Universität Potsdam
Mathematisch-Naturwisssenschaftliche Fakultät
Institut für Physik und Astronomie

Computational Physics

Falling liquid films
Kuramoto-Sivashinsky equation

Author: Christian Gößl
Matrikel-No.:762627

corrector: Prof. Dr. Arkady Pikovsky

July 24, 2019

Contents

1 Introduction

On a heavy raining day you see, that water is flowing down the window. There you can find also some small waves, they are going down. In order to describe this phenomenon, physicists used a plait and investigated the water flow on that. Usually, in our everyday life the window is perpendicular to the ground. They changed the angle to ground from 90 to 0 degrees. In that arrangement the flow depends on the angle and they find formulas to describe the behavior of the flow. The falling liquid films produces wavetrains see figure 1. They found, that the flowing can be turbulent see figure 2.

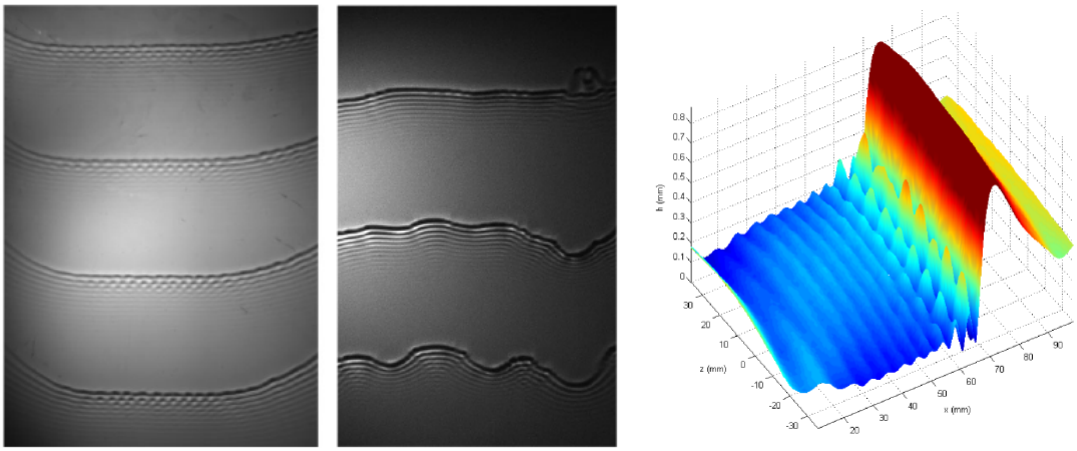

Figure 1: The left picture shows falling liquid films of physical experiment. The wave trains are going down and creating some nonregular structures.[12] The right picture shows a simulation of the falling liquid film wave trains in 3D. [15]

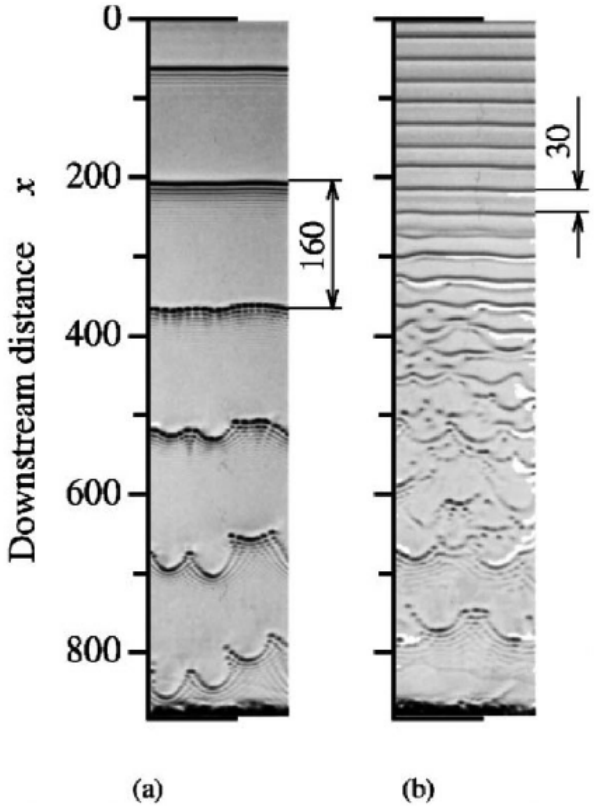

Figure 2: The falling liquid films producing waves. When the distance is growing the waves are transformed to a more unregular wave patterns, then we have a turbulent flow. [12]

In that work we want to simulate those turbulent flows. For that purpose we used a 1D formulation of

that problem, the Kuramoto-Sivashinsky equation. We will investigate, which parameters are important for producing turbulence and how the turbulent flow will appear.

2 Theory

In that work, we want to simulate falling liquid films on a plaid. There we can find turbulent solution. Usually, it is described in two or three dimensions, we took the one dimensional approach to show the turbulent behavior of the fluid at a specific choice of values of physical parameters. We describe the physical phenomenon with the typical variables: t, x and u, displacement, time and velocity. The width is L. We used the Kuramoto-Sivashinsky equation, which was derived by Yoshiki Kuramoto and Takeo Tsuzuki [10] and by Gregory Sivashinsky [16], here in a general form:

$$a\frac{\partial u}{\partial t} + b\frac{1}{2}\frac{\partial (u^2)}{\partial x} + c\frac{\partial^2 u}{\partial x^2} + d\frac{\partial^4 u}{\partial x^4} = 0 \tag{1}$$

We rescale the equation in the variables t, x and u. We take for u and x the transformation as follows

$$x = \frac{y}{b} \quad \partial_x = b\partial_x \quad u = \frac{v}{b}$$

and insert it into equation (1)

$$\frac{a}{b}\partial_t v + v\partial_y v + cb\partial_{yy} v + db^3 \partial_{yyyy} v = 0$$

At last, we transform the time t and set the parameter

$$\partial_t \frac{a}{b} = \partial_\tau \quad t = \frac{b}{a}\tau \quad c = \frac{1}{b} \quad d = \frac{1}{b^3}$$

The final equation becomes

$$\partial_\tau v + v\partial_y v + \partial_{yy} v + \partial_{yyyy} v = 0$$

We use the old notation with the modification $\partial_\tau v = v_{,\tau}$

$$u_{,t} + uu_{,x} + u_{,xx} + u_{,xxxx} = 0 \tag{2}$$

Furthermore, we can find a solution with the Fouriertransformation

$$u(t,x) = \sum_{n=-\infty}^{\infty} C_n(t) \exp\left(in\frac{2\pi}{L}x\right) \tag{3}$$

$$C_n(t) = \frac{1}{L}\int_0^L \mathrm{d}x\, u(t,x) \exp\left(-in\frac{2\pi}{L}x\right) \tag{4}$$

Additionally, we can use for the computation of the coefficients the property

$$C_{-n} = C_n^*$$

In the Fourierspace we have the functions ϕ_n and wave vector $K = 2\pi/L$

$$\phi_n(x) = \exp\left(inKx\right) \ .$$

They are orthogonal to each other

$$\frac{1}{L}\int_0^L \mathrm{d}x\, \phi_n \phi_l^* = \frac{1}{L}\int_0^L \mathrm{d}x\, \phi_n \phi_{-l} = \delta_{nl} \ . \tag{5}$$

We take the time derivative of the C_n and insert equation (2)

$$\begin{aligned} C_{n,t} &= \frac{1}{L}\int_0^L \mathrm{d}x\, u_{,t} \exp\left(-inKx\right) \\ &= -\frac{1}{L}\int_0^L \mathrm{d}x\ \left(uu_{,x} + u_{,xx} + u_{,xxxx}\right) \exp\left(-inKx\right) \end{aligned}$$

We use partial integration for every term in the integral

$$C_{n,t} = -\frac{1}{L}\int_0^L \mathrm{d}x \left(\frac{inK}{2}u^2 - n^2K^2u + n^4K^4u\right)\exp\left(-inKx\right)$$

Then we insert the equation (3) and use orthogonality property of the ϕ_n (see equation (5)), hence in the sum of all C_l we get only the nth C_n

$$\begin{aligned} C_{n,t} &= -\frac{inK}{2L}\int_0^L \mathrm{d}x \left(\sum_{l=-\infty}^{\infty} C_l \exp\left(ilKx\right)\right)^2 \exp\left(-inKx\right) + \left(n^2K^2 - n^4K^4\right)C_n \\ &= \left(n^2K^2 - n^4K^4\right)C_n + F_n(t, C_n) \end{aligned} \tag{6}$$

for $n, l \in (-\infty, \infty)$. We need a finite number of equations for the simulation, therefore we apply the Galerkin approximation on the equation (6). It means, that we restrict the number of modes to $2N+1$

$$C_{n,t} = a_n C_n - \frac{inK}{2L}\int_0^L \mathrm{d}x \left(\sum_{l=-N}^{N} C_l \exp\left(ilKx\right)\right)^2 \exp\left(-inKx\right)$$

with $a_n = \left(n^2K^2 - n^4K^4\right)$ and $|n|, |l| \leq N$. In the computation of the nonlinear part F_n we are using the orthogonality property of the ϕ_n

$$\begin{aligned} F_n(t, C_n) &= -\frac{inK}{2L}\int_0^L \mathrm{d}x \left(\sum_{l=-N}^{N} C_l \exp\left(ilKx\right)\right)\left(\sum_{q=-N}^{N} C_q \exp\left(iqKx\right)\right)\exp\left(-inKx\right) \\ &= -\frac{inK}{2L}\sum_{l=-N}^{N} C_l \sum_{q=-N}^{N} C_q \delta_{(l+q)n} \end{aligned}$$

The coefficients are determined with the $\delta_{(l+q)n}$ and hence we can set $q = n - l$

$$F_n = -\frac{in\pi}{L}\sum_{l=n-N}^{N} C_l C_{n-l} \tag{7}$$

We insert all together

$$C_{n,t} = a_n C_n - \frac{in\pi}{L}\sum_{l=n-N}^{N} C_l C_{n-l} \tag{8}$$

In linear approximation we have only the first term

$$C_{n,t}(t) = \left(n^2K^2 - n^4K^4\right)C_n(t)$$

We can find here some interesting properties of the coefficients. Our solution $u(t,x)$ consists of sum of Fouriercoefficients C_n. The total number of them are $M = 2N+1$. The linear approximation is a simple ODE in first order. Thus, the already known solution is $C_n = \exp(a_n t)$. In case of growing, we have to investigate the term $a_n > 0$

$$\begin{aligned} 0 &< \left(n^2\left(\frac{2\pi}{L}\right)^2 - n^4\left(\frac{2\pi}{L}\right)^4\right) \\ \frac{L}{2\pi} &> n_{\mathrm{grow}} \end{aligned} \tag{9}$$

Hence all modes below that value are growing and unstable. They are depended on the value of L. We can get the total number of modes from the equation 9 and n take values from $-N$ to N, hence we have

$$N_{\mathrm{grow}} = L/\pi \tag{10}$$

growing modes. Furthermore, in case of no growing, we get

$$\frac{L}{2\pi} = n$$

and the coefficient C_0 in (8) is not changing in time, because all terms are multiplied by zero

$$C_{0,t} = 0 \quad C_0 = \mathrm{const.}$$

So all other modes will be stable and the values will be falling.

2.1 Numerical Method

In order to find a solution of the equation (8), we used the numerical method: exponential time differencing method with Runge–Kutta time stepping (ETD2RK) of Cox and Matthews [9]

$$\begin{aligned} \tilde{C}_n^m &= C_n^m \exp(a_n h) + F_n(C_n^m, t^m)\frac{(\exp(a_n h) - 1)}{a_n} \\ C_n^{m+1} &= \tilde{C}_n^m + (F_n(\tilde{C}_n^m, t^m + h) - F_n(C_n^m, t^m))\frac{(\exp(a_n h) - 1 - h a_n)}{h a_n^2} \end{aligned} \tag{11}$$

with $h = \Delta t$.

2.1.1 Computation of the advection of particles

We want to follow the path of particles on top of film. We used the simple velocity law

$$\frac{\mathrm{d}X}{\mathrm{d}t} = u(X, t) \tag{12}$$

We used the simple Euler method to compute the path of particles. It is sufficient, because we only have calculated two points in one time step.

$$X^{m+1} = X^m + \Delta t u^m(X^m, t^m) \tag{13}$$

3 Simulation of turbulent falling liquid films

As we investigated in the theory part(see last section), that the solution depends on the choice of the parameter L and N. For the calculation of modes we had to shift the value interval of n from $n \in [-N,N]$ to $n \in [1,2N+1]$, because the program can only compute with positive indices of arrays. We fixed the parameter N to the triple of number of unstable modes for the next sections 3.1, 3.2 . In detail, we take the maximal value of growing modes from equation (9) and multiply it with three

$$3n = \frac{3L}{2\pi} = N_{\text{triple}}.$$

The total number of coefficients is

$$M = 2N_{\text{triple}} + 1 \ .$$

In section 3.3, we changed the number of coefficients to investigate their dependency on the solution. The physical quantity is changing over time t and in x direction, therefore we used a 3D plot in u, t and x. The point of view in the 3D plots is perpendicular to the t,x plain. We equally distributed the particles in the simulation along the x direction. With evolution in time we get the path of particles in the plot of time and x direction.

3.1 Stable waves

We see stable waves above values of $L \geq 7$. We used the following parameters

$$\Delta t = 10^{-3},\ t_{\text{end}} = 200,\ \Delta x = 0.1,\ C_n(0) = 0.001,\ L = 6,7,15\ ,N = 3,3,9$$

The stable waves are occurring only up to the limit of $L \leq 20$. The result is shown in the figure 3. At the beginning, we have not much changing in the amplitude and not a typical waveform. After some time, we can see an uprising of large waves, that flows down the plait. The waves are constant in their shape, but moves slowly to the x direction. The particles are collecting at the wavefronts, in the midpoint of the whole wave. Additionally, they are following that part, where the wavefront goes in the positive x direction and also the wavevector point in that direction. The values of the velocity u are increasing with the increment of the parameter L.

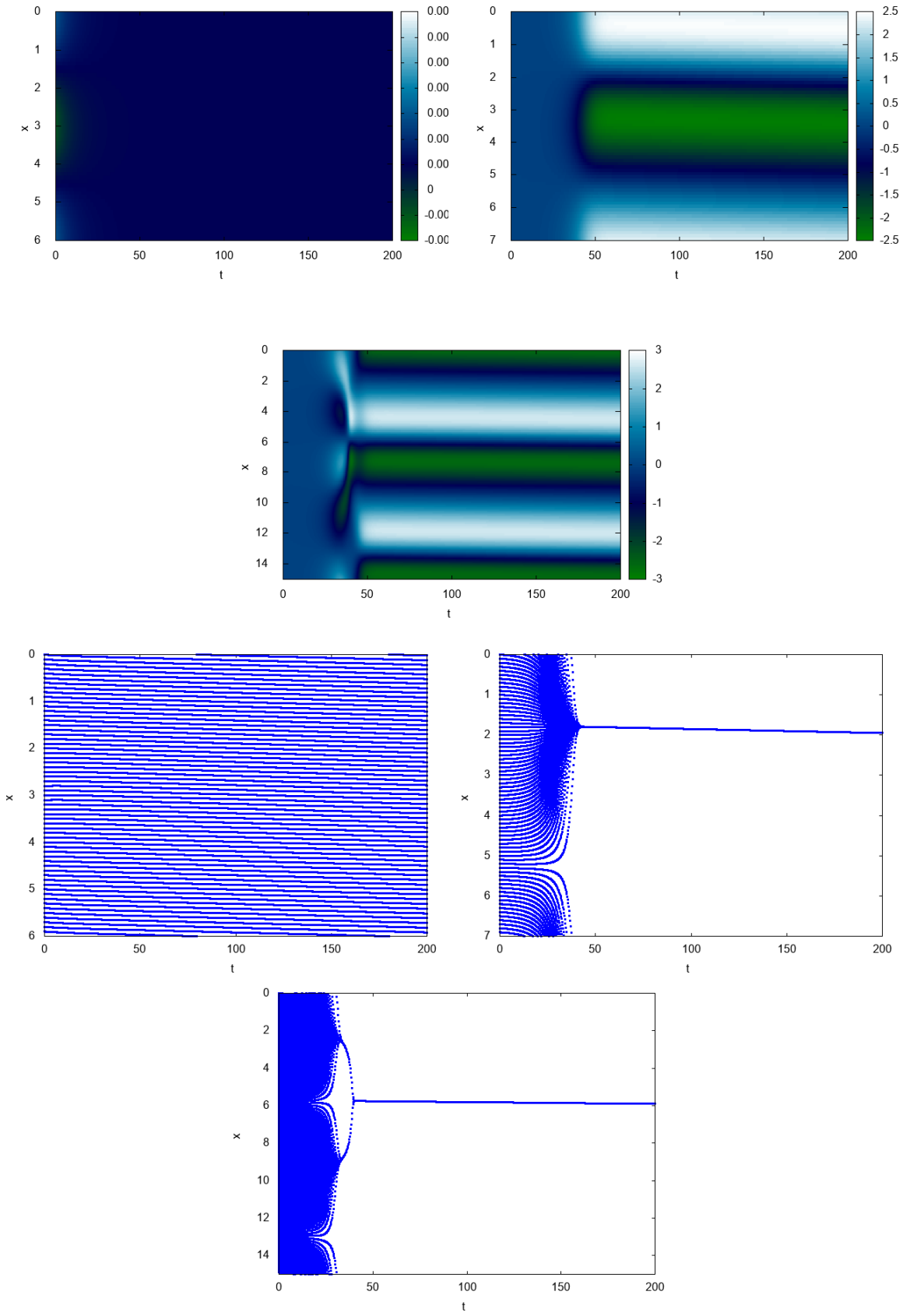

Figure 3: The first three plots are 3D plots and the other ones are the paths of particles. We used here $L = 6, 7, 15$ and $N = 3, 3, 9$, from top to down, the values of L are growing. We get stable wavetrains. The blue dots(particles) following the wave train in the positive x direction.

3.2 Turbulent waves

We used the same parameter range of the last section 3.1, but

$$L = 20, 27, 40, 64 \quad N = 9, 12, 19, 30$$

The result is shown in the figure 4. At the beginning of $L = 20$, we see some deformations of the waves, but they are still stable for a period. The width of the waves are getting smaller with increasing of L. When we increase the parameter to $L = 27$, then we get more deformations and the flowing becomes turbulent. As we saw before, the particles are following the wave trains at the midpoint with a moving in the x direction. The range of values of velocity are $u \in [-4,4]$ and is constant in the increment of parameter L.

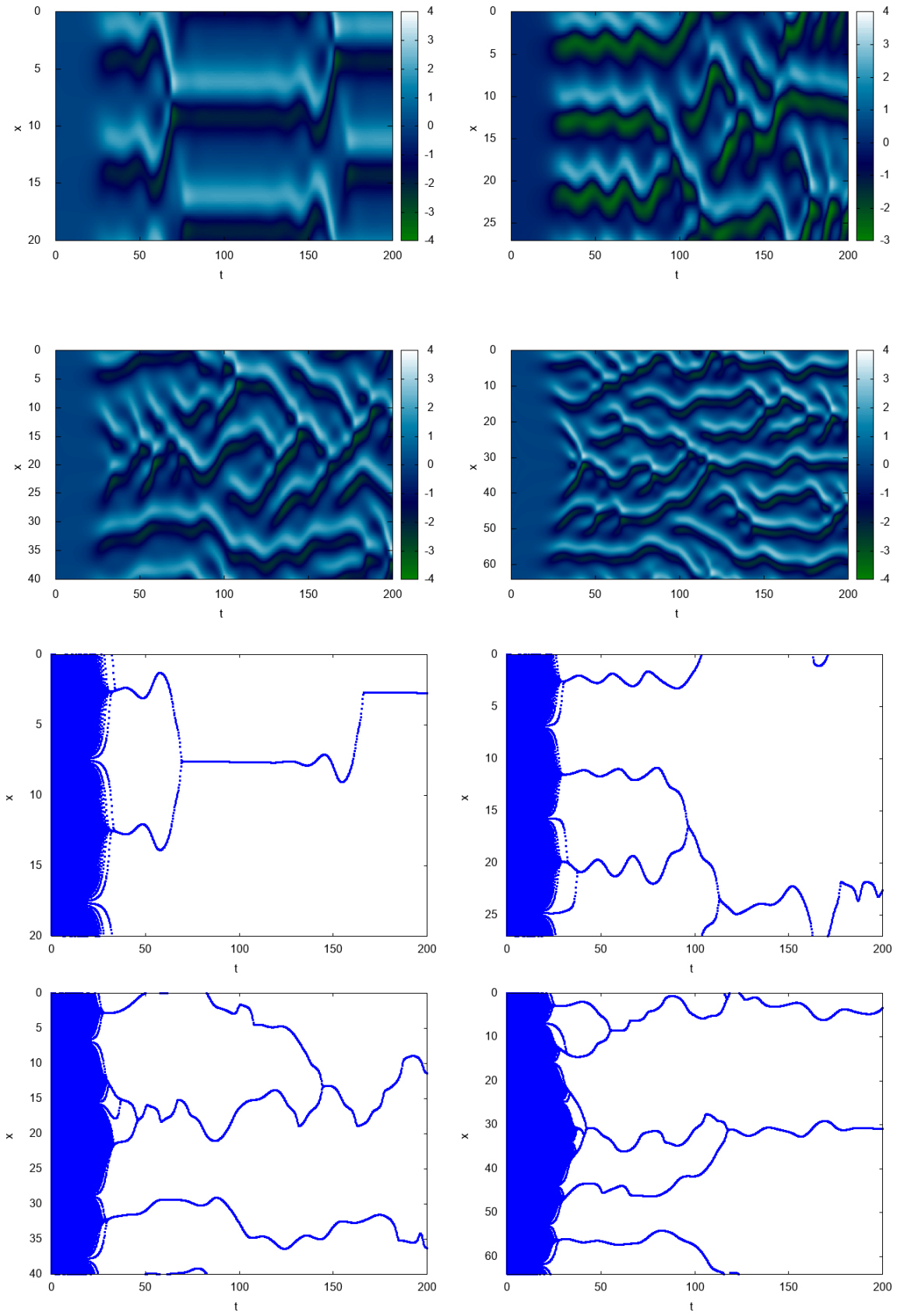

Figure 4: As before, the first three plots are 3D plots and the other ones are the paths of particles. We used here $L = 20, 27, 40, 64$ and $N = 9, 12, 19, 30$, from top to down, the values of L are growing. We increased the parameter L and get here a turbulent flow.

3.3 Dependency of number of coefficients

We want to investigate the dependency of number of coefficients. For that task we change the parameter N and the parameter L is constant. We used the following list of parameter

$$\Delta t = 10^{-3},\ t_{\text{end}} = 350,\ \Delta x = 0.1,\ C_n(0) = 0.001,\ L = 27,\ N = 6, 7, 10, 19$$

In case of $N = 6$, we have high amplitudes of u and a fast changing of the turbulent flow. In case of $N > 6$, we have a turbulent flow, but at specific times, there are stable wave fronts with small oscillations. Below $N < 6$, there exists no stable simulations, after a short time, the amplitude of u goes to infinity. There are more unstable then stable ones, because of the formula (9) for the number of growing modes

$$n_{\text{grow}} = \frac{L}{2\pi} \approx 4.2972$$

Hence, we need more stable modes above this value.

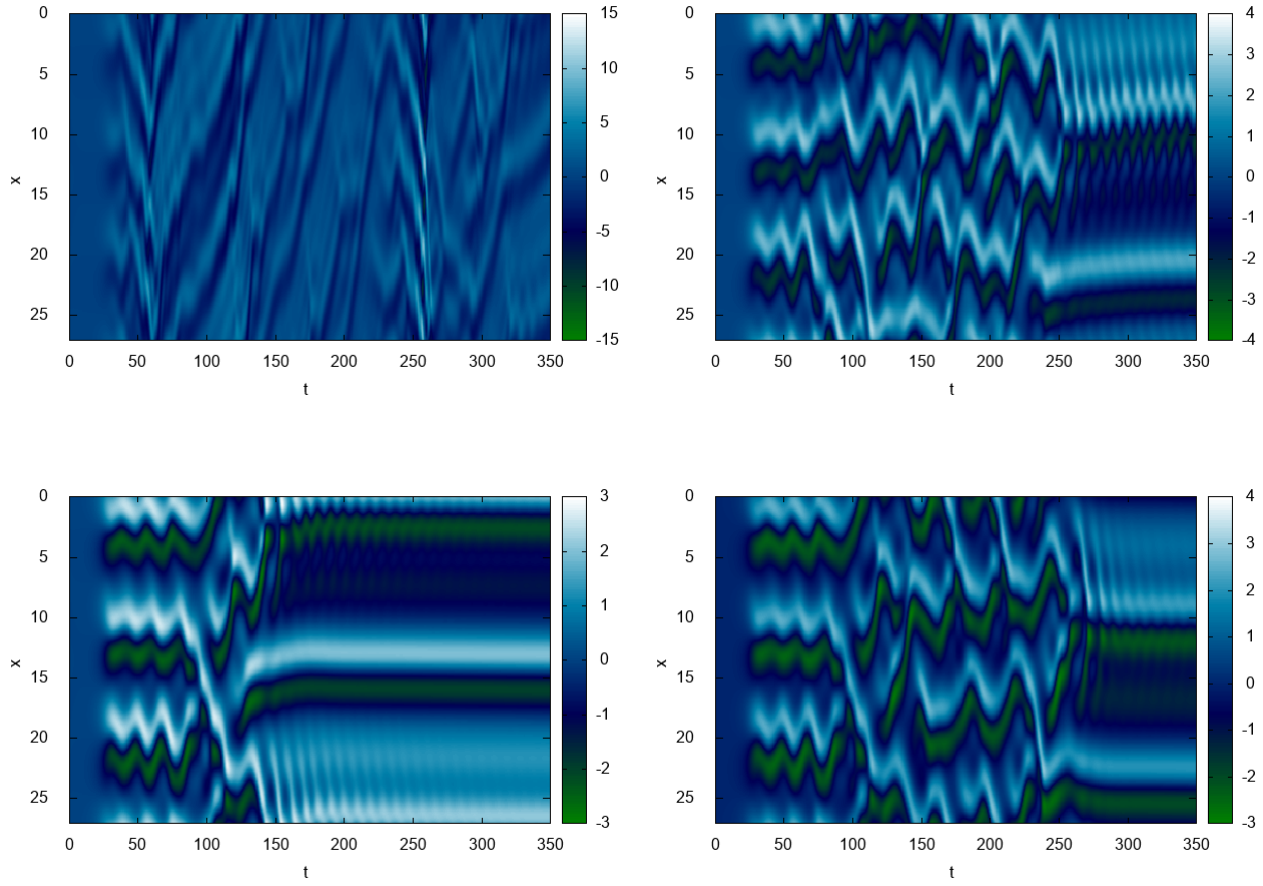

Figure 5: We used here $L = 27$ and $N = 6, 7, 10, 19$, from top to down, the values of N are growing. We increased the parameter N and get here a turbulent flow, that sometimes stabilizes at a specific time with small oscillations.

4 Conclusion

We found, that the flow is depended on the parameters L and N. L is the width of the simulation and N the half number of modes. In case of $L < 20$, we have stable waves with no turbulence. In case of $L \geq 20$, the flowing becomes unstable and it produces turbulence. The parameter N changes the number of modes. In the theory part we found, that we get stable and unstable modes. In case of $N \leq n_{\text{grow}}$, the simulation becomes unstable, hence we have not enough stable modes to compensate the unstable modes. In case of $N > n_{\text{grow}}$, the simulation becomes stable. When we distribute on that flow some particles, we can tracking them to their final destination. They follow the wave fronts in positive directions and after some time all particles are collected at a wave.

5 Program

The code is written in the programming language Julia (`https://julialang.org/`). You can see the code below this chapter, where you can find some annotations for clarification. The program has some parameter.

```
julia falling-liquid-films.jl p1 p2 p3 p4 p5 p6 p7 p8

p1 - L, lenght of the cell
p2 - N, half number of modes
p3 - delta_t, time step
p4 - limit_t, end time of the simulation
p5 - delta_x, x direction steps
p6 - schalter, choice between: live plot value=1 and save computed data value=0
p7 - C_n(0), start value for all avaible modes
p8 - out_time, time step for the calculation of velocity field u
```

An example execution command is

```
julia falling-liquid-films.jl 20 18 1e-3 200 1e-1 0 0.01 0.5
```

In the live plot mode you have to use the command

```
julia falling-liquid-films.jl 20 18 1e-3 200 1e-1 1 0.01 0.5 | gnuplot
```

13a ⟨*falling-liquid-films.jl* 13a⟩≡

```
# Falling liquid films: Kuramoto-Sivashinsky equation program

⟨Output 13b⟩
⟨Init 14⟩
⟨XPoints 15a⟩
⟨FunctionU 15b⟩
⟨Comp Coeff 15c⟩
⟨Comp Advect 16a⟩
⟨Comp SingAdvect 16c⟩
⟨Comp Velo 16b⟩
⟨NumCalc 17⟩

# Execution of algorithm to simulate the falling liquid films.

Init()

CompCoeff()

NumCalc()
```

The output function for gnuplot program

13b ⟨*Output* 13b⟩≡ (13a)

```
function output(channel, data1, data2, data3 )
   if channel == 1
      write(single_particle_file,"$data1 $data2 $data3 \n")
   elseif channel == 2
      write(xtrajectory_file,"$data1 $data2 $data3 \n")
   elseif channel == 3
      write(data_file,"$data1 $data2 $data3 \n")
   elseif (channel == 4 && schalter == 1 )
      println("$data1 $data2 $data3")
   end
end
```

Initialization of the starting parameter. The boundary conditions for $C_n(0)$ are arbitrary.

14 ⟨*Init* 14⟩≡ (13a)

```
function Init()

   global L = parse(Int, ARGS[1] )# length of the x direction
   global N = parse(Int, ARGS[2] )# number of modes
   global M = 2*N+1
   global delta_t = parse(Float64, ARGS[3] )# time step
   global limit_t = parse(Float64, ARGS[4] )# time of simulation
   global delta_x = parse(Float64, ARGS[5] )# x step
   global x = Float64[0:delta_x:L;] # grid for x plotting
   global xsize = length(x )
   global C = complex(zeros(Float64, M ) ) # complex coefficients C_n
   fill!( C, parse(Float64, ARGS[7] )) # start points of C_n(0) at t = 0
   global F = complex(zeros(Float64, M ) )# function for the nonlinear terms
   global X = zeros(Float64, xsize ) # particles in the flow
   global Y = zeros(Float64, xsize ) # particles in the flow
   global out_time = parse(Float64, ARGS[8] )
   global schalter = parse(Int, ARGS[6] ) # switch for live plotting of u,x
   global lineend = 1

   ### data files for the output and input
   if schalter == 0
      global data_file = open("data-$L-$N-$delta_t-$limit_t-$delta_x-$(parse(Float64, ARGS[7] ) )-$out_time.t
      global xtrajectory_file = open("xtrajectory-$L-$N-$delta_t-$limit_t-$delta_x-$(parse(Float64, ARGS[7] )
      global single_particle_file = open("single_particle-$L-$N-$delta_t-$limit_t-$delta_x-$(parse(Float64, A
   else
      global data_file = open("data.txt","w") # file for x,t,u plotting
      global xtrajectory_file = open("xtrajectory.txt","w") # file for x trajectory plotting of x,t
      global single_particle_file = open("single_particle.txt","w")
   end
   global xstart_file = open("xstart.txt")# file for positions of particles

   output(1, "#$(parse(Int,ARGS[1]))-$(parse(Int,ARGS[2]))-\\
#$(parse(Float64,ARGS[3]))-$(parse(Float64,ARGS[4]))-\\
#$(parse(Float64,ARGS[5]))-$(parse(Int,ARGS[6]))-\\
#$(parse(Float64,ARGS[7])))", " ", " ")
   output(2, "#$(parse(Int,ARGS[1]))-$(parse(Int,ARGS[2]))-\\
#$(parse(Float64,ARGS[3]))-$(parse(Float64,ARGS[4]))-\\
#$(parse(Float64,ARGS[5]))-$(parse(Int,ARGS[6]))-\\
#$(parse(Float64,ARGS[7])))", " ", " ")
   output(3, "#$(parse(Int,ARGS[1]))-$(parse(Int,ARGS[2]))-\\
#$(parse(Float64,ARGS[3]))-$(parse(Float64,ARGS[4]))-\\
#$(parse(Float64,ARGS[5]))-$(parse(Int,ARGS[6]))-\\
#$(parse(Float64,ARGS[7])))", " ", " ")
   output(4, "set yrange [-6:6]", " ", " " )

   # setup a number of particles
   XPoints()
end
```

The setup of a number of particles, that follow the stream.

15a ⟨*XPoints* 15a⟩≡ (13a)

```
function XPoints()
   # starting points of the grid
   for j in 0:xsize-1
      Y[j+1]=x[j+1]
   end
   # starting points for specifig number of particles, definied in the xstart file
   global lineend = 1
   while !eof(xstart_file)
     X[lineend] = parse(Float64, readline(xstart_file ) )
     #println("$lineend $X[lineend]")
     lineend += 1
   end
end
```

Calculation of the velocity $u(t,x)$

$$u(t,x) = \sum_{n=-\infty}^{\infty} C_n(t) \exp\left(in\frac{2\pi}{L}x\right)$$

We used here a Galerkin approximation in the number of modes of $C_n(t)$

15b ⟨*FunctionU* 15b⟩≡ (13a)

```
function FunctionU(x)
   u = complex(Float64(0 ) )
   for ne in 1:M
         if (ne > N+1)
            u = u + conj(C[2*N+2-ne])*exp(im*(ne-1-N )*2*pi*x/L )
         elseif (ne <= N+1)
            u = u + (C[ne])*exp(im*(ne-1-N )*2*pi*x/L )
         end
      end
   return u
end
```

We want to compute the $C_n(t)$ Coefficients for the Fouriertransformation. We used the complex coefficients. Usually, the index of coefficients goes from negative to positive values of n, then we shift the index from $n \in [-N,N]$ to $n \in [1,2N+1]$.

15c ⟨*CompCoeff* 15c⟩≡ (13a)

```
function CompCoeff()
   for ne in 1:N
      C[N+1-ne]=conj(C[N+1+ne] )
   end
   for ne in 2+N:M
      F[ne] = 0
      for l in ne-N:M
         F[ne] = F[ne] + C[l] * C[ne-l+N+1]
      end
      F[ne]=-F[ne]*im*(ne-N-1)*pi/L
   end
end
```

Calculation of the path of the particles. They follow the velocity law

$$\frac{\mathrm{d}X}{\mathrm{d}t} = u(X,t)$$

We used the simple Euler method to compute the path of the particles. This method is sufficient, because we only have calculated two points in one time step with the chosen method (ETD2RK).

$$X_{m+1} = X_m + \Delta t u(X_m, t_m)$$

16a ⟨*CompAdvect* 16a⟩≡ (13a)

```
function CompAdvect(t,j)
   Y[j+1]=Y[j+1] + out_time*real(FunctionU(Y[j+1] ) )
   if Y[j+1]>L
      Y[j+1]=Y[j+1]-L
   elseif Y[j+1]<0
      Y[j+1]=Y[j+1]+L
   end
   output(2, t, Y[j+1], " " )
end
```

Computation of the velocity $u(t,x)$

16b ⟨*CompVelo* 16b⟩≡ (13a)

```
function CompVelo(t)
   output(4, "plot '-' using 2:3 w l", " ", " " )
   for j in 0:xsize-1 #x direction steps
      w::Float64 = real(FunctionU(x[j+1] ) )
      output(3, t, j*delta_x, w )
      output(4, t, j*delta_x, w )

      CompAdvect(t,j)
   end
   output(2, " ", " ", " " )
   output(3, " ", " ", " " )
   output(4, "eol: 0", " ", " " )
end
```

The function computes the path of particles, that was given by a list of starting positions in the file `xstart.txt`.

16c ⟨*CompSingAdvect* 16c⟩≡ (13a)

```
function CompSingAdvect(t)
   for j in 1:lineend
      X[j]=X[j] + out_time*real(FunctionU(X[j] ) )
      if X[j]>L
         X[j]=X[j]-L
      elseif X[j]<0
         X[j]=X[j]+L
      end
      output(1, t, X[j], " " )
   end
   output(1, " ", " ", " " )
end
```

Numerical solution of the system of ordinary differential equations for $C_n(t)$

17 ⟨*NumCalc* 17⟩≡ (13a)

```
function NumCalc()
   Fe = complex(zeros(Float64, M ) )
   for t in 0:delta_t:limit_t
      Ce = complex(zeros(Float64, M ) )
      for ne in N+2:M
         a = Float64(((ne-N-1)*2*pi/L )^2-((ne-N-1)*2*pi/L )^4 )
         Fe[ne]=F[ne]
         Ce[ne] = C[ne]*exp(a*delta_t) + F[ne]*(exp(a*delta_t )-1 )/a
      end

      CompCoeff()

      for ne in N+2:M
         a = Float64(((ne-N-1 )*2*pi/L )^2-((ne-N-1 )*2*pi/L )^4 )
         C[ne] = Ce[ne] + (F[ne]-Fe[ne] )*(exp(a*delta_t ) -1 - a*delta_t )/(a^2*delta_t )
      end
      if (trunc(mod(t*1e4, out_time*1e4)) == 0 )
         CompVelo(t)
         CompSingAdvect(t)
      end
   end
end
```

References

[1] basis_projections_galerkin.pdf, . URL users.jyu.fi/~oljumali/teaching/TIES594/04/basis_projections_Galerkin.pdf.

[2] galerkin2.pdf, . URL fischerp.cs.illinois.edu/tam470/refs/galerkin2.pdf.

[3] Galerkin_method.pdf, . URL www.sd.ruhr-uni-bochum.de/downloads/Galerkin_method.pdf.

[4] Kuramoto-sivashinsky equation - encyclopedia of mathematics, . URL https://www.encyclopediaofmath.org/index.php/Kuramoto-Sivashinsky_equation.

[5] task sheet ks.pdf, .

[6] Hsueh-Chia Chang. Traveling waves on fluid interfaces: Normal form analysis of the kuramoto–sivashinsky equation. 29(10):3142. ISSN 00319171. doi: 10.1063/1.865965. URL https://aip.scitation.org/doi/10.1063/1.865965.

[7] S.M. Cox and P.C. Matthews. Exponential time differencing for stiff systems. 176(2):430–455. ISSN 00219991. doi: 10.1006/jcph.2002.6995. URL https://linkinghub.elsevier.com/retrieve/pii/S0021999102969950.

[8] Ricardo G Duran. Galerkin approximations and finite element methods. page 46. URL mate.dm.uba.ar/~rduran/class_notes/fem.pdf.

[9] Dr Noemi Friedman. Numerical methods for PDEs FEM - abstract formulation, the galerkin method. page 20. URL https://www.wire.tu-bs.de/lehre/ss14/pde2/PDE2_Tutorial_2_Galerkin_method.pdf.

[10] Yoshiki Kuramoto and Toshio Tsuzuki. Persistent propagation of concentration waves in dissipative media far from thermal equilibrium. 55(2):356–369. ISSN 0033-068X. doi: 10.1143/PTP.55.356. URL https://academic.oup.com/ptp/article/55/2/356/1883646.

[11] Daniel Michelson. Steady solutions of the kuramoto-sivashinsky equation. 19(1):89–111. ISSN 0167-2789. doi: 10.1016/0167-2789(86)90055-2. URL http://www.sciencedirect.com/science/article/pii/0167278986900552.

[12] Akio Miyara. Numerical simulation of wavy liquid film flowing down on a vertical wall and an inclined wall. 39(9):1015–1027. ISSN 1290-0729. doi: 10.1016/S1290-0729(00)01192-3. URL http://www.sciencedirect.com/science/article/pii/S1290072900011923.

[13] T. Nosoko, P. N. Yoshimura, T. Nagata, and K. Oyakawa. Characteristics of two-dimensional waves on a falling liquid film. 51(5):725–732. ISSN 0009-2509. doi: 10.1016/0009-2509(95)00292-8. URL http://www.sciencedirect.com/science/article/pii/0009250995002928.

[14] A. Pumir, P. Manneville, and Y. Pomeau. On solitary waves running down an inclined plane. 135(-1):27. ISSN 0022-1120, 1469-7645. doi: 10.1017/S0022112083002943. URL http://www.journals.cambridge.org/abstract_S0022112083002943.

[15] Christian Ruyer-Quil, Nicolas Kofman, Didier Chasseur, and Sophie Mergui. Dynamics of falling liquid films. 37(4):30. ISSN 1292-8941, 1292-895X. doi: 10.1140/epje/i2014-14030-5. URL http://link.springer.com/10.1140/epje/i2014-14030-5.

[16] G. I. Sivashinsky. Nonlinear analysis of hydrodynamic instability in laminar flames—i. derivation of basic equations. In Pierre Pelcé, editor, *Dynamics of Curved Fronts*, pages 459–488. Academic Press. ISBN 978-0-12-550355-6. doi: 10.1016/B978-0-08-092523-3.50048-4. URL http://www.sciencedirect.com/science/article/pii/B9780080925233500484.