^{k+1}) \\ = (1 - 3r)u_n^k + 2r(u_{n+1}^k + u_{n-1}^k) - \frac{r}{2}(u_{n+2}^k + u_{n-2}^k).\end{aligned}$$

By the same Fourier transform, we have

$$a^{k+1} = a^k \left[\frac{1 - 2r(1-\cos p)^2}{1 + 2r(1-\cos p)^2} \right].$$

Since $|1-2r(1-\cos p)^2| \le 1+2r(1-\cos p)^2$ for all $r > 0$, the implicit scheme is unconditionally stable.

Note that the above stability analysis is valid only when $V(t)$ is close to zero, and becomes invalid when $|V(t)|$ is large, e.g. near the blow-up time. Therefore, we shall use the adaptive method described in Section 9.3 of [6] to adjust Δt at each time step, so that the local truncation error is controlled in a certain tolerance level for all $t > 0$.

3.2 Error Analysis: Smoothness of the Solution

Another source of error, in addition to the truncation error from numerical approximation, is introduced when we determine the value of $V(t)$ from Equation (7), which is based on the assumption that a smooth solution exists at the boundary $x = 0$. In order to check this assumption, it is useful to find the residuals for higher order derivatives of u at $x = 0$.

In what follows, we shall denote $\beta(t) = u_x(0,t)$ for simplicity. Ideally, if we differentiate (5) with respect to x once and twice and set $x \to 0$, we should have

$$\frac{d\beta}{dt} + \left.\frac{\partial^5 u}{\partial x^5}\right|_{x=0} = -\frac{1}{2}V(t), \tag{18}$$

$$\left.\frac{\partial^6 u}{\partial x^6}\right|_{x=0} = 0. \tag{19}$$

However, since we do not have the data of u available at $x = x_{-3}$, we are not able to check if these two equalities hold independently. Alternatively, if we determine u_{-3} from (19):

$$\frac{u_3 - 6u_2 + 15u_1 - 20u_0 + 15u_{-1} - 6u_{-2} + u_{-3}}{(\Delta x)^6} \approx 0$$

$$\Rightarrow \quad u_{-3} = -u_3 + 12u_2 - 24u_1 + \frac{3}{2}(\Delta x)^2,$$

then the value of $d\beta/dt$ can be determined from Equation (18) and (12):

$$\begin{aligned} \frac{d\beta}{dt} &= -\left.\frac{\partial^5 u}{\partial x^5}\right|_{x=0} - \frac{1}{2}V(t) \\ &\approx -\frac{u_3 - 4u_2 + 5u_1 - 5u_{-1} + 4u_{-2} - u_{-3}}{2(\Delta x)^5} - \frac{u_2 - 4u_1 + 6u_0 - 4u_{-1} + u_{-2}}{(\Delta x)^3(u_1 - u_{-1})}. \end{aligned} \tag{20}$$

Thus, we can compare the value of $d\beta/dt$ determined from (20) with the numerical derivative

$$\left.\frac{d\beta}{dt}\right|_{t=t_k} \approx \frac{\beta(t_{k+1}) - \beta(t_{k-1})}{t_{k+1} - t_{k-1}} \tag{21}$$

to obtain an estimation of the residuals at the boundary $x = 0$.

3.3 Example: Negative Initial Velocity

The following numerical approximation is based on the initial function (9). The initial velocity determined from this initial condition is $V(0) = -1.25$.

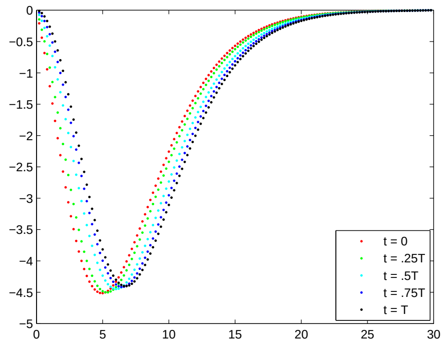

Figure 3: The dynamical evolution of u verses x at different time t (T is the terminal time).

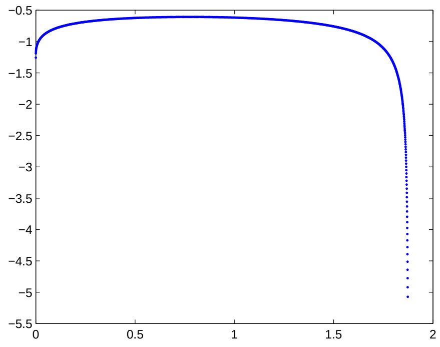

Figure 4: Change of velocity of the contact line V with respect to time t.

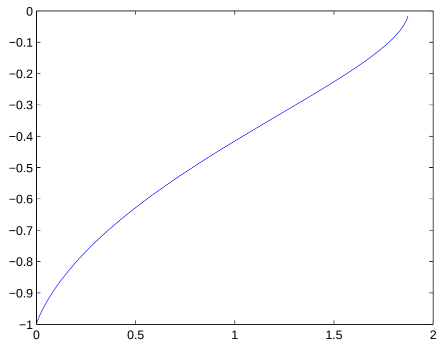

Figure 5: Boundary value $u_x|_{x=0}$ versus t.

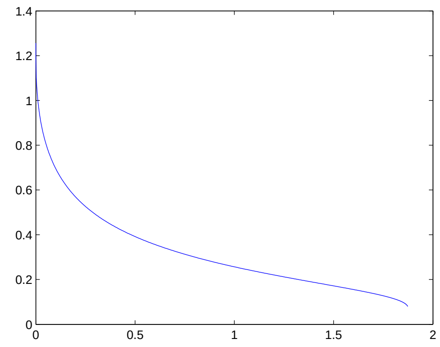

Figure 6: Boundary value $u_{xxxx}|_{x=0}$ versus t.

Figure 4 clearly shows a blow-up of V towards $-\infty$ at $t \approx 1.9$, while the solution $u(x,t)$ in Figure 3 remains regular at terminal time $t = T$. Recall that Equation (7) determines the velocity $V(t)$ by the quotient of $u_{xxxx}(0,t)$ and $\beta(t) = u_x(0,t)$, where $\beta(t)$ should be strictly negative for all $t > 0$. From Figure 5 we can see that the value of β is about to cross zero from the negative side at the terminal time, while $u_{xxxx}(0,t)$ in Figure 6 is also approaching zero near the terminal time, but with a much slower rate than $\beta(t)$. This also indicates that $|V(t)|$ is approaching infinity at the terminal time.

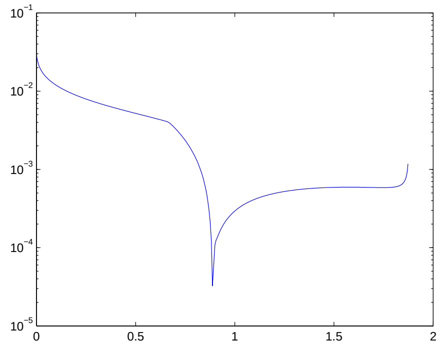

Figure 7: Error of $d\beta/dt$ versus t.

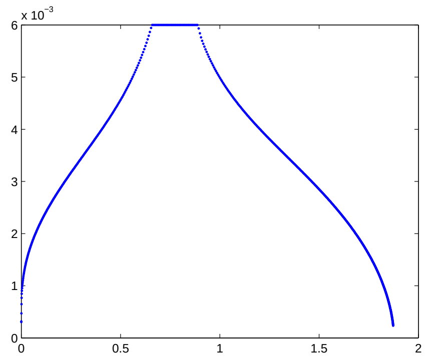

Figure 8: Time step sizes Δt versus t.

Figure 7 compares the value of $d\beta/dt$ between (20) and (21). The error remains small, therefore, the assumption that the solution is smooth (or at least C^6) at the boundary $x = 0$ is valid up to numerical accuracy. Figure 8 shows the time stepsize adjusted to preserve the same level of the local error of 10^{-5}, with an upper bound $t \leq 0.006$. This upper bound is needed because the error drops significantly near $t \approx 0.8$, and the time step adjustment procedure would otherwise produce large values of Δt.

3.4 Example: Positive Initial Velocity

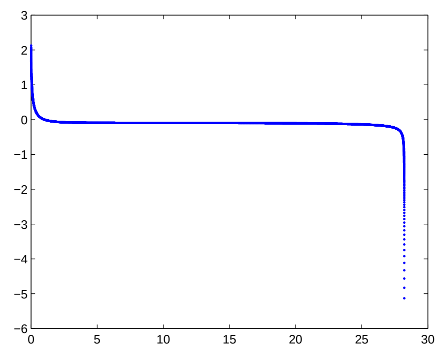

Figure 9: Change of velocity $V(t)$ starting from a large positive initial velocity.

Figure 9 shows the dynamical evolution of the velocity $V(t)$ when $V(0) = 2.35$, determined from the initial function with $a = 0.5$ and $b = 0.6$ in (8):

$$u_0(x) = -\frac{1}{4}e^{-x/2}x(4 + 3x + x^2 + 0.6x^3). \tag{22}$$

Although the terminal time $T \approx 28$ is much larger than that of a negative initial velocity, a blow-up is still detected from this particular initial condition. The solution $u(x,t)$ looks similar to the solution shown in Figure 3 and hence is not shown here.

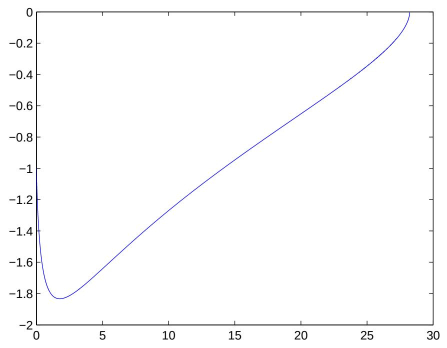

Figure 10: $u_x|_{x=0}$ versus t.

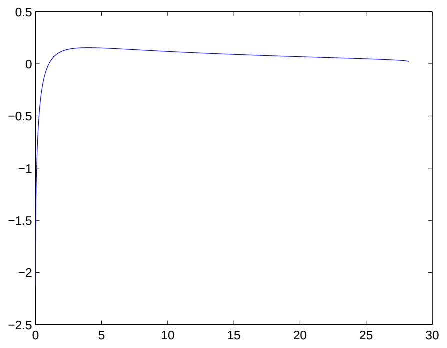

Figure 11: $u_{xxxx}|_{x=0}$ versus t.

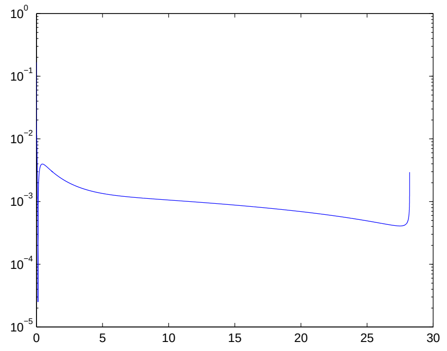

Figure 12: Error of $d\beta/dt$.

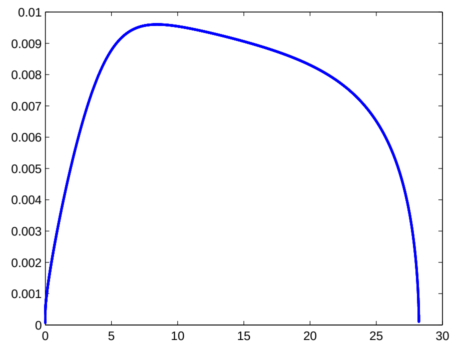

Figure 13: Time stepsize Δt.

Figures 10 and 11 show that $u_x|_{x=0}$ and $u_{xxxx}|_{x=0}$ cross zero at the terminal time. The level of error shown in Figure 12 is similar to the previous example and is acceptable. The adjusted time step sizes in Figure 13 agree roughly with the error for different time t, where the step size reduces when the error increases near the terminal time.

3.5 More Initial Velocities

The following plots illustrate the dynamical evolution of the velocities $V(t)$ under different initial conditions. From these plots, together with the previous examples, it is clear that the blow-up time depends on the initial velocity $V(0)$, where a large positive initial velocity leads to a much longer equilibrium state before the solution blows up.

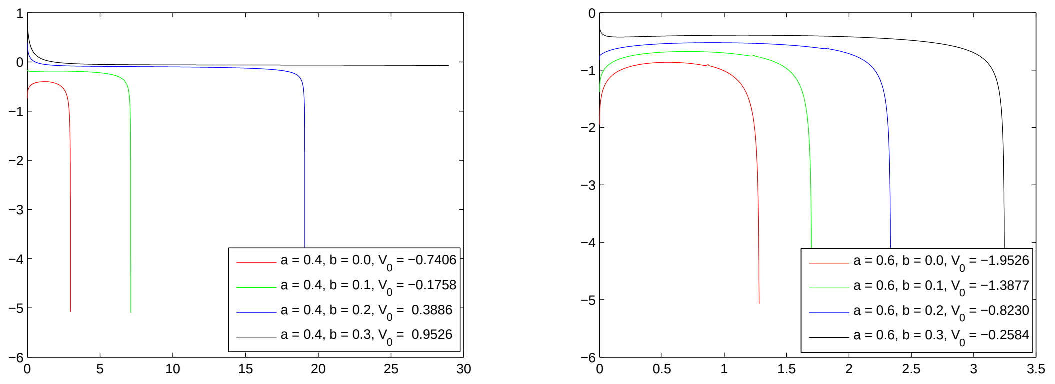

Figure 14: Comparison of velocities $V(t)$ between different initial conditions.

4 Blow-up Rate

In order to determine the exact blow-up time t_0, we will fit the numerical data near the terminal time T with the logarithmic function (3) and the power function (4).

To fit the data into (3), we first differentiate both sides of the expression with respect to t and take the inverse:

$$\frac{dV}{dt} = -\frac{C_1}{t_0 - t} \quad \Rightarrow \quad \left(\frac{dV}{dt}\right)^{-1} = \frac{t}{C_1} - \frac{t_0}{C_1}.$$

Then the constants C_1 and t_0 can be determined from a linear regression applied to the above equation. The other constant C_2 can be found from another linear regression applied to the original logarithmic function (3), where the values of $C_1 \log(t_0 - t)$ are known now.

The similar differentiation-regression method can also be used to find the coefficients in the power function (4), after taking the logarithm of both sides:

$$\log(-V(t)) = \log c - p \log(t_0 - t).$$

In particular, we differentiate the above expression and obtain:

$$\frac{1}{V(t)}\frac{dV}{dt} = \frac{p}{t_0 - t} \quad \Rightarrow \quad V(t)\left(\frac{dV}{dt}\right)^{-1} = \frac{t_0}{p} - \frac{t}{p}.$$

4.1 Examples

In practice, we found that the blow-up time t_0 and blow-up rate p in the power law or the coefficient C_1 in the logarithmic law vary with different time windows (i.e. the range of t which is used to fit the data). The following Matlab output gives a comparison of blow-up times and rates under different time windows and different tolerance levels, using the initial condition discussed in Section 3.3. Here *starting time* means the time t at which we start to fit the data, and *Error* is the mean squared error (MSE) defined by

$$\text{MSE} := \frac{1}{n-3}\sum (V_{\text{obs}} - V_{\text{fit}})^2,$$

where n is the total number of data points used in the regression.

```
Initial condition: a = 0.5, b = 0; initial velocity: V(0) = -1.2500

Tolerance level: 0.0001, number of iterations: 330, terminal time = 1.8729
  Starting time    Blowup time t0    Blowup rate p or C1       Error
powerlaw:
         1.8176         1.8749              0.3916            0.000017
         1.8356         1.8752              0.3994            0.000003
         1.8550         1.8756              0.4104            0.000000
loglaw:
         1.8176         1.8678              0.5371           23.732740
         1.8356         1.8695              0.6135           33.681247
         1.8550         1.8716              0.7578           68.934686

Tolerance level: 1e-006, number of iterations: 1448, terminal time = 1.8732
  Starting time    Blowup time t0    Blowup rate p or C1       Error
powerlaw:
         1.8172         1.8753              0.3927            0.000033
         1.8360         1.8757              0.4009            0.000006
         1.8547         1.8760              0.4118            0.000000
loglaw:
         1.8172         1.8688              0.5500           25.226547
         1.8360         1.8705              0.6343           33.937325
         1.8547         1.8724              0.7854           58.894321
```

The above table shows that the errors from logarithmic law are much larger than the errors from power law in all cases. Also, the error of power law reduces as the time window moves closer to the blow-up time, whereas the error of logarithmic law increases. Moreover, the blow-up times t_0 determined from the logarithmic law are smaller than terminal time.

Hence, the logarithmic law deviates from the numerical data near blow-up time. As we can see in Figure 15, the power function (4) fits our numerical data much better than the logarithmic function (3).

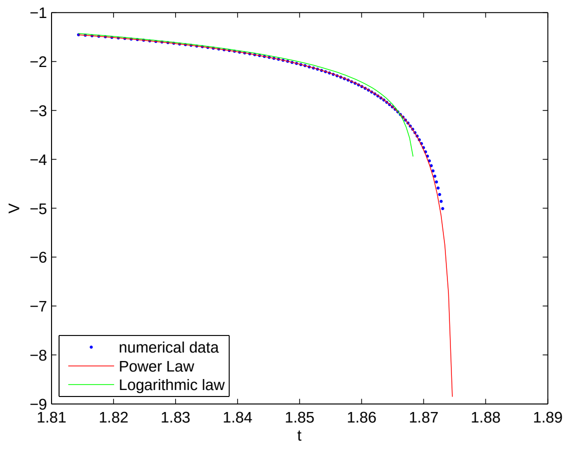

Figure 15: Comparison between data fitting with the logarithmic law and the power law.

Here are two more examples with different initial conditions:

```
Initial condition: a = 0.4, b = 0; initial velocity: V(0) = -0.7360

Tolerance level: 0.0001, number of iterations: 284, terminal time = 2.9646
  Starting time    Blowup time t0    Blowup rate p or C1       Error
powerlaw:
         2.8763          2.9652              0.3995             0.000104
         2.9058          2.9657              0.4078             0.000023
         2.9365          2.9662              0.4197             0.000002
loglaw:
         2.8763          2.9560              0.3906            12.400108
         2.9058          2.9585              0.4557            19.102593
         2.9365          2.9615              0.5870            37.017782

Tolerance level: 1e-006, number of iterations: 1364, terminal time = 2.9654
  Starting time    Blowup time t0    Blowup rate p or C1       Error
powerlaw:
         2.8765          2.9662              0.4001             0.000164
         2.9063          2.9666              0.4085             0.000030
         2.9358          2.9671              0.4201             0.000002
```

```
loglaw:
         2.8765          2.9574            0.3959          12.052519
         2.9063          2.9599            0.4640          18.350765
         2.9358          2.9626            0.5945          32.196180


Initial condition: a = 0.4, b = 0.2; initial velocity: V(0) = 0.4640

Tolerance level: 0.0001, number of iterations: 1154, terminal time = 19.0738
  Starting time    Blowup time t0    Blowup rate p or C1      Error
powerlaw:
        18.5099         19.0655            0.3972           0.000056
        18.6951         19.0684            0.4059           0.000012
        18.8868         19.0713            0.4189           0.000001
loglaw:
        18.5099         19.0162            0.1018           0.342647
        18.6951         19.0310            0.1200           0.554862
        18.8868         19.0475            0.1583           1.111706

Tolerance level: 1e-006, number of iterations: 4233, terminal time = 19.0885
  Starting time    Blowup time t0    Blowup rate p or C1      Error
powerlaw:
        18.5172         19.0815            0.3982           0.000715
        18.7069         19.0842            0.4073           0.000142
        18.8986         19.0867            0.4208           0.000009
loglaw:
        18.5172         19.0382            0.1044           0.412399
        18.7069         19.0521            0.1241           0.660228
        18.8986         19.0670            0.1659           1.472944
```

4.2 Blow-up Rate

The results of data fitting suggest that for both small and large blow-up times t_0, the power law gives a consistent estimation of the blow-up rate p, with $p \approx 0.4$. It is suggested in Section 4 of [5] that the power law (4) can be applicable with $p = 0.5$, provided that $u_{xxxx}|_{x=0}$ remains non-zero as $t \to t_0$. However, as we have seen in Figure 6 and 11, $u_{xxxx}|_{x=0}$ is also approaching zero near the blow-up time. Thus, we are not able to justify the power law (4) with a precise value of p at this time. More accurate and computationally efficient numerical simulations are needed before we can make any further conclusions.

5 Conclusion

We conclude from the numerical simulations of the model problem (5) - (6) that, for any suitable initial condition in the form (8), there always exists a finite positive time t_0 such that $V(t) \to -\infty$ as $t \to t_0$, although the blow-up time t_0 varies from different initial velocity $V(0)$. With a large positive initial velocity, the solution tends to have a longer equilibrium state before it eventually blows up, whereas a negative initial velocity yields a much smaller value of the blow-up time t_0.

The numerical results also suggest that the behavior of $V(t)$ near the blow-up time satisfies the power law (4), with a blow-up rate $p \approx 0.4$. However, the true value of p remains unknown at this time, and requires development of more precise and computationally efficient numerical methods.

Because the model equation (5) is already a fourth order PDE, we shall avoid using any numerical methods that involves higher order central differences. In addition, because of the unknown variable $V(t)$, it is difficult to use other higher order implicit methods to solve the system of ODEs (11). Thus, the finite difference method has a limited accuracy. Therefore, a different approach is needed, for instance, by using the collocation method involving discrete Fourier transform. This will remain open for further studies.

References

[1] E.S. Benilov and M. Vynnycky, *Contact lines with a* $180°$ *contact angle*, J. Fluid Mech. (2013), vol. 718, pp. 481-506.

[2] D.J. Benney and W.J. Timson, *The rolling motion of a viscous fluid on and off a rigid surface*, Stud. Appl. Math. (1980), vol. 63, pp. 9398.

[3] C.G. Ngan and V.E.B. Dussan, *The moving contact line with a* $180°$ *advancing contact angle*, Phys. Fluids (1984), vol. 24, pp. 2785-2787.

[4] D.E. Pelinovsky, A.R. Giniyatullin, and Y.A. Panfilova, *On solutions of a reduced model for the dynamical evolution of contact lines*, Transactions of Nizhni Novgorod State Technical University n.a. Alexeev N.4 (94), 45-60 (2012)

[5] D.E. Pelinovsky and A.R. Giniyatullin, *Finite-time singularities in the dynamical evolution of contact lines*, Bulletin of the Moscow State Regional University (Physics and Mathematics) 2012 N.3, 14-24 (2012)

[6] M. Grasselli and D.E. Pelinovsky, *Numerical Mathematics*, Jones and Bartlett, Boston (2008)

A MATLAB Codes

A.1 Finite Difference Method

```
function [k, t, V, x, u] = Diffusion_Diff(a, b, Etol, kbreak, L, N)
% This function uses finite difference method discussed in Section 3 to
% approximate the numerical solutions of u(x, t) and V(t). The iteration
% will terminate when absolute value of V is greater than 5, which is
% considered as numerical evidence of blow-up.
%
% k is the total number of iterations.
% a and b are parameters in the initial condition.
% Etol is the error tolerance level, the default value is Etol = 10^-5.
% The default value of the right boundary L is 15/a.
% The default value of the number of interior grid points N is 149.
%
% kbreak is the maximum number of iterations. If no blow-up is detected
% when this number is reached, the function will terminate automatically
% and save the data to Diffusion_Diff.mat for future computation. The
% default value is kbreak = 1000.
%
% If no input is specified, the function will load data from
% Diffusion_Diff.mat and continue the computation.


if nargin ~= 0
    if a <= 0, error('a must be strictly positive.'); end
    if b < 0, error('b must be non-negative.'); end
    if nargin < 5
        L = 15/a;   N = 149;
        if nargin < 4
            kbreak = 1001;
            if nargin < 3
                Etol = 10^-5;
            end
        end
    end

    % Discretization over x-axis:
    x = linspace(0, L, N+2);    x = x(2 : N+1)';    dx = L/(N+1);
    % Initial condition:
    u = -exp(-a*x).*x/4.*(4 + (4*a+1)*x + a*(2*a+1)*x.^2 + b*x.^3);
    V = 4/dx^3*(2*u(2) - 4*u(1) + dx^2)/(4*u(1) + dx^2);
    Vexact = 6*b - 4*a^3 - 3*a^2;
    fprintf('Initial velocity: numerical: %.4f, analytical: %.4f.\n', V, Vexact)

    t = 0;  dt = Etol/abs(V - Vexact);
else
    load Diffusion_Diff
end

% Adaptive method:
for k = length(t) - 1 + (1 : kbreak)
    if abs(V(k)) > 5, break; end

    % approximation at dt
    [uPred, VPred] = Heun(dx, dt, u(:,k), V(k));

    % approximation at dt/2
    [u1, V1] = Heun(dx, dt/2, u(:,k), V(k));
    [u2, V2] = Heun(dx, dt/2, u1, V1);

    % optimal time stepsize for next step
    Error = 4*norm(u2-uPred)/3;
    s = (Etol/Error)^(1/3);
    dt = dt*s;
    t(k+1) = t(k) + dt;
    [u(:,k+1), V(k+1)] = Heun(dx, dt, u(:,k), V(k));
end

if k < 10
    warning('initialFunction:Velocity',...
        'Initial velocity too large: V(0) = %.4f.', V(1));
elseif mod(k, kbreak) == 0
    warning('Diffusion_Diff:iterationTerminated',...
        'Number of iterations exceeds %g, no blow-up detected when t < %.4f.',...
        k, t(k));
    save Diffusion_Diff
end
```

A.2 Implicit Heun's Method

```
function [u, V] = Heun(dx, dt, u0, V0)
% This function uses implicit Heun's method to compute u and V for the next
% time step.
% dx and dt are space and time stepsizes, respectively.
% u0 and V0 take values of u and V, respectively, at the current time step.

N = length(u0);
b = [1/(2*dx^2); zeros(N - 1, 1)];
A0 = A_diff(dx, N, V0);

% prediction
u1 = (eye(N) - dt*A0/2)\((eye(N) + dt*A0/2)*u0 + dt*b);
V1 = (4/dx^3)*(2*u1(2) - 4*u1(1) + dx^2)/(4*u1(1) + dx^2);
A1 = A_diff(dx, N, V1);

% correction
u = (eye(N) - dt*A1/2)\((eye(N) + dt*A0/2)*u0 + dt*b);
V = (4/dx^3)*(2*u(2) - 4*u(1) + dx^2)/(4*u(1) + dx^2);


function A = A_diff(dx, N, V)
% Matrix A in the finite difference method.

A = - 1/dx^4*diag(ones(1, N-2), -2)...
    + (4/dx^4 - V/(2*dx))*diag(ones(1, N-1), -1)...
    - 6/dx^4*diag(ones(1, N))...
    + (4/dx^4 + V/(2*dx))*diag(ones(1, N-1), 1)...
    - 1/dx^4*diag(ones(1, N-2) ,2);
A = A + [1/dx^4 zeros(1, N-1); zeros(N-2, N); zeros(1, N-1) 1/dx^4];
```

A.3 Approximation of Solutions

```
clear all; close all
a = input('Set value of a = ');
b = input('Set value of b = ');
tol = input('Set tolerance level = 10^');
kbreak = 1000;
```

```
[k, t, V, x, u] = Diffusion_Diff(a, b, 10^tol, kbreak);
if k < 10, break, end
figure(1); plot(t(1:k-1), diff(t), '.'); title('Time stepsize')
figure(2); plot(t, V, '.'); title('V(t)')

while mod(k, kbreak) == 0 & k < 5*kbreak
    [k, t, V, x, u] = Diffusion_Diff;
    figure(1); plot(t(1:k-1), diff(t), '.'); title('Time stepsize')
    figure(2); plot(t, V, '.'); title('V(t)')
end
fprintf('Number of iterations: %g, terminal time: T = %.4f.\n', k, t(k))
if k < 5*kbreak, save Diffusion_Diff_2, end

K = floor(k/4)*4;
figure(3);
plot(x',u(:,1),'.r'); hold on;
plot(x',u(:,K/4+1),'.g');
plot(x',u(:,K/2+1),'.c');
plot(x',u(:,3*K/4+1),'.b');
plot(x',u(:,K+1),'.k'); hold off
legend('t = 0', 't = .25T', 't = .5T', 't = .75T', 't = T')


dx = x(2) - x(1);
for k = 1 : length(t)
    beta(k) = u(1,k)/dx + dx/4;
    u4(k) = (2*u(2,k) - 4*u(1,k) + dx^2)/dx^4;

    dbeta1(k) = -V(k)/2 - (2*u(3,k) - 12*u(2,k) + 18*u(1,k) - 3*dx^2)/(2*dx^5);
    if k > 2
        dbeta2(k-2) = (beta(k) - beta(k-2))/(t(k) - t(k-2));
        dbetaError(k-2) = abs(dbeta1(k-1) - dbeta2(k-2));
    end
end
figure(4); plot(t, beta); title('u_x(0, t)');
figure(5); plot(t, u4); title('u_{xxxx}(0, t)');
figure(6); semilogy(t(2:k-1), dbetaError); title('Error of d\beta/dt');
```

A.4 Power Law

```
function [c, p, t0, Error] = blowupPower(t, V)

w = abs(V);
n = length(t);
dt = diff(t(1:n-1) + t(2:n));
dw = diff(w(1:n-1) + w(2:n));

y = w(2:n-1).*dt./dw;
b = polyfit(t(2:n-1), y, 1);

p = -1/b(1);
t0 = b(2)*p;

y = log(w) + p*log(t0-t);
c = exp(mean(y));

Error = norm(w - c./(t0-t).^p)^2/(n-3);
```

A.5 Logarithmic Law

```
function [c1, c2, t0, Error] = blowupLog(t, V)

w = abs(V);
n = length(t);
dt = diff(t(1:n-1) + t(2:n));
dw = diff(w(1:n-1) + w(2:n));

y = dt./dw;
b = polyfit(t(2:n-1), y, 1);

c1 = -1/b(1);
t0 = b(2)*c1;

y = w + c1*log(t0-t);
c2 = mean(y);

Error = norm(w - c1*log(t0 - t) - c2)^2/(n-3);
```

A.6 Blow-up Rate

```
% This script fits the numerical data of V(t) near the blow-up time, and
% compares the results between the power law and the logarithmic law. The
% numerical data that are subject to large error near the terminal time are
% excluded in the data fitting.


clear all
a = input('set value of a = ');
b = input('set value of b = ');
kbreak = 2000;                  % change maximum number of iterations here
fileID = fopen('blowupRate.txt', 'w');
fprintf(fileID, 'Initial condition: a = %g, b = %g; ', a, b);

for Etol = 10.^-[4 6]           % change tolerance level here
    [k, t, V] = Diffusion_Diff(a, b, Etol, kbreak);
    if k < 10, fprintf(fileID, ['\r\nWarning: ' lastwarn]); break, end
    if Etol == 10^-4, fprintf(fileID, 'initial velocity: %.4f\r\n', V(1)); end
    fprintf(fileID, '\r\nTolerance level: %g, ', Etol);
    if k == kbreak, fprintf(fileID, ['\r\nWarning: ' lastwarn]); break, end
    fprintf('Number of iterations: %g, terminal time: %.4f.\n', k, t(k));

    for j = 1 : 3
        span = t > t(k)*(.96+.01*j) & t < t(round(.98*k));
        tSpan = t(span);    VSpan = V(span);    tStart(j) = tSpan(1);

        [c(j), p(j), tpwr(j), Epwr(j)] = blowupPower(tSpan, VSpan);
        [c1(j), c2(j), tlog(j), Elog(j)] = blowupLog(tSpan, VSpan);
    end

    fprintf(fileID, 'Number of iterations: %g, terminal time: %.4f\r\n', k, t(k));
    fprintf(fileID, '%15s %15s %20s %12s\r\n', 'Starting time',...
        'Blowup time t0', 'Blowup rate p or C1', 'Error');
    fprintf(fileID, 'powerlaw:\r\n');
    fprintf(fileID, '%15.4f %15.4f %15.4f %17.6f\r\n', [tStart; tpwr; p; Epwr]);
    fprintf(fileID, 'loglaw:\r\n');
    fprintf(fileID, '%15.4f %15.4f %15.4f %17.6f\r\n', [tStart; tlog; c1; Elog]);
end
fclose(fileID);
```

A.7 Blow-up Time

```
% This script plots the fitted data from power law and logarithmic law,
% together with the numerical data of V(t) near the blow-up time.


clear all
key = input(['Type 1 to load previously saved data, \n'...
    '0 to start with new initial condition: ']);
if key == 0
    a = input('Set value of a = ');
    b = input('Set value of b = ');
    tol = input('Set tolerance level = 10^');
    kbreak = 3000;              % change maximum number of iterations here
    [k, t, V] = Diffusion_Diff(a, b, 10^tol, kbreak);
    if k < 10 | mod(k, kbreak) == 0, break, end
elseif key == 1
    load Diffusion_Diff_2;
    clear x u
else
    error('invalid input')
end

span = t > .98*t(k);
figure(7); plot(t(span), V(span), '.'); hold on
span = t > .98*t(k) & t < t(round(.98*k));
tSpan = t(span);    VSpan = V(span);

[c, p, tpwr] = blowupPower(tSpan, VSpan);
tt = linspace(tSpan(1),tpwr,101);
tt = tt(1:100);
w = -c./(tpwr-tt).^p;
plot(tt,w,'-r')

[c1, c2, tlog] = blowupLog(tSpan, VSpan);
tt = linspace(tSpan(1),tlog,101);
tt = tt(1:100);
w = c1*log(tlog-tt) - c2;
plot(tt,w,'-g')
legend('numerical approximation', 'power law', 'log law'); hold off
```