Exam 2018 - Computational Physics

Frode Børseth

February 2018

I have worked a lot in the physics lunch room, where I have had lots of good discussions with Nastasia, Michael and Vidar.

1 Schemes and plots

The set of partial differential equations to be solved is in the functions a, b, c, s with variables x, t,

$$\frac{\partial a(x,t)}{\partial t} = D_a \frac{\partial^2 a(x,t)}{\partial x^2} - R\, a(x,t)b(x,t) \tag{1a}$$
$$\frac{\partial b(x,t)}{\partial t} = D_b \frac{\partial^2 b(x,t)}{\partial x^2} - R\, a(x,t)b(x,t) \tag{1b}$$
$$\frac{\partial c(x,t)}{\partial t} = D_c \frac{\partial^2 c(x,t)}{\partial x^2} + R\, a(x,t)b(x,t) - N_1\, \theta\big(c(x,t) - c_0\big)c^2(x,t) - N_2\, c(x,t)s(x,t) \tag{1c}$$
$$\frac{\partial s(x,t)}{\partial t} = N_1\, \theta\big(c(x,t) - c_0\big)c^2(x,t) + N_2\, c(x,t)s(x,t) \tag{1d}$$

The boundary conditions are of Dirichlet type, specifically,

$$\begin{aligned} a(0,t) &= a_0, \qquad b(0,t) = 0, \\ a(L,t) &= 0, \qquad b(L,t) = b_0, \end{aligned}$$

By introducing the dimensionless variables by $\xi = x/x_c$ and $\tau = t/t_c$ for some characteristic length and time x_c, t_c, the partial derivatives can be written

$$\frac{\partial}{\partial t} = \frac{1}{\Delta t}\frac{\partial}{\partial \tau}, \qquad \frac{\partial^2}{\partial x^2} = \frac{1}{(x_c)^2}\frac{\partial^2}{\partial \xi^2}$$

Now let us define new quantities that are all dimensionless using these factors. Because $[g] = 1/[x_c]$, the units of the equation are $1/[x_c][t_c]$, so that

$$[D_g] = \frac{[x_c]^2}{[t_c]}, \qquad [R] = [N_1] = [N_2] = \frac{[x_c]}{[t_c]}$$

We can define dimensionless quantities from the old ones by multiplying and dividing by powers of x_c and t_c,

$$\bar{g} \equiv x_c\, g, \qquad \bar{D_g} \equiv \frac{t_c}{x_c^2} D_g, \qquad \bar{R} \equiv \frac{t_c}{x_c} R, \qquad \bar{N_1} \equiv \frac{t_c}{x_c} N_1, \qquad \bar{N_2} \equiv \frac{t_c}{x_c} N_2,$$

or going the other way,

$$g = \frac{1}{x_c}\bar{g}, \qquad D_g = \frac{x_c^2}{t_c}\bar{D_g}, \qquad R = \frac{x_c}{t_c}\bar{R}, \qquad N_1 = \frac{x_c}{t_c}\bar{N_1}, \qquad N_2 = \frac{x_c}{t_c}\bar{N_2}.$$

When inserting all these new dimensionless quantities in the old equation with the factors of x_c and t_c, we get for the first equation

$$\frac{1}{t_c}\frac{\partial}{\partial \tau}\frac{1}{x_c}\bar{a}(\xi,\tau) = \frac{x_c^2}{t_c}\bar{D_a}\frac{1}{x_c^2}\frac{\partial^2}{\partial \xi^2}\frac{1}{x_c}\bar{a}(\xi,\tau) - \frac{x_c}{t_c}\bar{R}\,\frac{1}{x_c}\bar{a}(\xi,\tau)\frac{1}{x_c}\bar{b}(\xi,\tau) \tag{2}$$

which after some cleaning up becomes

$$\frac{1}{t_c x_c}\frac{\partial}{\partial \tau}\bar{a}(\xi,\tau) = \frac{1}{t_c x_c}\bar{D_a}\frac{\partial^2}{\partial \xi^2}\bar{a}(\xi,\tau) - \frac{1}{t_c x_c}\bar{R}\,\bar{a}(\xi,\tau)\bar{b}(\xi,\tau) \tag{3}$$

and finally, upon multiplication by $(t_c x_c)$, we get

$$\frac{\partial}{\partial\tau}\bar{a}(\xi,\tau) \;=\; \bar{D_a}\frac{\partial^2}{\partial\xi^2}\bar{a}(\xi,\tau) \;-\; \bar{R}\,\bar{a}(\xi,\tau)\bar{b}(\xi,\tau). \tag{4}$$

Doing the same on the other three equations as well, we obtain a dimensionless set of partial differential equations.

$$\frac{\partial}{\partial\tau}\bar{a}(\xi,\tau) \;=\; \bar{D_a}\frac{\partial^2}{\partial\xi^2}\bar{a}(\xi,\tau) \;-\; \bar{R}\,\bar{a}(\xi,\tau)\bar{b}(\xi,\tau) \tag{5a}$$

$$\frac{\partial}{\partial\tau}\bar{b}(\xi,\tau) \;=\; \bar{D_b}\frac{\partial^2}{\partial\xi^2}\bar{b}(\xi,\tau) \;-\; \bar{R}\,\bar{a}(\xi,\tau)\bar{b}(\xi,\tau) \tag{5b}$$

$$\frac{\partial}{\partial\tau}\bar{c}(\xi,\tau) \;=\; \bar{D_c}\frac{\partial^2}{\partial\xi^2}\bar{c}(\xi,\tau) \;+\; \bar{R}\,\bar{a}(\xi,\tau)\bar{b}(\xi,\tau) \;-\; \bar{N_1}\,\theta\big(\bar{c}(\xi,\tau)-\bar{c_0}\big)\,\bar{c}^2(\xi,\tau) \;-\; \bar{N_2}\,\bar{c}(\xi,\tau)\bar{s}(\xi,\tau) \tag{5c}$$

$$\frac{\partial}{\partial\tau}\bar{s}(\xi,\tau) \;=\; \bar{N_1}\,\theta\big(\bar{c}(\xi,\tau)-\bar{c_0}\big)\,\bar{c}^2(\xi,\tau) \;+\; \bar{N_2}\,\bar{c}(\xi,\tau)\bar{s}(\xi,\tau) \tag{5d}$$

Evidently, nothing much changes when rephrasing the problem in dimensionless quantities. The constants may have to be scaled by powers of x_c and t_c, but essentially the equations are the same.

To arrive at a scheme for solving this set of equations numerically, it will be useful to recall some findings from assignment 3. The diffusion equation in 1 spatial dimension for the function u reads

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(D\frac{\partial u}{\partial x}\right). \tag{6}$$

The functions u and D (I will try to include some notion of spacial dependence in D already) were discretized by writing

$$u_i^n \equiv u(x_i,t_n) = u(i\,\delta x, n\,\delta t)$$
$$D_i \equiv D(x_i) = D(i\,\delta x).$$

Taking the second partial derivative and evaluating in x_i, we have

$$\frac{\partial}{\partial x}\left(D(x,t)\frac{\partial u(x,t)}{\partial x}\right)\bigg|_{x_i} = \left(\frac{\partial D(x,t)}{\partial x}\frac{\partial u(x,t)}{\partial x} + D(x,t)\frac{\partial^2 u(x,t)}{\partial x^2}\right)\bigg|_{x_i}$$

Our three-point scheme for $\partial^2 u/\partial x^2$ is of order 2, so we want the same to be true for the first derivatives of D and u as well. This will not hold for the forward or backward difference approximations, but we may use the central difference instead. Central difference is of order 2, so we have

$$\frac{\partial}{\partial x}\left(D(x,t)\frac{\partial u(x,t)}{\partial x}\right)\bigg|_{x_i} = \frac{(D_{i+1}-D_{i-1})}{2\Delta x}\frac{(u_{i+1}-u_{i-1})}{2\Delta x} + D_i\frac{u_{i+1}-2u_i+u_{i-1}}{\Delta x^2} + \mathcal{O}(\Delta x^2)$$

or rewriting as an approximation,

$$\frac{\partial}{\partial x}\left(D(x,t)\frac{\partial u(x,t)}{\partial x}\right)\bigg|_{x_i} \approx \frac{1}{(2\Delta x)^2}\Big(\big(D_{i+1}+4D_i-D_{i-1}\big)u_{i+1} - 8D_iu_i + \big(D_{i-1}+4D_i-D_{i+1}\big)u_{i+1}\Big) \tag{7}$$

Note that, for the much simpler case when D is a constant, this simplifies to the familiar expression

$$\frac{\partial}{\partial x}\left(D(x,t)\frac{\partial u(x,t)}{\partial x}\right)\bigg|_{x_i} \approx \frac{D}{\Delta x^2}\left(u_{i+1}-2u_i+u_{i-1}\right)$$

and in general we can write down all of the above as a matrix operating on the vector $\mathbf{u} = (u_1,\ldots,u_N)$, by a tridiagonal matrix M

$$M\mathbf{u} = \frac{\Delta t}{(2\Delta x)^2}\begin{pmatrix} \ddots & & \\ (D_{i+1}+4D_i-D_{i-1}) & -8D_i & (D_{i-1}+4D_i-D_{i+1}) \\ & & \ddots \end{pmatrix}\begin{pmatrix}\vdots\\ u_i \\ \vdots\end{pmatrix}$$

For Dirichlet boundary conditions, the top and bottom rows of M must be zeroed out.

Note that I have neglected to specify the time in the expressions above in order to remain as general as possible. The time derivative on the left hand side of (6) can be made as either a forward or backward difference approximation, leading to either the explicit or implicit Euler methods respectively. The full scheme then reads, for explicit Euler as an example as a matrix expression,

$$\mathbf{u}^{n+1} = \mathbf{u}^n + M\mathbf{u}^n$$

All three schemes that were studied in assignment 3, explicit Euler, implicit Euler, and Crank-Nicolson, could be written in this way, as matrices A and B

$$A\mathbf{u}^{n+1} = B\mathbf{u}^n. \tag{8}$$

where, in each of the three, the matrices were

$$\mathbf{u}^{n+1} = (I + M)\mathbf{u}^n \qquad \text{(Explicit Euler)} \tag{9}$$

$$(I - M)\mathbf{u}^{n+1} = \mathbf{u}^n \qquad \text{(Implicit Euler)} \tag{10}$$

$$\left(I - \frac{1}{2}M\right)\mathbf{u}^{n+1} = \left(I + \frac{1}{2}M\right)\mathbf{u}^n \qquad \text{(Crank-Nicolson)} \tag{11}$$

When we now go on to discretize in (5), the diffusion terms can be simplified to such matrix products immideately, without yet specifying which of the three schemes they represent. Evaluating the nonlinear terms in t_n for the moment, we get

$$\sum_{j=1}^{N} A^a_{i,j} a^{n+1}_j = \sum_{j=1}^{N} B^a_{i,j} a^n_j - \Delta\tau\bar{R}\, a^n_i b^n_i \tag{12a}$$

$$\sum_{j=1}^{N} A^b_{i,j} b^{n+1}_j = \sum_{j=1}^{N} B^b_{i,j} b^n_j - \Delta\tau\bar{R}\, a^n_i b^n_i \tag{12b}$$

$$\sum_{j=1}^{N} A^c_{i,j} c^{n+1}_j = \sum_{j=1}^{N} B^c_{i,j} c^n_j + \Delta\tau\bar{R}\, a^n_i b^n_i - \Delta\tau\bar{N}_1\theta\big(c^n_i - \bar{c_0}\big)\, c^n_i c^n_i - \Delta\tau\bar{N}_2\, c^n_i s^n_i \tag{12c}$$

$$s^{n+1}_i = s^n_i + \Delta\tau\bar{N}_1\theta\big(c^n_i - \bar{c_0}\big)\, c^n_i c^n_i + \Delta\tau\bar{N}_2\, c^n_i s^n_i \tag{12d}$$

$$\tag{12e}$$

This is looking promising, and is in fact a fully fledged scheme for solving the equations numerically. It would be very convenient to find a matrix expression instead of these, but for that we will have to deal with the nonlinear terms. To that end, let us define the five diagonal matrices

Reaction terms Nucleation terms (13)

$$R^n_a = R_a(\mathbf{b}^n) = \Delta\tau\bar{R}\begin{pmatrix} b^n_0 & & \\ & \ddots & \\ & & b^n_N \end{pmatrix} \qquad N^n_1 = N_1(\mathbf{c}^n) = \Delta\tau\bar{N}_1\begin{pmatrix} \theta(c^n_0 - \bar{c_0})c^n_0 & & \\ & \ddots & \\ & & \theta(c^n_N - \bar{c_0})c^n_N \end{pmatrix} \tag{14}$$

$$R^n_b = R_b(\mathbf{a}^n) = \Delta\tau\bar{R}\begin{pmatrix} a^n_0 & & \\ & \ddots & \\ & & a^n_N \end{pmatrix} \qquad N^n_{2c} = N_{2c}(\mathbf{s}^n) = \Delta\tau\bar{N}_2\begin{pmatrix} s^n_0 & & \\ & \ddots & \\ & & s^n_N \end{pmatrix} \tag{15}$$

$$N^n_{2s} = N_{2s}(\mathbf{c}^n) = \Delta\tau\bar{N}_2\begin{pmatrix} c^n_0 & & \\ & \ddots & \\ & & c^n_N \end{pmatrix} \tag{16}$$

All of which have to have their top and bottom rows zeroed out with our Dirichlet boundary conditions.

With these, we may write the scheme more compactly as

$$A_a\mathbf{a}^{n+1} = (B_a - R^n_a)\mathbf{a}^n \tag{17a}$$

$$A_b\mathbf{b}^{n+1} = (B_b - R^n_b)\mathbf{b}^n \tag{17b}$$

$$A_c\mathbf{c}^{n+1} = R^n_b\mathbf{b}^n + (B_c - N^n_1 - N^n_{2c})\mathbf{c}^n \tag{17c}$$

$$\mathbf{s}^{n+1} = N^n_1\mathbf{c}^n + (I + N^n_{2s})\mathbf{s}^n \tag{17d}$$

Because the nonlinear terms belonging to time n are multiplied by matrices that depend on n also, it is difficult to write down a scheme where these are included implicitly. In such a scheme, the matrices R^{n+1}_* N^{n+1}_* would themselves be unknown.

One possibility might be to include terms like $M^n_R\mathbf{b}^{n+1}$, where the matrix is built from the previous time step while the vector belongs to the next. This does not appear physically sensible however, as the instantaneous reaction rate does not depend on relative concentrations at previous or later times, but at the *same* time. For this reason I first want to leave the nonlinear terms on the $t = t_n$ side of the equation, unless and until any problems appear.

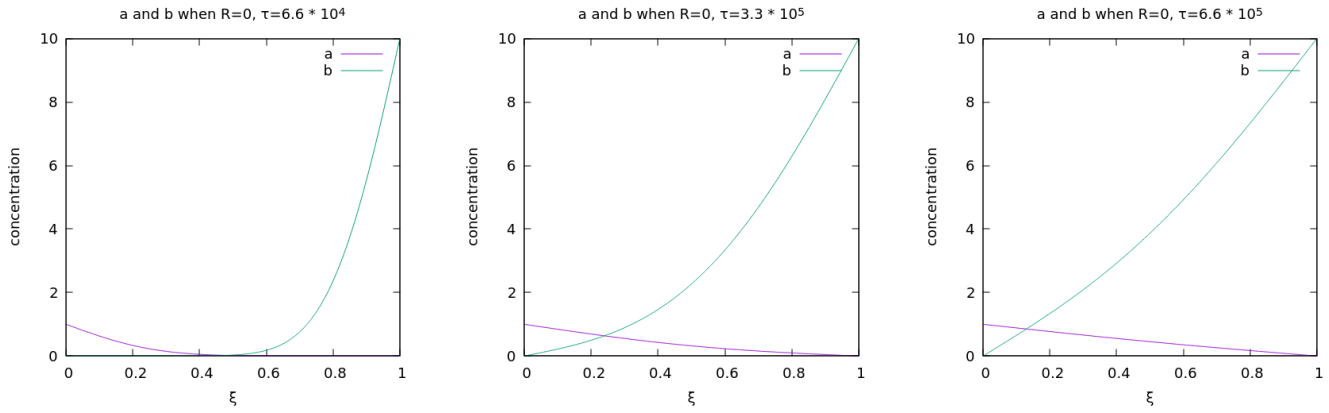

Figure 1: Trivial trial, $R = 0$, concentrations a and b.

We may write the scheme above in a more transparent way by placing all four of $\mathbf{a}^n, \mathbf{b}^n, \mathbf{c}^n \mathbf{s}^n$ in one single collumn vector,

$$\begin{pmatrix} A_a & & & \\ & A_b & & \\ & & A_c & \\ & & & I \end{pmatrix} \begin{pmatrix} \mathbf{a}^{n+1} \\ \mathbf{b}^{n+1} \\ \mathbf{c}^{n+1} \\ \mathbf{s}^{n+1} \end{pmatrix} = \begin{pmatrix} B_a - R_a^n & & & \\ & B_b - R_b^n & & \\ R_b^n & & B_c - N_1^n - N_{2c}^n & \\ & & N_1^n & I + N_{2s}^n \end{pmatrix} \begin{pmatrix} \mathbf{a}^n \\ \mathbf{b}^n \\ \mathbf{c}^n \\ \mathbf{s}^n \end{pmatrix} \tag{18}$$

The matrix on the left is at most tridiagonal, no matter which of the three schemes mentioned above that we decide to use, so it can be solved for in linear computation time using a tridiagonal solver. The matrix to the right is very sparse, all of its blocks being either diagonal or tridiagonal, so the time needed to multiply it with the column vector is linear as well when stored as a sparse matrix. This is looking promising indeed!

The reason I keep things so general at this point is because I was put off by the long simulation times I initially got. Granted, I was using Python, which can easily become inefficient when not written with care, but I still wanted to see if there was any way to safely speed up the simulation. I will go into detail on this in the stability section below.

I decided to run a test where $R = 0.0$, that is without any reaction and creation of the gas c. This would imply just regular diffusion of a and b, so the step length $\Delta\tau$ was chosen to satisfy the CFL condition,

$$\frac{D_g \Delta t}{\Delta \xi^2} < \frac{1}{2}.$$

I used a completely explicit scheme for this initial trial, by formulas

$$a_i^{n+1} = a_i^n + \frac{\bar{D}_a \Delta\tau}{\Delta\xi^2}(a_{i-1}^n - 2a_i^n + a_{i+1}^n) - \Delta\tau \bar{R} a_i^n b_i^n \tag{19a}$$

$$b_i^{n+1} = b_i^n + \frac{\bar{D}_b \Delta\tau}{\Delta\xi^2}(b_{i-1}^n - 2b_i^n + b_{i+1}^n) - \Delta\tau \bar{R} a_i^n b_i^n \tag{19b}$$

$$c_i^{n+1} = c_i^n + \frac{\bar{D}_c \Delta\tau}{\Delta\xi^2}(c_{i-1}^n - 2c_i^n + c_{i+1}^n) + \Delta\tau \bar{R} a_i^n b_i^n - \Delta\tau \bar{N}_1 \theta(c_i^n - c_0) c_i^n c_i^n - \Delta\tau \bar{N}_2 c_i^n s_i^n \tag{19c}$$

$$s_i^{n+1} = s_i^n + \Delta\tau \bar{N}_1 \theta(c_i^n - c_0) c_i^n c_i^n + \Delta\tau \bar{N}_2 c_i^n s_i^n \tag{19d}$$

where I set $R = 0$, as mentioned. I got the diffusion behaviour familiar to the one from assignment 3, as seen in figure 1. This was done with $N = 500$, for $3 \cdot 10^5$ time steps, with $\Delta\xi = \frac{1}{x_c}\frac{L}{N-1}$ and $\Delta\tau = \frac{1}{t_c}\frac{L^2}{6D(N-1)^2}$. The functions even out with time, becoming straighter and straighter lines between their boundary conditions.

Out of curiosity, after noticing that the intersection point between a and b kept moving towards the left, I decided to plot its position as a function of time. Because both gasses are turned into c at the same rate, I expect that the region of overlap between the two functions will gradually disappear once the reaction term is introduced. The intersection point of the two functions is then probably the place where c is produced with highest rate, which I am sure is related to the formation of the plates.

As we can see in figure 2, the intersection point of the two functions is around the middle initially, at $\xi \approx 0.55$, and gradually moves towards lower and lower ξ. If the location where most of c is produced is moving along the ξ-axis, it seems reasonable that the location where s is formed also moves. The fact that s should be formed in plates, however, is still a bit strange.

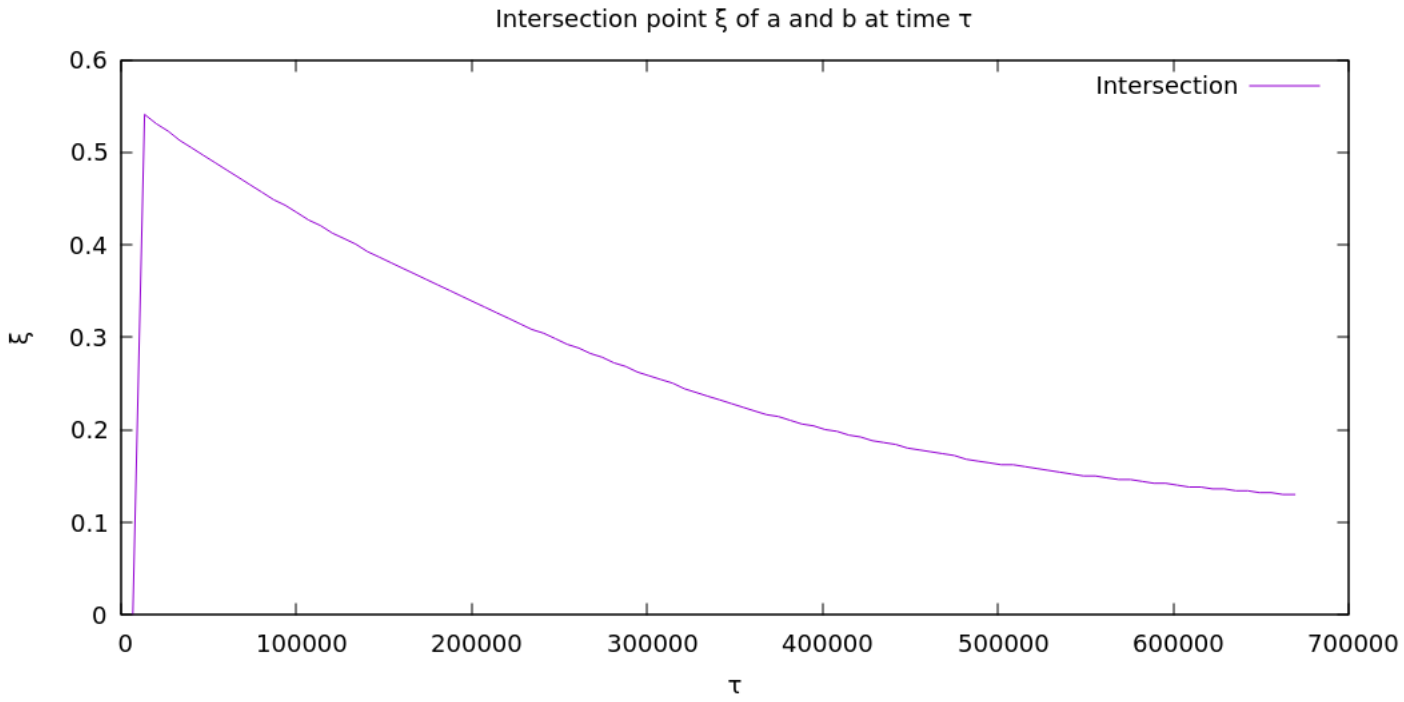

Figure 2: Intersection point between the a and b as a function of time.

2 Stability

I immediately got some stability issues when using a completely explicit scheme. At very few time steps, about enough for the concentration a to be barely nonzero towards the other end, I got fluctuations in c with negative and positive values around $x = L$, which only grew exponentially before the simulation failed.

My first suspicion was that I had a scaling problem, so I went over my dimensionality analysis looking for an error. Unfortunately I could not find one. I did however realize that I had not been careful enough in when formulating a stability condition. The fluctuations involved negative values in the concentrations, which is not physically meaningful. I therefore looked for places where this negative value might have arisen.

If we look more thoroughly at the main diagonal of the matrix scheme detailed above, we can formulate a requirement to make sure that no negative values crop up. The reaction terms involving $\Delta\tau\bar{R}g_i^n$ have the potential to be massive compared to $1 - 2\frac{\bar{D}_g\Delta\tau}{\Delta\xi^2}g_i^n$, and so their negative sign could easily become a problem for stability. If we want the main diagonal on the right-hand-side to be positive, we have to require that

$$0 \leq 1 - 2\frac{\bar{D}_g\Delta\tau}{\Delta\xi^2} - \Delta\tau\bar{R}\bar{b}_0$$

Another motivation for this inequality is that, in order for the new value a_i^{n+1} not to be negative, we must have

$$0 \leq a_i^{n+1} \;=\; a_i^n + \frac{\bar{D}_a\Delta\tau}{\Delta\xi^2}(a_{i-1}^n - 2a_i^n + a_{i+1}^n) - \Delta\tau\bar{R}a_i^n b_i^n$$

In the worst thinkable situation, the positive terms a_{i+1}^n and a_{i-1}^n would both be zero, and b_i^n would be its maximal value b_0, so

$$0 \;\leq\; a_i^{n+1} \;\leq\; a_i^n - \frac{\bar{D}_a\Delta\tau}{\Delta\xi^2}2a_i^n - \Delta\tau\bar{R}a_i^n\bar{b}_0 \;=\; a_i^n\left(1 - \frac{2\bar{D}_a\Delta\tau}{\Delta\xi^2} - \Delta\tau\bar{R}\bar{b}_0\right)$$

and to guarantee that this is satisfied we need

$$0 \;\leq\; \left(1 - \frac{2\bar{D}_a\Delta\tau}{\Delta\xi^2} - \Delta\tau\bar{R}\bar{b}_0\right)$$

which can be solved for $\Delta\xi$ as

$$\Delta\xi \geq \sqrt{\frac{2\bar{D}_g\Delta\tau}{1 - \Delta\tau\bar{R}\bar{b}_0}}$$

or for $\Delta\tau$,

$$\Delta\tau \leq \left(\frac{2\bar{D}_g}{\Delta\xi^2} + \bar{R}\bar{b}_0\right)^{-1}$$

Because $\Delta\xi$ is decided by N, L and the length scale x_c, it is most convenient to use the bound on $\Delta\tau$ in order to ensure stability. It can also be written using dimensionful quantities, as

$$\Delta\tau \leq \frac{1}{t_c}\left(\frac{2D_g}{(x_c\Delta\xi)^2} + Rb_0\right)^{-1} = \frac{1}{t_c}\left(\frac{2D_g}{L^2}(N-1)^2 + Rb_0\right)^{-1} \tag{20}$$

This bound seemed to be sufficient to get stability, and the fact that it simplifies to the old CFL-condition when $R = 0$ is a good indication. We could also write such an inequality from the part of our scheme for c, to get

$$\Delta\tau \leq \frac{1}{t_c}\left(\frac{2D_g}{L^2}(N-1)^2 + N_1c_0 + N_2S_0\right)^{-1}$$

where S_0 is the largest concentration of s to be expected. This S_0 is unknown initially, but in order for this upper bound to be more pessimistic that the one in (20), it would have to grow something of order 100, which seems unreasonable at this point.

I use the bound in Eq. (20) to define $\Delta\tau$ by some factor $\rho < 1$,

$$\Delta\tau \equiv \rho\left(\frac{2\bar{D}_g}{\Delta\xi^2} + \bar{R}\bar{b}_0\right)^{-1}, \qquad \text{CFL} < 1 \tag{21}$$

Typically I would use $\rho = 1/3$.

However, this bound on the time step length is very small compared to the standard CFL-criterion for the systems we want to study. We have $Rb_0 = 10$ and $2D_g(N-1)^2/L^2 \leq 8 \cdot 10^{-7}(N-1)^2$, which for $N \lesssim 10^3$ means the reaction term is the biggest restriction on our time steps. That means a lot of iterations to get to larger time scales, where the diffused gasses have progressed far into the tube from either end.

This estimate is reflected in the numerical experiments. I stopped getting failures in my explicit scheme once I used the bound in (20), so I have not found any reason to distrust it. But the diffusion went by very slowly in the purely explicit implementation.

Because I at first ran simulations with quite low N, about $N = 200$, this seemed like a major obstacle to be overcome. Below I show that phrasing our scheme semi-implicitly and getting rid of this reaction term in the time-step bound will be good for the running time. However, once we start looking at higher resolutions, in order to study the more precise distances between the plates, this point is rendered moot. I somewhat regret spending all my time writing nifty matrix schemes in Python just so I could use the half-implicit (and arguably half-baked) scheme I cooked up. On the third day I wrote everything over again in C, without any matrix solvers. Most of my plots are from that far simpler program.

To see how the time step will affect the running time, recall that the diffusion time τ for a gas g to significantly propagate over a length $\bar{L}$ is of order

$$\tau \sim \frac{\bar{L}^2}{\bar{D}_g}$$

Using this along with the upper bound on $\Delta\tau$, we get an approximate number of time steps

$$N_t \sim \frac{\tau}{\Delta\tau} \sim \frac{\bar{L}^2}{\bar{D}_g\Delta\tau} > \frac{\bar{L}^2}{\bar{D}_g}\left(\frac{2\bar{D}_g}{\Delta\xi^2} + \bar{R}\bar{b}_0\right) = 2(N-1)^2 + \frac{\bar{L}^2\bar{R}\bar{b}_0}{\bar{D}_g}$$

When we insert the values of our experiment above, leaving N, we get the following bound for number of necessary iterations:

$$N_t \sim 2(N-1)^2 + 2.5 \cdot 10^6$$

That is a lot of time steps just in order for the reaction to really get going. After that, we would still have to do a number of iterations of similar order before seeing any evidence of nucleation. Getting rid of the reaction term would let us get to this part of the reaction in a time that only depends on N, which for low N seems like a huge pluss. Take a look at figure **??**where I compare the two. The implicit method got the same exact behaviour, yet using 10000 times less time steps.

Hopefully it is understandable from this why I may have gotten carried away and too excited for this alternative scheme. When trying out my code I used low N, and it seemed like a clear winner. Because of that, I went "all in" in my implicit formulation, changing all of the negative nonlinear terms on the main diagonal into mixed terms in my scheme, like $R_a^n\mathbf{a}^{n+1}$, etc. I also defined all of the diffusion matrices A_g and B_g from the implicit Euler forms,

$$A_g = (I - M_g), \qquad B_g = I$$

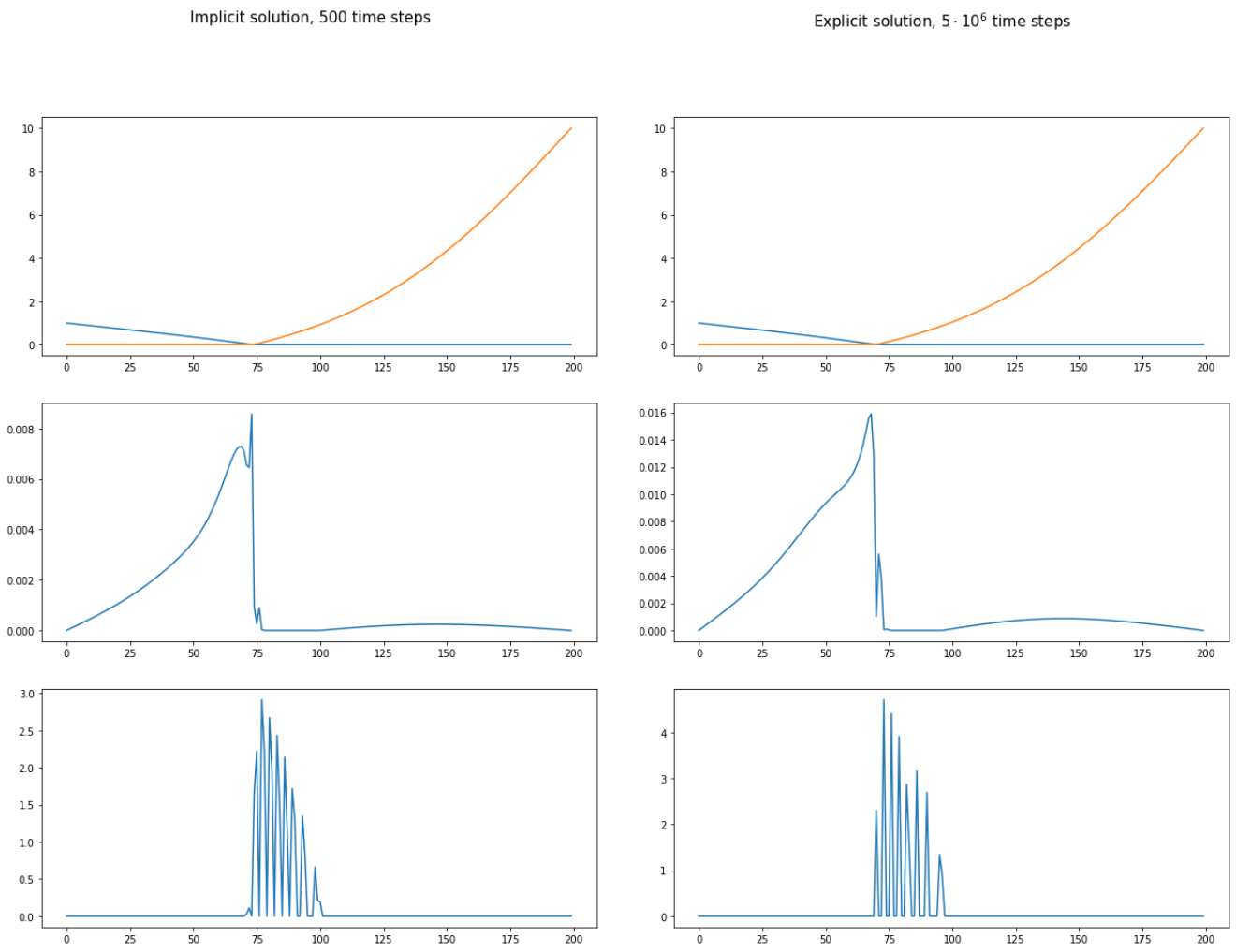

Figure 3: Comparison between the explicit and implicit methods I implemented in Python. The reason for the low resolution was because the explicit was so slow. The experiments seem to have reached about the same point in time, yet the implicit method has done so in 500 steps rather than 5000000, as was necessary for the exlicit.

So that the complete scheme looked like

$$\begin{pmatrix} I - M_a + R_a^n & & & \\ & I - M_b + R_b^n & & \\ & & I - M_c + N_1^n + N_{2c}^n & \\ & & & I \end{pmatrix} \begin{pmatrix} \mathbf{a}^{n+1} \\ \mathbf{b}^{n+1} \\ \mathbf{c}^{n+1} \\ \mathbf{s}^{n+1} \end{pmatrix} = \begin{pmatrix} I & & & \\ & I & & \\ R_b^n & & I & \\ & & N_1^n & I + N_{2s}^n \end{pmatrix} \begin{pmatrix} \mathbf{a}^n \\ \mathbf{b}^n \\ \mathbf{c}^n \\ \mathbf{s}^n \end{pmatrix} \tag{22}$$

Note that I left some of the non-linear terms completely explicit, either because they were not on the main diagonal, or because they were not negative in value. (In hindsight I realize that this might come off as rather ad hoc, but I argue below that this scheme should at least not obviously fail)

With this in place, there was no real bound on $\Delta\tau$ for the sake of stability (at least not for constant D ... Making D a functional will make it much harder to argue for this "implicit" scheme), and I was able to get into the times I wanted in hardly any time-steps at all.

And I have to say, despite how crazy this scheme may look on paper, the results were not bad. They looked exactly like my explicit scheme came up with, yet did so in orders of magnitude fewer iterations, because there was no limit to how large I could make my time steps. So changing the time and space steps $\Delta\tau$ and $\Delta\xi$ evidently had no great impact on the results, once scaled appropriately. Obviously, for greater precision in the sheet-to-sheet distances the resolution had to be larger, but the behaviour was the same.

However, for e.g. $N = 4000$, we get

$$N_t \sim 3.2 \cdot 10^7 + 2.5 \cdot 10^7$$

at which point it is clear that N is soon becoming the leading term in the necessary number of time steps. I did not use that high values for N, my highest was 2000, but still the point stands.

So I left this program after a while. In the end, when I saw that the diffusion matrices would soon depend on $\mathbf{s}^n$, I realized I had to keep that part of the scheme explicit. What I started using instead was a scheme where the diffusion matrices were of explicit type, but the implementation of the negative reaction and nucleation terms on the main diagonal were half-implicit. For sizeable N upto 2000, this still left me with a much larger time step than a purely explicit method would have needed, comparing the CFL chriterion with the upper bound I explained above.

Now I want to defend this way of handling the nonlinear terms a little bit. Implicit methods in general are useful precisely in situations such as this, where the restriction on the time step of the explicit method are impractical on the length scales we want to study.[1, section on diffusion] There are virtually no fluctuations to be lost in a and b, and they are at least slow moving in c, so it is not likely that we will have too much trouble with the loss of precision that is known to accompany implicit methods. In fact, because the values of a and b change so slowly, the error in approximating b^{n+1} by b^n is not likely to be very large.

In a sense, the cross terms can even be argued to better handle the time-dichotomy of explicit and implicit methods. By using one of each times, t_n and t_{n+1}, one for each concentration factor, the end product can be thought of as belonging to a time in between, like $t_{n+1/2}$.

Furthermore, a big issue with explicit methods is their tendency to amplify errors and noise, which only gets worse the more time steps are involved. As outlined above, "having to perform a large number of time steps" happens to be exactly the situation we are in. The error we would accumulate after several million time steps may very well be worse than any slight loss of precision in an implicit implementation.

The reason I in the end phrased the diffusion parts explicitly was to be able to define the diffusion coefficient as a functional of $s(x,t)$. In that situation, the matrices A_g and B_g defined from standard diffusion behaviour, become dependant of n as well. Specifically,

$$A_g^n = A_g(\mathbf{s}^n), \qquad B_g^n = A_g(\mathbf{s}^n)$$

because

$$M_g^n = M_g(\mathbf{s}^n).$$

In these cases, it will be necessary to let all of the $\partial_\xi(\bar{D}_g \partial_\xi g)$ approximations be explicit, because $\bar{D}_g^n$ will depend on $\mathbf{s}^n$, which is going to fluctuate wildly. Therefore the diffusion coefficients may fluctuate wildly as well, and it is unsafe to use previous values to approximate the next ones directly, unlike a_i^n compared to a_i^{n+1}.

I rewrote the entire thing in C after I realized I would need to use an explicit scheme for the diffusion terms anyway in the case of a functional D. But this did not prevent me from trying out the cross terms, $R_a^n \mathbf{a}^{n+1}$ and so on. The scheme I chose to implement could be written in matrices as

$$\begin{pmatrix} I + R_a^n & & & \\ & I + R_b^n & & \\ & & I + N_1^n + N_{2c}^n & \\ & & & I \end{pmatrix} \begin{pmatrix} \mathbf{a}^{n+1} \\ \mathbf{b}^{n+1} \\ \mathbf{c}^{n+1} \\ \mathbf{s}^{n+1} \end{pmatrix} = \begin{pmatrix} I + M_a & & & \\ & I + M_b & & \\ R_b^n & & I + M_c & \\ & & N_1^n & I + N_{2s}^n \end{pmatrix} \begin{pmatrix} \mathbf{a}^n \\ \mathbf{b}^n \\ \mathbf{c}^n \\ \mathbf{s}^n \end{pmatrix} \tag{23}$$

and because the matrix on the left only has elements along the main diagonal, this can be written in component form:

$$a_i^{n+1} = \left(a_i^n + \frac{\bar{D}_a \Delta\tau}{\Delta\xi^2}(a_{i-1}^n - 2a_i^n + a_{i+1}^n)\right)\left(1 + \Delta\tau \bar{R} b_i^n\right)^{-1} \tag{24a}$$

$$b_i^{n+1} = \left(b_i^n + \frac{\bar{D}_b \Delta\tau}{\Delta\xi^2}(b_{i-1}^n - 2b_i^n + b_{i+1}^n)\right)\left(1 + \Delta\tau \bar{R} a_i^n\right)^{-1} \tag{24b}$$

$$c_i^{n+1} = \left(c_i^n + \frac{\bar{D}_c \Delta\tau}{\Delta\xi^2}(c_{i-1}^n - 2c_i^n + c_{i+1}^n) + \Delta\tau \bar{R} a_i^n b_i^n\right)\left(1 + \Delta\tau \bar{N}_1 \theta(c_i^n - c_0)c_i^n + \Delta\tau \bar{N}_2 s_i^n\right)^{-1} \tag{24c}$$

$$s_i^{n+1} = s_i^n + \Delta\tau \bar{N}_1 \theta(c_i^n - c_0)c_i^n c_i^n + \Delta\tau \bar{N}_2 c_i^n s_i^n \tag{24d}$$

Note that the stability requirement for $\Delta\tau$ using this scheme is only concerned with the diffusion part, so I could use the higher CFL lower bound for the time step,

$$\Delta\tau < \frac{\Delta\xi^2}{2\bar{D}}$$

which for the discretizations N I was using gave me results far more quickly. The combination of using a quicker language combined with this quite quick scheme made a world of difference from the explicit implementation in Python.

I ran it once at $N = 2000$, with $\Delta\xi = \frac{\bar{L}}{N-1}$ and $\Delta\tau = \frac{\Delta\xi^2}{6\bar{D}_g}$, and got the results in figure 4 for the four concentrations, which clearly shows the sheets that we are searching for.

3 Physical mechanism

I now went on to investigate how the sheet spacing varies. The sheet spacing was found by going over the final vector $\mathbf{s}$, and checking for a leap of values above a low value, s_{low}. I used $s_{low} = 0.01$, and scanned the array from left to right, for a leap up through this value. That is, I scanned for whenever $((s_i < s_{low})\ and\ (s_{i+1} > s_{low}))$, and recorded that i, or rather $x_n = i_n \Delta\xi$, as the position of the corresponding sheet.

Because the width of the sheets vary, we have to make up our minds about from where to measure the inter-sheet distance. I opted for simply using the left edge of each sheet as the point to be measured, as i felt that was the least ambiguous, and because, as will be discussed later, most of the sheet is concentrated on the left side.

I also decided to record the time at which all the sheets first appeared, t_n. The notation here gets messy, because our numbering of the positions, with $x_n < x_{n+1}$, is contrary to when they actually appear. To remove potential for confusion, I will instead let $t_n < t_{n+1}$, and time t_n correspond to sheet at position x_n.

This way, it becomes much clearer that the inter-sheet distance $\Delta\xi_n = x_{n+1} - x_n$ in fact decreases with time. This is also very clear from the graph of the final $\mathbf{s}$.

I wanted to test my hypothesis from before, that the intersection between the functions a and b are important to understand the phenomenon in question. So, I plotted the appearence times against the positions for the sheets in the same plot as the recorded intersection points. This is shown in figure 6.

The fit is really good! It seems that the intersection of a and b, where in the end c-production is at a maximum, *leads* the sheets. **It is as if there is a source of c moving through space, leaving a trail of substance s behind it.**

And that I think is exactly what is going on. The nature of this trail being left behind will depend on the various constants involved. For some combinations of $D_g, R, N_i, and g_0$, the trail might be a continuous surface. For others, there may be one single spot of the solid s, where all of c accumulates, regardless of where it is produced.

The crucial thing for sheets to be formed is that the concentration c is able to reach c_0 again, in a point far enough away from the previous sheet. The nucleation rate must also be large enough so that, immediately once a bit of s is formed, it starts eating up all the gas c around it.

I want to dwell a bit more on the point like source of c that results from the two functions a and b. I think it will be point like because, after a while, the thin strips of a that stretches into the region dominated by b is eaten up and turned into c. Because the reaction rate is so high compared to the diffusion process, the two gasses will disappear more quickly than they can propagate into the "enemy territory". The result is that the only place where a and b come into contact is in the place where they meet. This is seen visually in figure 7

As for the exact moment that a sheet is formed, there is also something worth saying. A lot can be learned from just looking at how c changes the moment it pushes up above c_0, shown in figure 8. The bulb-like shape seems to burst, creating a sharp valley, and where the sheet starts to form we see the concentration c rapidly decrease to zero. The width of the sheets ought to depend on the nucleation rates, because they determine how far up above c_0 the concentration c manages to grow before collapsing. If the reaction rate is high enough, and the diffusivity of a and b is high eough that they can propagate to eachother quckly enough and react, the sheets might never be able to deplete c, and we end up with a big slab of s instead of sheets. **The diffusivities must be small enough that this does not happen.**

(Some more analysis of what might happen if we increase or decrease certain values)

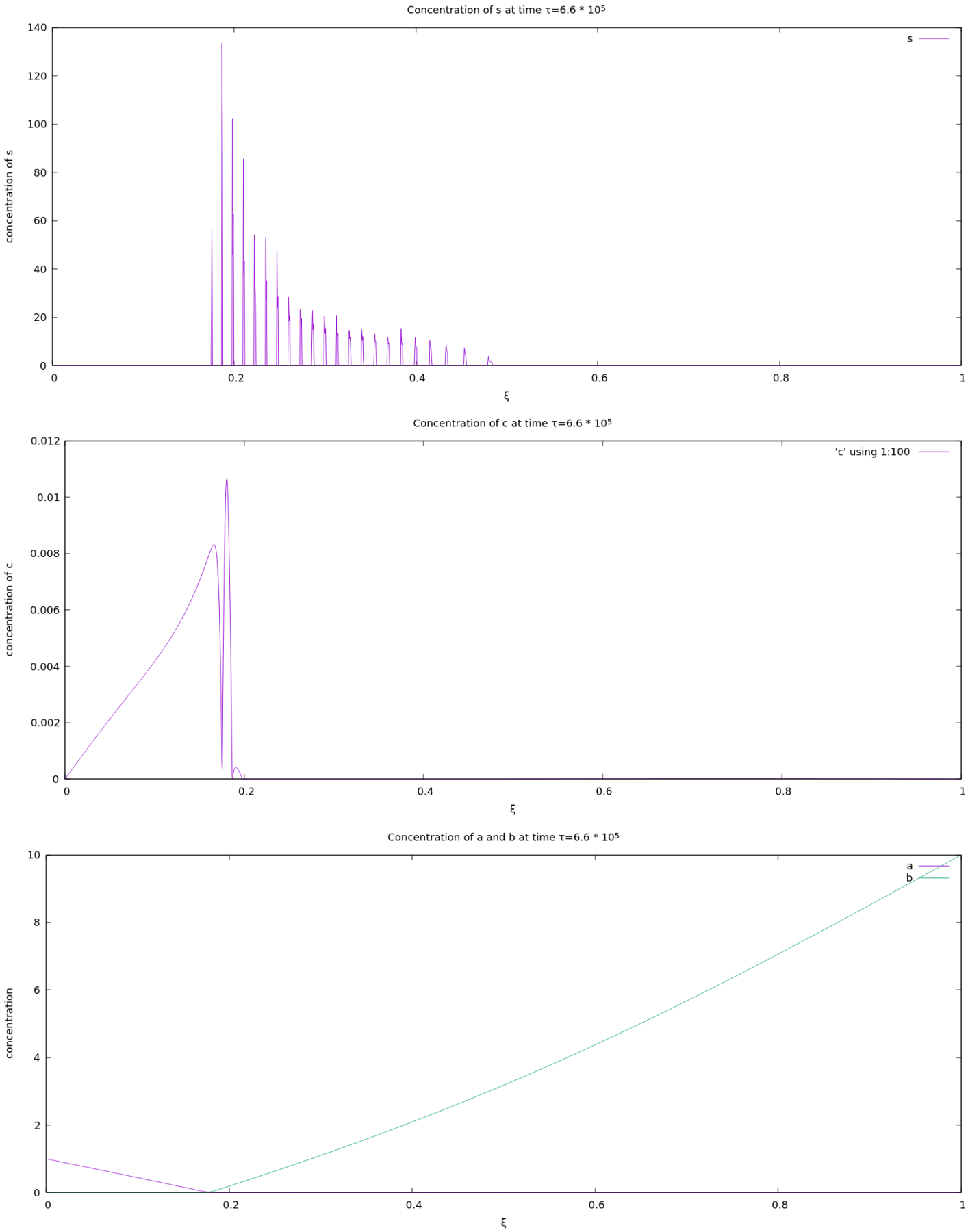

Figure 4: The final concentrations of a, b, c, and s for $N = 2000$ and $\Delta\xi = \frac{\bar{L}}{N-1}$ and $\Delta\tau = \frac{\Delta\xi^2}{6\bar{D}_g}$ at time $\tau = 6.6 \cdot 10^6$

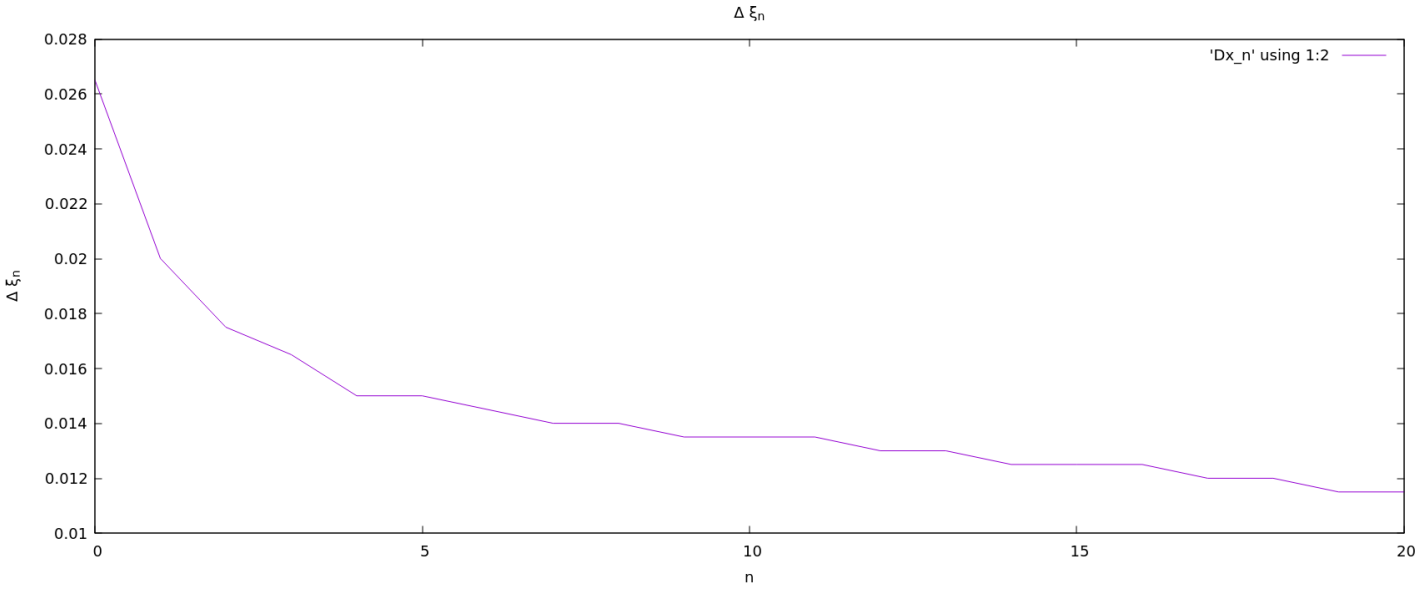

Figure 5: The sheet distance $\Delta\xi_n$, numbered here from first to last in time. They are growing closer and closer together.

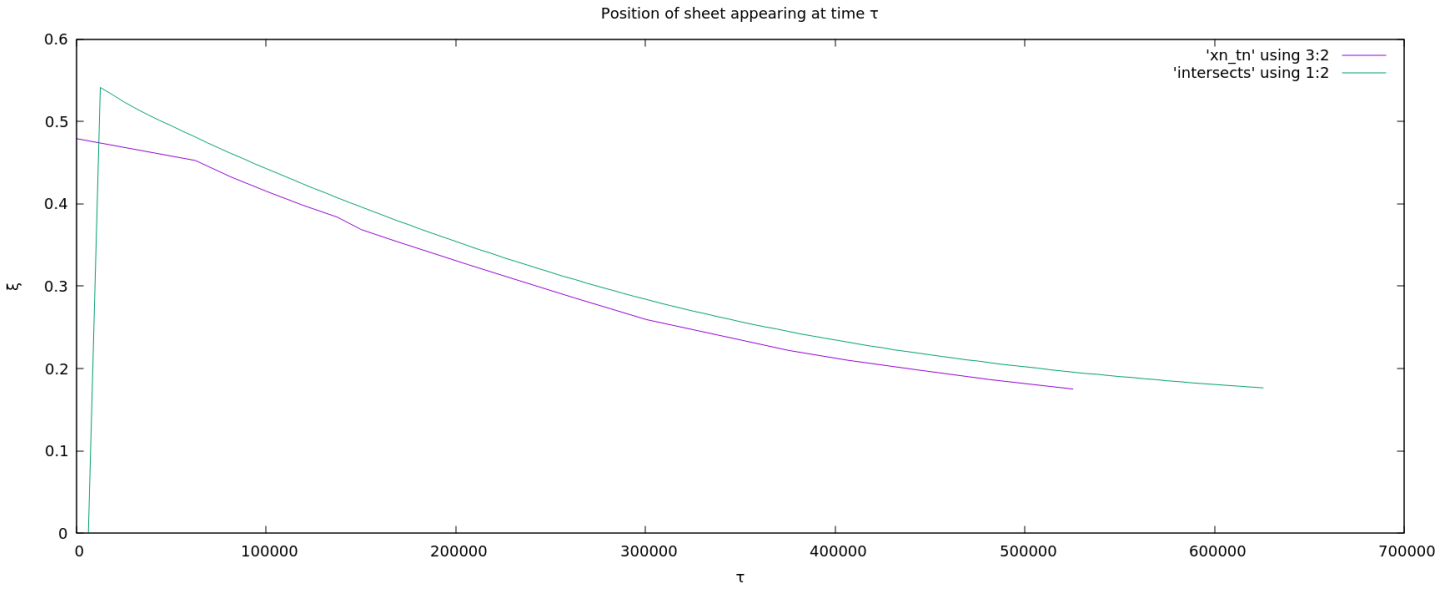

Figure 6: The position and time at which a sheet first appeared, in purple, and the place where the concentrations a and b meet, in blue. The fit shows that the a, b intersection leads the sheet formation, leaving it as a trail. The jump at the start of the intersection line is due to zero values making it unclear exactly where they intersect.

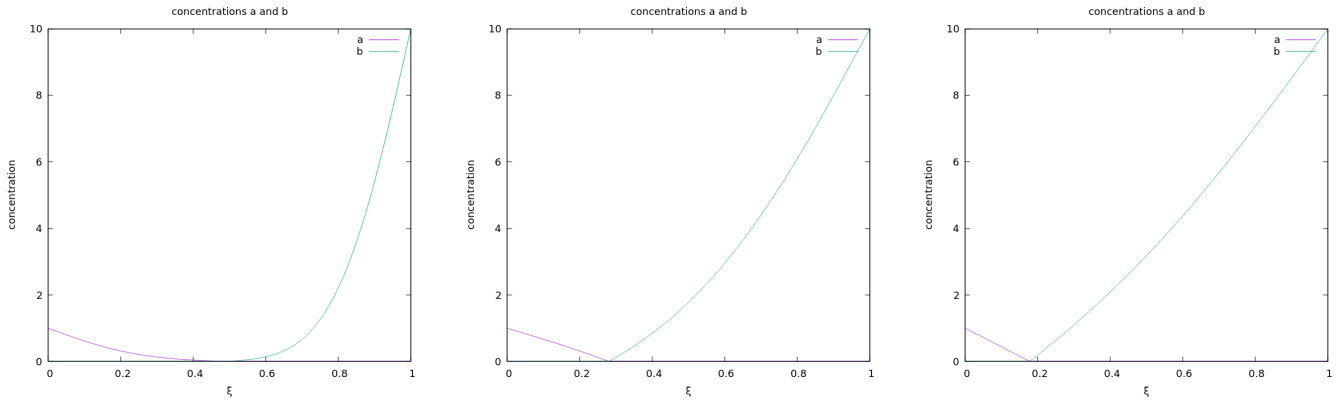

Figure 7: Clearer illustration of where a and b meet when $R = 1.0$.

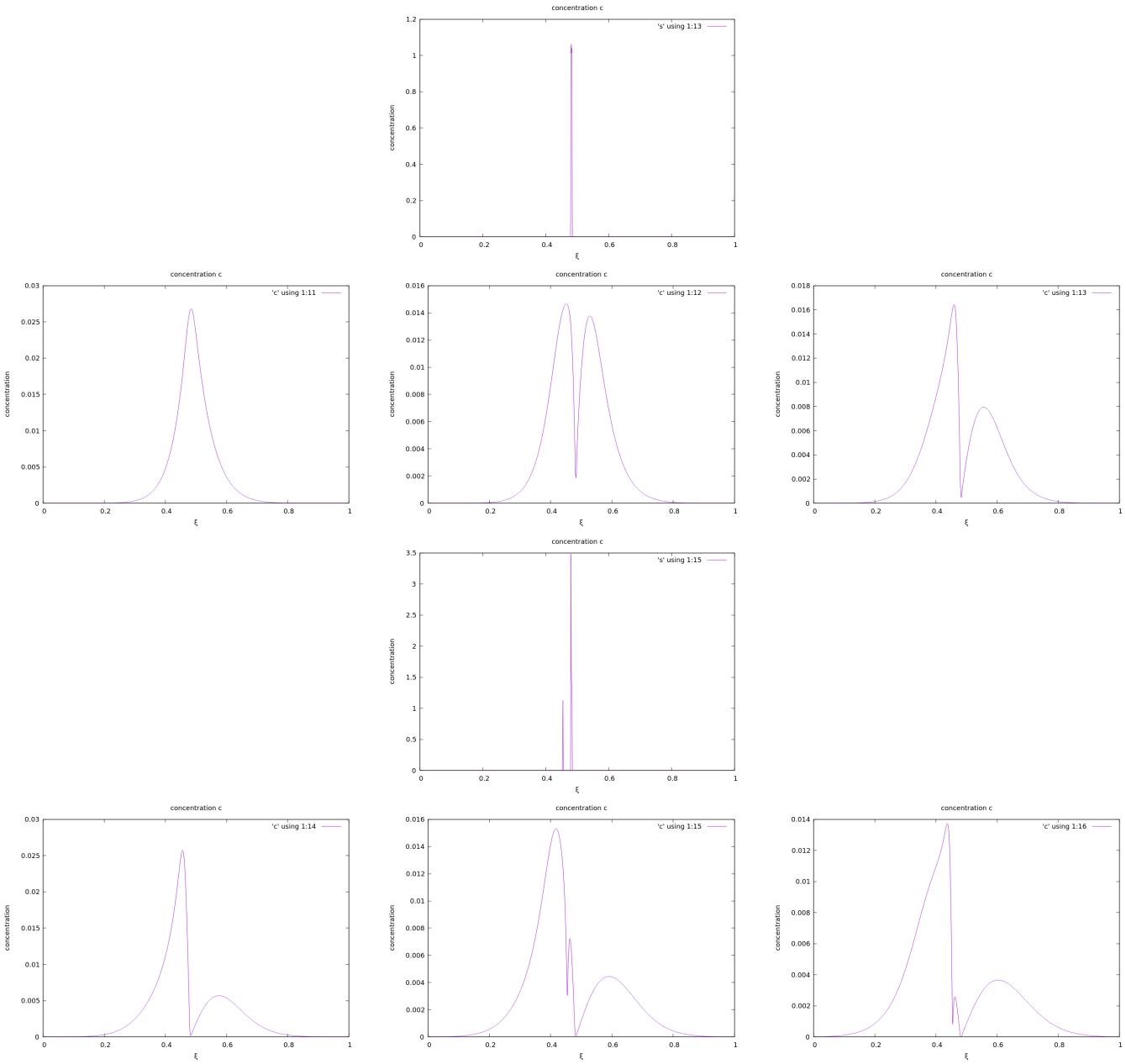

Figure 8: Illustration of how a sheet is formed. On top is a graph of s immediately after the formation of the first sheet. The function c bulges up, but once it reaches c_0 it bursts, collapsing into the newly formed layer of s, and only one of the two resulting halves get replenished by the location where a and b meet. The half that gets replenished ballons up again, repeating the process once it reaches c_0, demonstrated in the next four plots.

Afterwards, the concentration c is divided into two halves. One will be replenished by the point like source described above, while the other is gradually eaten up by the b reservoir and newly formed sheet. Because the sheet eats up all of c that it comes into contact with, it is very difficult for the other half to gain more c afterwards.

A sign that this is a correct interpretation is found in the actual shape of the sheets, when plotting s. They have spikes on each side, because once c goes to zero above the newly formed spike, the gas barely comes into contact with the center of the sheet. It only touches the edges, which then end up being built up way above the center of the sheet.

Then, as the left side of c grows larger again from the point like source, it creates a bulb again, and once its peak reaches c_0 the process starts again. Now, however, there will be a pocket of c trapped between two sheets, and it is eaten up by both of them.

This seems to repeat indefinitely, but obviously that is not possible with a finite length tube. The distance between sheets appears to reach a steady value after some time, but it is decreasing. We know that the intersection between a and b eventually must stop moving, either at $\xi = 0$ or at some other position, at which point no more sheets will be formed.

Note that the formation of c does not impede the propagation of a and b directly. All that is needed for that is the reaction $a + b \to c$, to remove their overlap. But next we might consider what happens the moment we let s affect the diffusivity of all three gasses.

4 D as a functional of s

What might we expect to see? Well, if there suddenly appears a barrier in the middle of the tube, the point like source from a and b might no longer appear. If b is not able to get through the sheets easily enough that more c is formed quickly enough, the critical concentration c_0 may never be reached.

This is modelled by changing the diffusion coefficient to

$$D_{g,new}[s(x,t)] \equiv \frac{D_g}{1 + \frac{s(x,t)}{s_0}}$$

which will decrease considerably whenever s is of order the threshold s_0.

I expect that, if the value of s_0 is too low, we will maybe see a couple of sheets forming, but once enough barriers have been placed between the a and b reservoirs, there will be little transfer from one side to the other, and thus little to no c will be produced. Whatever amount of c was present at the time when these sheets were formed will slowly diffuse away, either out into the reservoirs of a and b, or precipitate onto the sheets, making matters worse.

We have seen that the sheets get thicker and thicker one after the other, so whatever we put as a threshold s_0 it will be hit eventually.

In the end, the sheets will behave almost like reflective boundaries to the chemicals a and b, though maybe not perfectly reflective. So for really high time I expect to see the two concentrations flattening out in the tube, becoming constant in each their halves, and zero in the other half.

All of this is shown in figure 9, in which the equations were solved using $s_0 = 1.0$ and 5.0. The higher value of s_0 permits flow of gasses for a while, until a sheet appears that is big enough to halt the process.

What surprises me is that there is no c at all in between the sheets, in the case of multiple ones for $s_0 = 5.0$, yet the corners of the concentrations a and b appear to touch. This may indicate that they do propagate, slowly, through the sheets, and the moment they meet and produce c it turns into s.

5 Changing the reaction equation

For a reaction equation by

$$aA + bB \to pP + qQ$$

where capital letters indicate substance and minuscule letters are integer coefficients, the rate of reaction is related to the concentrations $[A]$ etc. by the equation [2]

$$r = -\frac{1}{a}\frac{\mathrm{d}[A]}{\mathrm{d}t} = -\frac{1}{b}\frac{\mathrm{d}[B]}{\mathrm{d}t} = \frac{1}{p}\frac{\mathrm{d}[P]}{\mathrm{d}t} = \frac{1}{q}\frac{\mathrm{d}[Q]}{\mathrm{d}t} \tag{25}$$

or in our case, where

$$\alpha A + B \to C, \qquad a = 1 \text{ or } 2$$

we get

$$r = -\frac{1}{\alpha}\frac{\mathrm{d}[A]}{\mathrm{d}t} = -\frac{\mathrm{d}[B]}{\mathrm{d}t} = \frac{\mathrm{d}[C]}{\mathrm{d}t} \tag{26}$$

The reaction is related to the concentrations of the reactants by some power law, as given by the rate equation,

$$r = k[A]^x[B]^y$$

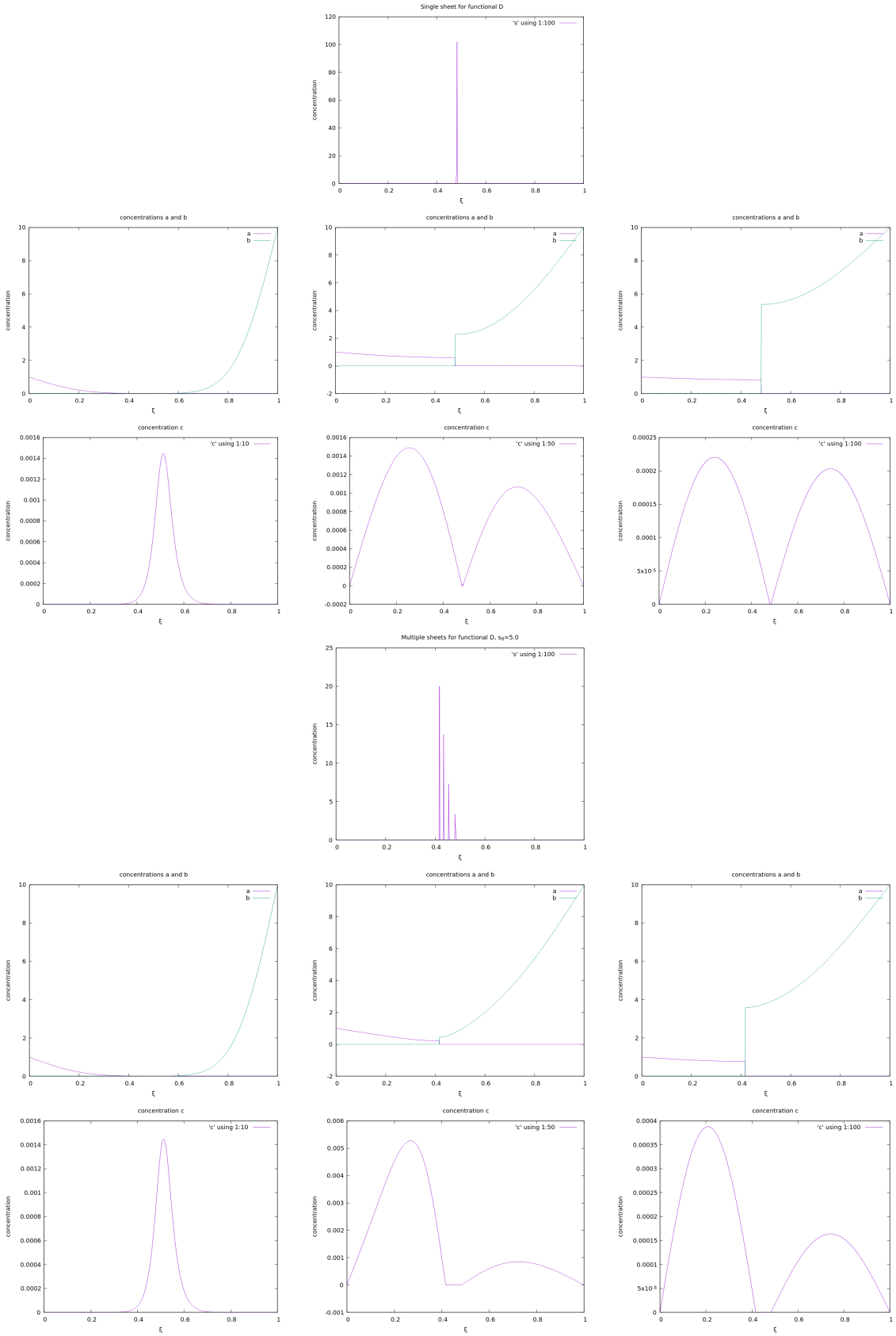

Figure 9: Illustrations of how when sheets are able to slow down the diffusion, the sheets that are formed end up blocking the gasses a and b, so that no more c is produced. Depending on s_0, being 1.0 in the figures on top and 5.0 in those on the bottom, there may be one or several sheets formed, but eventually one grows big enough to clog up the whole diffusion. As a result, the two gasses a and b fill their respective halves of the tube to concentrations equal to their reservoirs..

This is also what we have used in our equations, where the reaction rate $r = Rab$ is linear in both. As is also clear from the equations above, the time rate of change in concentration due to the reaction, writing g now instead of $[G]$, is for each

$$\frac{\partial a}{\partial t} = -\alpha Rab, \qquad \frac{\partial b}{\partial t} = -Rab, \qquad \frac{\partial c}{\partial t} = Rab$$

In other words, the only thing that changes is the number α in front of the reaction term in just one of the four equations. That should be simple enough to implement.

This could also have been understood from the equations we started out with alone. If we change the reaction to $2a+b \to c$, yet somehow have the same reaction coefficient R, the only thing to change would be that we lose twise as much a from reactions as we lose b or gain c.

Seen from a different perspective, if we make $\alpha = 2$ instead of 1, it takes much less b to eat up the a. The analysis from earlier must then be modified slightly, where I explained why plates form using the intersection point between the no-reaction solution to a and b. The overlapping area of the functions a and b no longer represents all of what will be converted into c. Rather, we should be looking at the intersection of the functions a and $2b$, because their overlapping area represents the amount of a that will be used up.

The implication is quite clear. The "wave" of gas b rolling in will more efficiently eat up the gas a, so the process will essentially speed up. Whether this means that the sheet spacing increases, or whether we just get more sheets more quickly, I am not sure of at the time of writing. But if I had to guess I would say that the sheet distances get longer. There is less gas being produced, and the point-like source is moving more quickly, but the simulations will give us the real answer. These are presented in figure

References

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery. *Numerical Recipes - The Art of Scientific Computing; Third Edition*. Cambridge University Press 2017.

[2] Wikipedia - *Reaction rate*,
`https://en.wikipedia.org/wiki/Reaction_rate`

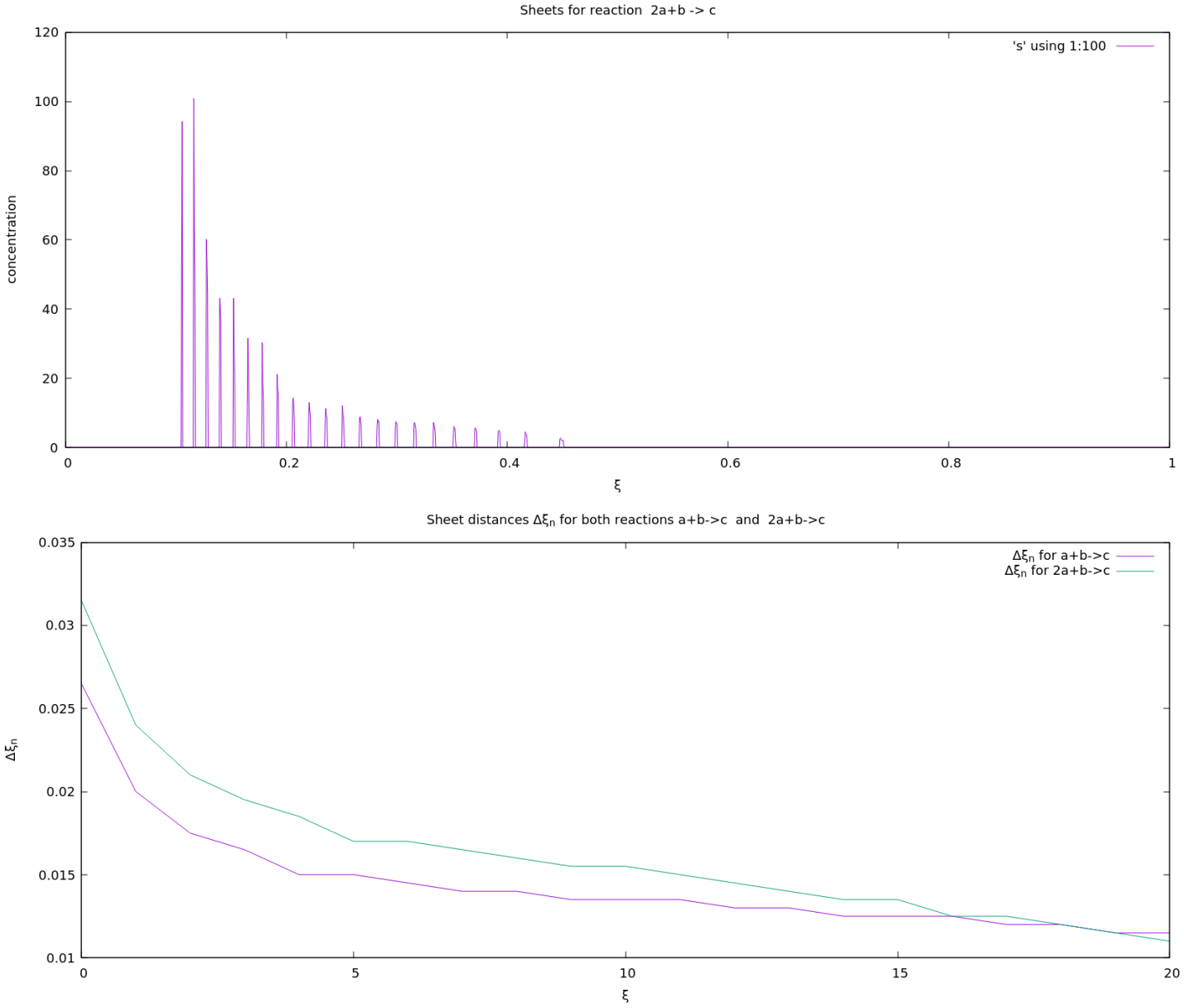

Figure 10: Top: the sheets formed from a reaction equation of the alternative form, $2a + b \to c$. They do appear more spaced out than in the experiments we have been running so far, and the bottom plot confirms this, where $\Delta\xi_n$ is plotted for the two reactions. Evidently the factor 2 speeds up the drift of the moving border between a and b, as the gas a is consumed at twice the rate. As a result the sheet distances are larger, and end up further towards $\xi = 0$.

Appendix A - C program

I include the program I wrote in C, as it was the most useful in the end. I have handed in my python scripts among the files included, but there are dragons lurking in that code, so this is better for seeing what I did.

I am not very comfortable using C yet, but this did the job, and I hope I did not offend anyone with any weird choices in this, for me, new language.

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>

double theta(double x);
void solve_save(double xc, double tc, int N, double L, double Da0, double Db0, double Dc0, double R,
    double N1, double N2, double a0, double b0, double c0, double s0, int t_steps, int t_inter, bool D_var
    , int a_k);


//=============================================================================
//      M A I N
//=============================================================================
void main(void)
{
    // Units xc, tc scale all other quantities.
    double xc = 1.0, tc = 1.0;
    // Discretization
    int N = 2000;
    int t_steps = 6000000;
    int t_inter = t_steps/100;
    // Parameters made unitless
    double L = (1.0/xc)     * 1.0;
    double D = (tc/(xc*xc)) * 4.0e-7;
    double R = (tc/xc)      * 1.0;
    double g0 = (xc)        * 1.0;
    // Quantities scaled from the above
    double Da = 1.0*D, Db = (2.0/3.0)*D, Dc = (8.0/15.0)*D;
    double N1 = 0.1*R, N2 = 0.1*R;
    double a0 = 1.0*g0, b0 = 10.0*g0, c0 = 3.0e-2*g0, s0 = 1.0*g0;
    // Let D vary?
    bool D_var = false;
    // Coeff of (a) in reaction    a_k*a + b  ->  c
    int a_k = 2;

    // Solve the system and save to file.
    clock_t start = clock();
    solve_save(xc,tc,N,L,Da,Db,Dc,R,N1,N2,a0,b0,c0,s0,t_steps,t_inter,D_var,a_k);
    clock_t end = clock();
    printf("Time taken: %f\n", (double)(end - start)/CLOCKS_PER_SEC);
}


//=============================================================================
// Solve and save
//=============================================================================
void solve_save(double xc, double tc, int N, double L, double Da0, double Db0, double Dc0, double R,
    double N1, double N2, double a0, double b0, double c0, double s0, int t_steps, int t_inter, bool D_var
    , int a_k)
{
    // Compute discretization steps, expressions explained in report.
    double dx = L / (N-1);
    //double dt = (1.0/3.0) / ((2*Da0*(N-1)*(N-1)/(L*L)) + R*b0);
    double dt = (1.0/6.0)*L*L/(Da0*(N-1)*(N-1));

    // The vectors being iterated over.
    // There are two of each copy. Explained below.
    double a1[N], a2[N], b1[N], b2[N], c1[N], c2[N], s1[N], s2[N];
    double Da[N], Db[N], Dc[N];

    int i,j,n;
    int N_save = t_steps/t_inter; // = 100 usually is what I use

    double a_list[N_save][N];
    double b_list[N_save][N];
    double c_list[N_save][N];
    double s_list[N_save][N];
```

```
    // I record the intersection points of  a  and  b
    int intersects[N_save];
    double dintersects[N_save];
    // Locations of sheets
    int x_n[N];
    int Dx_n[N];
    // Appearence times
    int x_prev = N;
    double appearence_time[N];
    int peak_n = 0;

    FILE * fp;
    char fname[30];

    // Initial conditions:
    for (i=0; i<N; i++) {
        Da[i] = Da0;
        Db[i] = Db0;
        Dc[i] = Dc0;
        a1[i] = 0.0; // Redundant, but eh
        b1[i] = 0.0;
        c1[i] = 0.0;
        s1[i] = 0.0;
        a2[i] = 0.0;
        b2[i] = 0.0;
        c2[i] = 0.0;
        s2[i] = 0.0;
    }
    a1[0] = a0;
    a2[0] = a0;
    b1[N-1] = b0;
    b2[N-1] = b0;

    //-------------------------------------------------------------------------
    // The scheme being applied
    //
    for (n=0; n<t_steps/2; n++) {
        // Make the diffusion coefficients
        if (D_var) {
            for (i=0; i<N; i++) {
                Da[i] = Da0 / (1.0 + s1[i]/s0);
                Db[i] = Db0 / (1.0 + s1[i]/s0);
                Dc[i] = Dc0 / (1.0 + s1[i]/s0);
            }
        }
        // I use two versions of all four vectors.
        // I don't want to overwrite a1 before I have found s1.
        // This was the simplest way I could think of that would
        // not require one empty run to copy g2 onto g1.
        //
        // There are probably more proper ways to do this, with
        // pointer mess or something, but I am not all that
        // comfortable in C.
        //
        // g2 = g2(a1,b1,c1,s1)
        for (i=1; i<N-1; i++) {
            a2[i] = ( a1[i] + (dt/(4*dx*dx))*(
                          (Da[i-1] + 4*Da[i] - Da[i+1])*a1[i-1]
                        - 8*Da[i]*a1[i]
                        + (Da[i+1] + 4*Da[i] - Da[i-1])*a1[i+1]
                        //) - dt*R*a1[i]*b1[i] );
                        ))/(1.0 + a_k*dt*R*b1[i] );
            b2[i] = ( b1[i] + (dt/(4*dx*dx))*(
                          (Db[i-1] + 4*Db[i] - Db[i+1])*b1[i-1]
                        - 8*Db[i]*b1[i]
                        + (Db[i+1] + 4*Db[i] - Db[i-1])*b1[i+1]
                        //) - dt*R*a1[i]*b1[i] );
                        ))/(1.0 + dt*R*a1[i]);
            c2[i] = ( c1[i] + (dt/(4*dx*dx))*(
                          (Dc[i-1] + 4*Dc[i] - Dc[i+1])*c1[i-1]
                        - 8*Dc[i]*c1[i]
                        + (Dc[i+1] + 4*Dc[i] - Dc[i-1])*c1[i+1]
                        //) + dt*R*a1[i]*b1[i] - dt*N1*theta(c1[i]-c0)*c1[i]*c1[i]
                        //- dt*N2*c1[i]*s1[i] );
                        ) + dt*R*a1[i]*b1[i])/(1.0 + dt*N1*theta(c1[i]-c0)*c1[i]
                        + dt*N2*s1[i] );
```

```
            s2[i] = ( s1[i]  + dt*N1*theta(c1[i]-c0)*c1[i]*c1[i]
                             + dt*N2*c1[i]*s1[i] );
        }
        //
        //
        // Now do the same as above over again, except from
        // the vectors  g2  to the vectors  g1.
        //
        //
        // Make the diffusion coefficients
        if (D_var) {
            for (i=0; i<N; i++) {
                Da[i] = Da0 / (1.0 + s2[i]/s0);
                Db[i] = Db0 / (1.0 + s2[i]/s0);
                Dc[i] = Dc0 / (1.0 + s2[i]/s0);
            }
        }
        // g1 = g1(a2,b2,c2,s2)
        for (i=1; i<N-1; i++) {
            a1[i] = ( a2[i] + (dt/(4*dx*dx))*(
                              (Da[i-1] + 4*Da[i] - Da[i+1])*a2[i-1]
                            - 8*Da[i]*a2[i]
                            + (Da[i+1] + 4*Da[i] - Da[i-1])*a2[i+1]
                            //) - dt*R*a2[i]*b2[i] );
                            ))/(1.0 + a_k*dt*R*b2[i] );
            b1[i] = ( b2[i] + (dt/(4*dx*dx))*(
                              (Db[i-1] + 4*Db[i] - Db[i+1])*b2[i-1]
                            - 8*Db[i]*b2[i]
                            + (Db[i+1] + 4*Db[i] - Db[i-1])*b2[i+1]
                            //) - dt*R*a2[i]*b2[i] );
                            ))/(1.0 + dt*R*a2[i]);
            c1[i] = ( c2[i] + (dt/(4*dx*dx))*(
                              (Dc[i-1] + 4*Dc[i] - Dc[i+1])*c2[i-1]
                            - 8*Dc[i]*c2[i]
                            + (Dc[i+1] + 4*Dc[i] - Dc[i-1])*c2[i+1]
                            //) + dt*R*a2[i]*b2[i] - dt*N1*theta(c1[i]-c0)*c2[i]*c2[i]
                            //- dt*N2*c2[i]*s2[i] );
                            ) + dt*R*a2[i]*b2[i])/(1.0 + dt*N1*theta(c1[i]-c0)*c2[i]
                            + dt*N2*s2[i] );
            s1[i] = ( s2[i]  + dt*N1*theta(c2[i]-c0)*c2[i]*c2[i]
                             + dt*N2*c2[i]*s2[i] );
        }
        // Every (t_steps/t_inter) steps, store vectors for later,
        // and check for new sheets.
        if ((n+1)%(t_inter/2) == 0) {
            // Store all vectors:
            for (i=0; i<N; i++) {
                a_list[(n+1)/(t_inter/2)][i] = a1[i];
                b_list[(n+1)/(t_inter/2)][i] = b1[i];
                c_list[(n+1)/(t_inter/2)][i] = c1[i];
                s_list[(n+1)/(t_inter/2)][i] = s1[i];
                // Check if a new sheet has appeared
                if (s1[i] > 0.01) {
                    // Sometimes the neighbours also grow, a bit
                    // delayed. Make sure that is not the case.
                    if (i == x_prev-1) {
                        x_prev = x_prev - 1;
                    } else if (i < x_prev - 1) {
                        // count number of peaks
                        peak_n += 1;
                        appearence_time[peak_n] = 2*n*dt;
                        x_prev = i;
                        // I think I'll leave this.
                        printf("%d %d\n", i, n);
                    }
                }
            }
            // Find intersect between  a  and  b.
            i = 0;
            while (a1[i] > b1[i])
                i++;
            intersects[(n+1)/(t_inter/2)] = i; }
    }
    //-------------------------------------------------------------------
```

```
    //------------------------------------------------------------------------
    //
    // Write to file
    //------------------------------------------------------------------------
    //
    // I do this very carefully because copy/paste is cheap
    //
    // a
    sprintf(fname, "a");
    fp = fopen(fname, "w+");
    for (i=0; i<N; i++) {
        fprintf(fp, "%f ", dx*i);
        for (n=0; n<N_save; n++) {
            fprintf(fp, "%f ", a_list[n][i]);
        }
        fprintf(fp, "\n");
    }
    fclose(fp);
    //------------------------------------------------------------------------
    // b
    sprintf(fname, "b");
    fp = fopen(fname, "w+");
    for (i=0; i<N; i++) {
        fprintf(fp, "%f ", dx*i);
        for (n=0; n<N_save; n++) {
            fprintf(fp, "%f ", b_list[n][i]);
        }
        fprintf(fp, "\n");
    }
    fclose(fp);
    //------------------------------------------------------------------------
    // c
    sprintf(fname, "c");
    fp = fopen(fname, "w+");
    for (i=0; i<N; i++) {
        fprintf(fp, "%f ", dx*i);
        for (n=0; n<N_save; n++) {
            fprintf(fp, "%f ", c_list[n][i]);
        }
        fprintf(fp, "\n");
    }
    fclose(fp);
    //------------------------------------------------------------------------
    // s
    sprintf(fname, "s");
    fp = fopen(fname, "w+");
    for (i=0; i<N; i++) {
        fprintf(fp, "%f ", dx*i);
        for (n=0; n<N_save; n++) {
            fprintf(fp, "%f ", s_list[n][i]);
        }
        fprintf(fp, "\n");
    }
    fclose(fp);
    //------------------------------------------------------------------------
    // intersects
    sprintf(fname, "intersects");
    fp = fopen(fname, "w+");
    for (n=0; n<N_save; n++) {
        fprintf(fp, "%f %f\n", dt*t_inter*(n+1), dx*intersects[n]);
    }
    fclose(fp);
    //------------------------------------------------------------------------
    // derivative of intersects
    for (n=0; n<N_save-1; n++) {
        dintersects[n] = (dx/(dt*t_inter))*(intersects[n+1] - intersects[n]);
    }
    sprintf(fname, "dintersects");
    fp = fopen(fname, "w+");
    for (n=10; n<N_save-1; n++) {
        fprintf(fp, "%f %f\n", dt*t_inter*(n+1), dintersects[n]);
    }

    //------------------------------------------------------------------------
    // Now to calculate sheet distances:
    n = 0;
```

```
    for (i=1; i<N; i++) {
        if ((s1[i-1] < 0.01) && (s1[i] > 0.01)) {
            x_n[n] = i;
            n++;
        }
    }
    for (i=0; i<n-1; i++) {
        Dx_n[i] = x_n[i+1] - x_n[i];
    }
    //------------------------------------------------------------------------
    // store x_n
    sprintf(fname, "xn_tn");
    fp = fopen(fname, "w+");
    for (i=0; i<n; i++) {
        fprintf(fp, "%d %f %f\n", i, dx*(double)(x_n[i]), appearence_time[n-1-i]);
    }
    fclose(fp);
    //------------------------------------------------------------------------
    // store Dx_n
    sprintf(fname, "Dx_n");
    fp = fopen(fname, "w+");
    for (i=0; i<n-1; i++) {
        fprintf(fp, "%d %f\n", i, dx*(double)(Dx_n[n-2-i]));
    }
    fclose(fp);
    //------------------------------------------------------------------------
    // derivative of x_n with respect to time
    sprintf(fname, "Dx_nDt");
    fp = fopen(fname, "w+");
    for (i=1; i<n-1; i++) {
        fprintf(fp, "%f %f\n", appearence_time[i],
                - dx*(double)(Dx_n[n-1-i])/(appearence_time[i+1] - appearence_time[i]));
    }
    fclose(fp);
}



//============================================================================
// Heaviside step function
//============================================================================
double theta(double x)
{
    if (x < 0.0)
        return 0.0;
    else
        return 1.0;
}
```