Utrecht University

BACHELOR THESIS

Scaling of attractor dimension in the discretized Kuramoto-Sivashinsky equation

Author:
Sebastian ZASADNY

Supervisor:
Prof. Jason FRANK

Faculty of Science
Department of Mathematics

January 27, 2023

Contents

Chapter 1

Introduction

The Kuramoto-Sivashinsky equation is a well-researched fourth-order nonlinear PDE that was first introduced by Yoshiki Kuramoto[6] in 1976 to model pattern formation in thermohydraulics and a year later by Gregory Sivashinsky[8] to model diffusive–thermal instabilities. The equation is well known for its chaotic behavior and has many interesting properties.

The property that we are going to look into inside this thesis is the dimension of the attractor and how it scales with size of the truncation when the equation is numerically approximated. It is already known from [3] that the value of the dimension of an attractor corresponds with the domain size L, meaning that a bigger domain also increases the calculated dimension. In this thesis we will primarily look at how the dimension scales with the grid size M, meaning the quantity of elements into which the Kuramoto-Sivashinsky equation is divided when discretizing the function.

In the rest of this chapter we will discuss our definition of the Kuramoto-Sivashinsky equation and the definition that we will use for the attractor dimension. Then in chapter 2 we will define our method for numerically solving the Kuramoto-Sivashinsky equation and after that, in chapter 3 we will take a look at how we can make use of the solutions of the equation to numerically approximate the Lyapunov exponents, which will be needed for the calculation for the attractor dimension as can be seen in section 1.2. Next, in chapter 4, we will go over the results of numerical experiments that we got from applying the defined numerical methods into Python code which will follow into the conclusion in chapter 5. Finally, in Appendix A the reader can see the actual application of the described numerical methods in code.

1.1 The Kuramoto-Sivashinsky equation

For this thesis we define the Kuramoto-Sivashinsky equation as follows:

Definition 1 *Given function $u(x,t)$ and a domain L, the Kuramoto-Sivashinsky equation is defined as:*

$$u_t + u_{xx} + u_{xxxx} + uu_x = 0, \tag{1.1}$$

With the initial conditions:

$$(x,t) \in \mathbf{R} \times \mathbf{R}^+, \quad u(x,0) = u_0,$$

and the boundary conditions:

$$u(x+L,t) = u(x,t), \quad 0 \leq x \leq L, \quad t \geq 0.$$

An interesting property of this equation is the second order term u_{xx} is destabilizing while the fourth order term u_{xxxx} is stabilising giving this equation bounded solutions.

1.2 Attractor dimension

Some PDEs have clearly defined attractors. The Kuramoto-Sivashinsky equation however is known for its chaotic attractors, which can be pretty hard to define. It is proven however that the Kuramoto-Sivashinsky equation is one of the few PDEs to have a finite dimensional invariant manifold [4], where the solution of the equation clearly converges towards. Therefore, to be able to quantitatively express an attractor, and to be able to measure possible changes of said attractor, Kaplan and Yorke[5] came up with a definition for an attractor dimension which we express in Lyapunov exponents. This attractor dimension is also called the Lyapunov dimension or the Kaplan-Yorke dimension and is defined as follows:

Definition 2 *The Lyapunov dimension D_L is defined as:*

$$D_L = k + \frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{|\lambda_{k+1}|}, \tag{1.2}$$

where the Lyapunov exponents λ_i are defined so that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$ for $i \in \{1, \ldots, m\}$ and k is the maximum value such that $\lambda_1 + \lambda_2 + \cdots + \lambda_k > 0$.

To calculate the attractor dimension for the Kuramoto-Sivashinsky equation, we thus need to first calculate the Lyapunov exponents of the equation. Given the chaotic nature of this equation, it's the best course of action to approximate the Lyapunov exponents numerically. To do this, we will first have to numerically approximate a solution for the Kuramoto-Sivashinsky equation as defined in chapter 2 which we will use to approximate the Lyapunov exponents in chapter 3.

Chapter 2

Numerically solving the Kuramoto-Sivashinsky equation

In this section we will work towards a numerical method for an approximated solution for the Kuramoto-Sivashinsky equation. We shall do this by first discretizing the equation while taking the equation into Fourier space in section 2.1. Then, in section 2.2 we will apply the Implicit-Explicit Runge-Kutta method on the discretized equation to numerically approximate a solution. Finally, in section 2.3, we will dive a bit more into the properties of the Implicit-Explicit Runge-Kutta method.

Readers who wish to directly look at the approximation of the Lyapunov components and the calculation of the attractor dimension may skip to the next chapter 3. However, these readers should be aware that we will also apply the Implicit-Explicit Runge Kutta method as described in section 2.2 in the process of approximation of components of the Lyapunov exponents in section 3.3.

2.1 Discrete Fourier transformation

To solve the Kuramoto-Sivashinsky equation, we will calculate each derivative inside of Fourier space. Moreover, in section 2.2 we will be working with a spectral method where we have to make some algebraic manipulations inside Fourier space. Thus as a preliminary, we will first apply a discrete Fourier transformation on the terms of the Kuramoto-Sivashinsky equation.

We start by rewriting equation (1.1) as:

$$u_t = -u_{xx} - u_{xxxx} - uu_x = Au + f(u). \tag{2.1}$$

Here, we split the equation into the linear operator

$$Au = -u_{xx} - u_{xxxx}, \tag{2.2}$$

and the nonlinear operator;

$$f(u) = -uu_x. \tag{2.3}$$

As we want to numerically approximate the solution of this equation, we numerically discretize each component of equation (2.1). For this we let $u(n)$ be the non derivative term of equation (2.1). Now let $u_j = u(n_j)$ [1] be its values at the grid points $n_j\, jM/2, \ldots, M/2 - 1$. Next, we define the Discrete Fourier Transformation as:

[1] Note that this notation here doesn't mean the derivative as it's the case with u_x and u_t in equation (2.1).

Definition 3 *Given a sequence of N numbers defined as x_n with Δx as the distance between the grid points, the Discrete Fourier Transform (DFT) of x_n is given by:*

$$\hat{x}_n = \mathcal{F}(x_n) = \sum_{-N/2}^{N/2-1} e^{-ij\frac{2\pi\Delta x}{N}} x_j.$$

Similarly there also follows the definition of the inverse:

Corollary 1 *The Inverse Discrete Fourier Transform (IDFT) is given by:*

$$x_n = \mathcal{F}^{-1}(\hat{x}_n) = \frac{1}{N} \sum_{-N/2}^{N/2-1} e^{ijk\frac{2\pi\Delta x}{N}} \hat{x}_j.$$

Applying the Discrete Fourier Transform on our discretized version of u with a $\Delta x = M/L$ for a domain length L gives us the following Fourier interpolant at the defined discretized space:

$$\hat{u} = \mathcal{F}(u) = \sum_{-M/2}^{M/2-1} e^{-ijk\frac{2\pi}{L}} u_j.$$

Where u_j depicts the value of u for each gridspot $j = M/2, \ldots, M/2-1$. Similarly, the derivative of x over u, is defined by the derivatives of the truncated interpolant. Thus we get:

$$\mathcal{F}(u_x) = \hat{u}_x = \sum_{-M/2}^{M/2-1} (-i\frac{2\pi jk}{L}) e^{-ijk\frac{2\pi}{L}} u_j = \sum_{-M/2}^{M/2-1} (-i\frac{2\pi jk}{L})\hat{u}. \tag{2.4}$$

And by repeatedly taking the derivative it follows that:

$$\mathcal{F}(u_{xx}) = \sum_{-M/2}^{M/2-1} -(\frac{2\pi}{L})^2 e^{-ijk\frac{2\pi}{L}} u_j = \sum_{-M/2}^{M/2-1} -(\frac{2\pi}{L})^2 \hat{u}, \tag{2.5}$$

and finally that:

$$\mathcal{F}(u_{xxxx}) = \sum_{-M/2}^{M/2-1} (\frac{2\pi i}{L})^4 e^{-ijk\frac{2\pi}{L}} u_j = \sum_{-M/2}^{M/2-1} (\frac{2\pi i}{L})^4 \hat{u}. \tag{2.6}$$

Notice that by applying equation (2.5) and equation (2.6), the linear operator of equation (2.2) can be rewritten as:

$$\begin{aligned}
\mathcal{F}(Au) &= -\mathcal{F}(u_{xx}) - \mathcal{F}(u_{xxxx}) \\
&= \sum_{-M/2}^{M/2-1} (\frac{2\pi}{L})^2 \hat{u} - \sum_{-M/2}^{M/2-1} (\frac{2\pi i}{L})^4 \hat{u} \\
&= (\sum_{-M/2}^{M/2-1} ((\frac{2\pi}{L})^2 - (\frac{2\pi i}{L})^4))\hat{u}
\end{aligned}$$

Which we can rewrite as:

$$\mathcal{F}(Au) = \mathcal{A}\hat{u}. \tag{2.7}$$

Here is $\mathcal{A} = (\sum_{-M/2}^{M/2-1} ((\frac{2\pi}{L})^2 - (\frac{2\pi i}{L})^4))$ the projection of the operator A in Fourier space. With equation (2.7) established, we are now able to use the Implicit-Explicit

Runge-Kutta spectral method for numerical calculations as described in the following section.

2.2 The Implicit-Explicit Runge-Kutta method

One aspect that makes the Kuramoto-Sivashinsky equation computationally expensive to calculate is that typically the maximum stable step size of an explicit numerical method is inversely proportional to the largest eigenvalue of the linear terms, while implicit numerical methods are generally computationally heavy. In this case, since we have a fourth-order term, it means that we would need a very small time step to solve this function explicitly. To deal with this implication, we will make use of the Implicit-Explicit Runge-Kutta method as described in [1]. How this method makes the calculation relatively efficient will be shown in section 2.3. In this section we will go over the definition of the Implicit-Explicit Runge-Kutta method and its application into numerically solving the Kuramoto-Sivashinsky equation.

We start by defining the Implicit-Explicit Runge-Kutta (IMEX RK) method as follows:

Definition 4 *Given a known x_n at time n and the functions $f(x)$, $g(x)$ such that $\dot{x} = f(x) + g(x)$. We define the Implicit-Explicit Runge-Kutta midpoint (IMEX RK) method to approximate x_{n+1} as:*

$$x_{n+\frac{1}{2}} = x_n + \frac{\tau}{2} f(x_{n+\frac{1}{2}}) + \frac{\tau}{2} g(x_n), \tag{2.8}$$

$$x_{n+1} = x_n + \tau f(x_{n+\frac{1}{2}}) + \tau g(x_{n+\frac{1}{2}}), \tag{2.9}$$

where τ denotes the size of a discrete time step from x_n to x_{n+1}.

Applying this method to the Kuramoto-Sivashinsky equation as seen in equation (2.1), with the linear equation (2.2) as f and the nonlinear equation (2.3) as g and an arbitrary time index n, we get the following equations:

$$u_{n+\frac{1}{2}} = u_n + \frac{\tau}{2} A u_{n+\frac{1}{2}} + \frac{\tau}{2} f(u_n), \tag{2.10}$$

$$u_{n+1} = u_n + \tau A u_{n+\frac{1}{2}} + \tau f(u_{n+\frac{1}{2}}). \tag{2.11}$$

Note however, that in the half-step (2.10), in order to calculate $u_{n+\frac{1}{2}}$, we make use of the linear operator $Au_{n+\frac{1}{2}}$ while the value is still unknown. To take this into account, we take $Au_{n+\frac{1}{2}}$ into Fourier space by applying the discrete Fourier transformation as described in equation (2.7). By doing this, we can now do matrix calculations by splitting A of $u_{n+\frac{1}{2}}$, assuming that we do this inside Fourier space. Thus we can rewrite equation (2.10) as:

$$u_{n+\frac{1}{2}} - \frac{\tau}{2} A u_{n+\frac{1}{2}} = u_n + \frac{\tau}{2} f(u_n),$$

so that we get

$$(I - \frac{\tau}{2} A) u_{n+\frac{1}{2}} = u_n + \frac{\tau}{2} f(u_n).$$

Left multiplying with $(I - \frac{\tau}{2} A)^{-1}$ gives us finally that:

$$u_{n+\frac{1}{2}} = (I - \frac{\tau}{2} A)^{-1} (u_n + \frac{\tau}{2} f(u_n)). \tag{2.12}$$

Wherein $(I - \frac{\tau}{2}A)^{-1}$ is calculated inside of the Fourier space. For the actual implementation of this calculation using a pseudospectral method, the reader can find the evaluation in section A.1 of the Appendix. By solving equation (2.12) and using its solution into equation (2.11), we can now propagate u_t over t and thus given a u_0 we can now approximate a solution for the Kuramoto-Sivashinsky equation.

2.3 Properties of the IMEX RK method

In this section we shall take a look at some interesting properties of the IMEX RK method. We will specifically look at the truncation and global error of this method, and then we will show what makes this method more efficient than regular implicit and explicit methods for this problem. If the reader wants to read further about the process of establishing the numerical methods to approximate the attractor dimension, then the reader can continue safely from chapter 3 onward.

We start by taking a look at the truncation and global errors. The IMEX RK method works by the approximation of a half step which is then used into the approximation of the full step. The truncation error of the half-step can be found as follows: Suppose $v(t)$ to be the exact solution of the Kuramoto-Shivanchinsky equation, then taking the half step (2.12) we see that:

$$\begin{aligned} &v(t_{n+1}) - v(t_i) - \tau(I - \frac{\tau}{2}A)^{-1}(v_n + \frac{\tau}{2}f(v_n)) \\ &= 0 - \mathcal{O}(\tau^2). \end{aligned}$$

Whereas, for the full step the truncation error follows from a multiplication with another τ:

$$\begin{aligned} &v(t_{n+1}) - v(t_i) - \tau(v_n + \tau A v_{n+\frac{1}{2}} + \tau f(v_{n+\frac{1}{2}})) \\ &= 0 - \mathcal{O}(\tau^3). \end{aligned}$$

The global error for the full step follows to be $\mathcal{O}(\tau^2)$. So as you can see, the IMEX RK method works by taking a more accurate half step to approximate a lesser accurate full step, making the higher error on the full step less important.

Finally to show that the IMEX RK method is computationally more efficient than regular explicit methods, note that in equation (2.10) we only calculate the nonlinear part of the equation explicitly. As the maximum stable step size of an explicit numerical method is inversely proportional to the largest eigenvalue of the linear term, we are only limited to a step size that is limited by the term u_x (the linear product of the nonlinear part of the equation). Thus by solving the linear operator of the equation implicitly, which is still computationally expensive, we do save on overflow problems that would happen with explicit-only methods.

Chapter 3

Approximating the Lyapunov exponents

Now that we are able to numerically approximate a solution for the Kuramoto-Sivashinsky equation, our next step is to use that solution to approximate the Lyapunov exponents. There are many methods for the approximation of Lyapunov exponents for linear PDEs, but as the Kuramoto-Sivashinsky equation is nonlinear, we will need to make use of a specialized method. This is why we will make use of the QR method as is described in [2].

We start this chapter by defining prelimaries in section 3.1. Then we will describe the QR method in section 3.2 and finally, in section 3.3, we will go over the numerical implementation of this method by applying the IMEX RK method as we described in definition 4.

3.1 Preliminaries

For this section we follow the Introduction section from [2], where we modify the values to fit our problem. Let $g(u) = u_t$ be the Kuramoto-Sivashinsky equation as defined in equation (2.1) and let $u(t)$ be its solution at timestep t. Define now the linearized problem

$$\dot{Y} = g'((u(t)) = G(t)Y. \tag{3.1}$$

where $G(t) \in \mathcal{R}^{m\times m}$ is a bounded and continuous function and $Y(0) = Y_0$ as an invertible matrix. Define the numbers $\mu_j, \quad j = 1, \ldots, m$ as:

$$\mu_j = \limsup_{t\to\infty} \frac{1}{t} \log ||Y(t)e_j||, \tag{3.2}$$

where e_j are the standard unit vectors and $Y(t)$ is the solution of

$$\dot{Y} = G(t)Y. \tag{3.3}$$

By minimizing the sum of all μ_j over all possible initial conditions Y_0, then the μ_j become the upper Lyapunov exponents of the system. We order these upper Lyapunov exponents and define them as $\lambda_j^s, \quad j = 1, \ldots, m$. In the same way we define the the ordered lower Lyapunov exponents as $\lambda_j^i, \quad j = 1, \ldots, m$ which we derive from the following linearized problem:

$$\dot{Z} = -G^T(t)Z.$$

Now we define the following definition:

Definition 5 *A system $\dot{Y} = G(t)Y$ consisting of upper Lyapunov exponents as λ_j^s and lower Lyapunov exponents as λ_j^i is called **regular** if the following conditions hold:*

1. $\lambda_j^i = \lambda_j^s = \lambda_j$,
2. $\lim_{t\to\infty} \frac{1}{t} \log(\det(Y(t))) = \sum_{j=1}^{m} \lambda_j$.

Finally, for a regular system we have the following theorem as defined by [7]:

Definition 6 *If we have a regular system with upper triangular coefficient matrix $B(t)$ such that $\dot{y} = B(t)y$, then, for $i = 1, \ldots, m$, its Lyapunov exponents are given by:*

$$\lambda_i = \lim_{t\to\infty} \frac{1}{t} \int_0^t B_{ii}(s)ds \tag{3.4}$$

Regularity is formally a requirement to stably compute the Lyapunov exponents, but proving that this system is indeed regular is out of the scope for this thesis. Thus for the rest of this paper we will assume that we are indeed working with a regular system.

3.2 QR Decomposition

With definition 6 established to derive the Lyapunov exponents, we now need to find a upper triangular coefficient matrix $B(t)$ such that $\dot{y} = B(t)y$ to comply with the definition. For this we will be following the 'Continuous QR method' section from [2]. We start by taking the QR decomposition $Y = QR$. Applying this to equation (3.3) gives us

$$\dot{Q}R + Q\dot{R} = G(t)QR,$$

Multiplying to the left with Q^T gives us:

$$Q^T\dot{Q}R + \dot{R} = Q^TG(t)QR,$$

which we can rewrite as

$$\dot{R} = BR. \tag{3.5}$$

where $B = Q^TG(t)Q - Q^T\dot{Q}$. Since the R part of a QR decomposition is upper triangular, the change $\dot{R}$ must also be upper triangular and thus B must also be upper triangular. As B is upper triangular and our system is regular, we can now use definition 3.4 to approximate the Lyapunov exponents. Notice however that we have $\dot{Q}$ in our definition of B, making it a ODE that we first would need to solve. We can work around this by taking $S = Q^T\dot{Q}$ such that:

$$B = Q^TG(t)Q - S.$$

Now since $Q^TQ = I$, S is skew-symmetric, and since B is upper triangular, it follows that the strict lower triangular part of S must be equal to the strict lower part of $Q^TG(t)Q$. And since S is skew-symmetric, we define the upper triangular part as the negative of the lower triangular part, giving us the following definition for S:

$$S_{ij} = \begin{cases} (Q^TGQ)_{ij} & i > j \\ 0 & i = j \\ -S_{ji} & i < j \end{cases} \tag{3.6}$$

With this definition of S, we can now calculate B without solving $\dot{Q}$. However, we can make the calculation even more efficient, because since we are only interested in the diagonal of B, and S has a diagonal of 0. It follows that

$$B_{ii} = (Q^T G(t) Q)_{ii}. \tag{3.7}$$

Thus we have now defined B_{ii} in terms of $G(t)$ and Q. In chapter 2 we have already seen how we can numerically approximate $G(t)$ and in the following section we will derive a method to numerically approximate Q.

3.3 Approximating Q using the IMEX RK method

In this section we will derive a method for when given the linearized problem (3.3), to numerically approximate Q using the QR decomposition of $Y = QR$. We start by rewriting equation (3.3) as follows:

$$\dot{Y} = G(t)Y = (L + f'(u(t)))Y = LY + f'(u(t))Y.$$

Applying the IMEX RK method as in definition 4 gives us:

$$Y_{n+\frac{1}{2}} = Y_n + \frac{\tau}{2} L Y_{n+\frac{1}{2}} + \frac{\tau}{2} f'(u(t)) Y_n, \tag{3.8}$$

$$Y_{n+1} = Y_n + \tau\, L Y_{n+\frac{1}{2}} + \tau\, f'(u(t)) Y_{n+\frac{1}{2}}. \tag{3.9}$$

Next we rewrite equation (3.8) as

$$Y_{n+\frac{1}{2}} - \frac{\tau}{2} L Y_{n+\frac{1}{2}} = Y_n + \frac{\tau}{2} f'(u(t)) Y_n.$$

Taking $LY_{n+\frac{1}{2}}$ into Fourier space using the DFT as defined in definition 3, lets us split off the matrix L of the approximated $Y_{n+\frac{1}{2}}$ giving us:

$$(I - \frac{\tau}{2} L) Y_{n+\frac{1}{2}} = (I - \frac{\tau}{2} f'(u(t))) Y_n,$$

Multiplying to the left with $(I - \frac{\tau}{2} L)^{-1}$ gives us then

$$Y_{n+\frac{1}{2}} = (I - \frac{\tau}{2} L)^{-1} (I - \frac{\tau}{2} f'(u(t))) Y_n,$$

which we write shortened as

$$Y_{n+\frac{1}{2}} = C_n Y_n, \tag{3.10}$$

where $C_n = (I - \frac{\tau}{2} L)^{-1} (I - \frac{\tau}{2} f'(u(t)))$.

Notice that by substituting equation (3.10) into equation (3.9), we can rewrite the equation as:

$$Y_{n+1} = Y_n + \tau\, L C_n Y_n + \tau\, f'(u(t)) C_n Y_n.$$

And if we take this equation into Fourier space using DFT again, we can split off the matrices such that:

$$Y_{n+1} = (I + \tau\, C_n\, (L + f'(u(t))))Y_n,$$

where $(L + f'(u(t))$ is calculated inside of Fourier space. Finally we shorten the equation in as

$$Y_{n+1} = D_n Y_n. \tag{3.11}$$

where $D_n = (I + \tau C_n \mathcal{F}(L + f'(u(t))))$.

Using equation (3.10) and equation (3.11) we can thus obtain Y_{n+1} when Y_n is known. To be able to do this calculation, however, we need to calculate the derivative $f'(u(t))$. This will be done using the finite difference method for a small $\epsilon > 0$ and for each column $q \in Q$ such that:

$$\mathrm{f'(u)\ q} = \frac{\mathrm{f(u + \epsilon q) - f(u)}}{\epsilon}. \tag{3.12}$$

Now that we have defined a method for the propagation of Y_n over time, we take the QR decomposition of Y_n as $Y_n = Q_n R_n$. Substituting this into equation (3.11) gives us:

$$Q_{n+1} R_{n+1} = D_n Q_n R_n$$

which we rewrite as

$$\tilde{Y}_{n+1} = Q_{n+1} \tilde{R}_{n+1} = Q_{n+1} R_{n+1} R_n^{-1} = D_n Q_n.$$

By taking a new QR decomposition of $\tilde{Y}_{n+1}$ to split of Q_{n+1}, we can now propagate for each step for Q_{n+1} when a Q_n and A_n is known for $0 < n < t_{max}$.

Thus to summarize, we can numerically approach Q by starting with a randomly defined initial Q_0 and then by propagating each Q_n to Q_{n+1} for $0 < n < t_{max}$. To do this we first calculate $\tilde{Y}_{n+1}$ by calculating each column $\tilde{q}_{n+1} \in \tilde{Y}_{n+1}$ using the following method:

$$\tilde{q}_{n+\frac{1}{2}} = (I - \frac{\tau}{2} L)(I - \frac{\tau}{2} f'(u(t)))\tilde{q}_n = C_n \tilde{q}_n, \tag{3.13}$$

$$\tilde{q}_{n+1} = (I + \tau C_n \mathcal{F}(L + f'(u(t))))\tilde{q}_n. \tag{3.14}$$

And then we apply a QR decompostion on $\tilde{Y}_{n+1} = Q_{n+1} \tilde{R}_{n+1}$ to split of Q_{n+1}.

Chapter 4

Numerical experiments

In the previous chapters we have discussed various numerical methods and established a workflow for the calculation of the attractor dimension. In this chapter we will look at the results obtained by the mentioned numerical methods for different initial conditions, specifically at its scalability according to the domain length L and the grid size M. To see how the mathematical concepts of the previous chapters have been converted to code, the reader can look in appendix A.

4.1 Dependence on the domain length

As mentioned in the introduction, it has been already found by [3] that the size of the Lyapunov attractor depends on the domain size L. Our calculations in figure 4.1 are able to confirm these results.

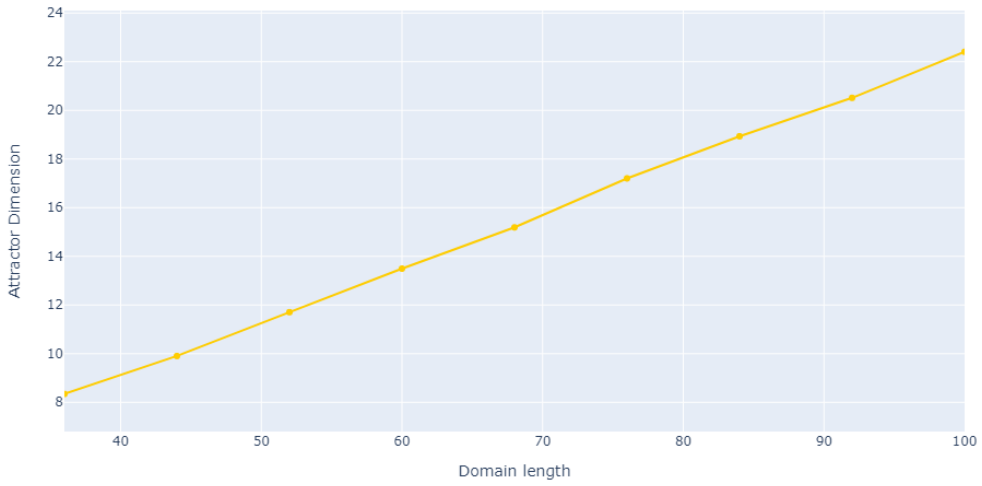

FIGURE 4.1: The approximated attractor dimension plotted over different values for the domain length. Calculated at $M = 100$, $tn = 1000$ and $dt = 0.001$.

This phenomenon can be explained as the Lyapunov exponents that are needed for the attractor dimension are calculated by using the solution of the initial equation. The increase and decrease in L appear to produce a different chaotic pattern in the result, as can be seen in Figure 4.2.

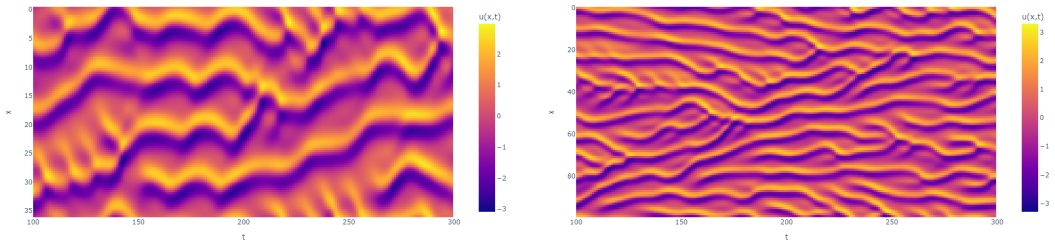

FIGURE 4.2: The approximated solution of the Kuramoto-Sivashinsky equation plotted for $L = 36$ (left) and $L = 100$ (right).

4.2 Dependence on the grid size

Unlike the domain length, the grid size produces no difference in the chaotic pattern of the solution of the KS equation. However, the number of total computed Lyapunov exponents depends directly on the size of the grid size. This is why we initially asked the question of how the scaling of the attractor depends on the grid size. The results can be seen in figure 4.3.

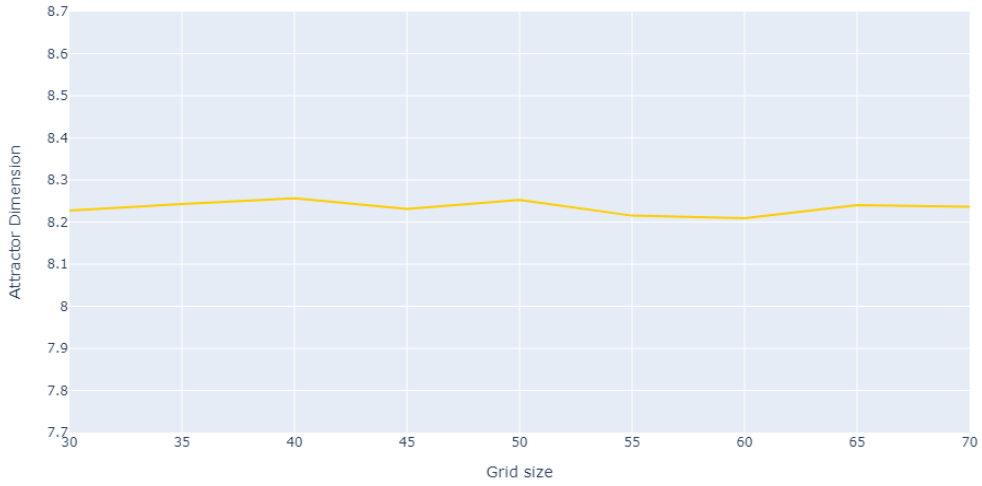

FIGURE 4.3: The approximated attractor dimension plotted over different values for the grid size. Calculated at $L = 36$, $tn = 2000$ and $dt = 0.0005$.

It seems that the size of the attractor dimension is independent of the grid size M, even though there are more Lyapunov values calculated for a bigger M. To see how this reflects the Lyapunov values themselves, we show in figure 4.4 and figure 4.5 the development of the Lyapunov values on different M. We see that the larger the grid size, the larger the set of the smallest Lyapunov exponents grows with an exponentially smaller size for the smallest exponent.

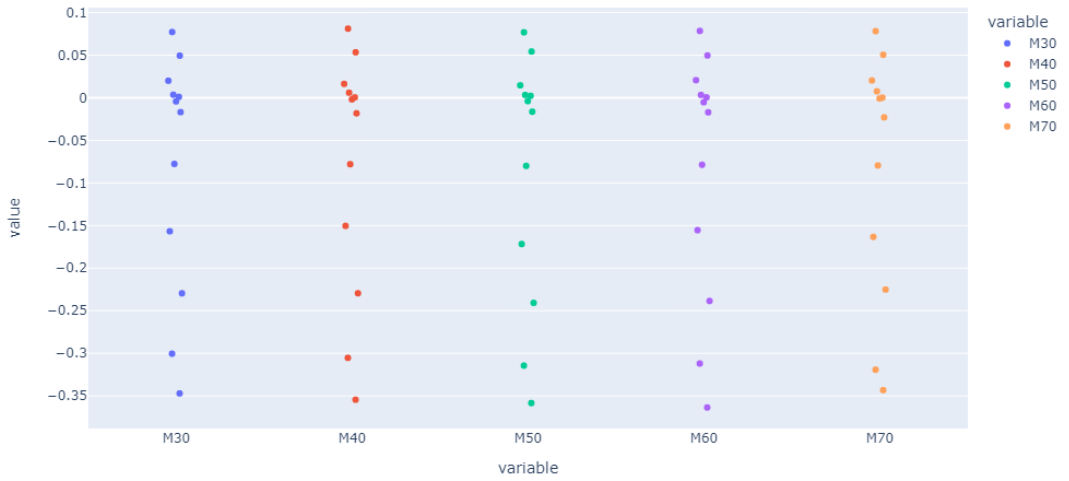

FIGURE 4.4: Comparison of the 12 biggest Lyapunov exponents for values of $30 \leq M \leq 70$.

Number of descending sorted LE	M = 30	M=50	M=70
28	-29,58	-29,64	-29,64
29	-29,63	-29,64	-29,64
30	-39,89	-40,09	-40,09
⋮		⋮	⋮
48		-290,24	-290,30
49		-290,32	-290,30
50		-343,22	-343,42
⋮			⋮
68			-1.203,80
69			-1.205,92
70			-1.355,16

FIGURE 4.5: Comparison of the smallest Lyapunov exponents for $M = 30$, $M = 50$ and $M = 70$.

4.3 Analysis on other equations

Finally, we checked whether the numerical methods described in this thesis also work for other PDEs. For this we have looked at other similar PDEs that consist of a linear and nonlinear operator, specifically the Burgers' equation and the Korteweg-De Vries equation. However, since these two are Hamiltonian systems that require energy-conserving time discretization, they do not have an asymptotically stable attractor. Therefore, the method that we used for the Kuramoto-Sivashinsky equation doesn't seem to be a good match for these particular PDEs. What makes the Kuramoto-Sivashinsky equation special compared to the two equations described above is that the Kuramoto-Sivashinsky equation is one of the few PDEs to have a finite dimensional invariant manifold [4], meaning that the solutions of the Kuramoto-Sivashinsky equation have a subspace wherein they are attracted, and thus if there is an attractor, its support must be in that subspace.

Chapter 5

Conclusion

In this thesis we have seen how we can numerically approximate the attractor dimension of the Kuramoto-Sivashinsky equation by approximating the Lyapunov exponents. To approximate these Lyapunov exponents we needed to numerically approximate the solution of the discretized version of the Kuramoto-Sivashinsky equation $G(t)$ and we needed to numerically approximate the component Q from QR decomposing the solution $Y(t)$ which is the solution of the linearized problem $\dot{Y} = G(t)Y$. We have seen how we could do both of these numerical approximations by applying the Implict-Explicit Runge-Kutta numerical method to propagate both values over time, given an initial solution and an initial value for Y.

We then applied our numerical method in Python to numerically approximate the attractor dimension for different values of the domain length and a fixed grid size, and for different values for a grid size and a fixed domain length. We have seen from these numerical experiments that the scaling of the size of the attractor dimension in the discretized Kuramoto-Sivashinsky equation is proportional to the domain length, where a larger domain results in a larger attractor dimension. We have also seen that the size of the grid size has no effect on the approximated attractor dimension as an increased grid size only added more negative Lyapunov exponents.

Finally, we took a look at the application of the discussed numerical methods to other similar PDEs which consist of a linear and a nonlinear part. We concluded that the discussed numerical methods applied for these PDEs only work if the PDEs in question also have a finite dimensional invariant manifold and therefore actually have an attractor.

Appendix A

Implementation

In this chapter we will go over how the numerical methods that we defined in chapter 2 and chapter 3 are applied in Python code. We shall go over the code snippet by snippet to discuss the implementation at length. The reader can find the full files of the discussed Python code on the following GitHub page.

A.1 Solving the Kuramoto-Sivashinsky equation

We start by going over the implementation of the numerical method as described in chapter 2. As described in section 2.1, we make the calculations of the derivatives inside of Fourier space. For this we discretize the derivatives over M for a j such that we have $j = M/2, \ldots, M/2 - 1$ grid points. Then we use a pseudospectral method to evaluate each term of the Kuramoto-Sivashinsky equation. Below you see the implementation for the diagonals of these discretised gridpoints inside of Fourier space for the different terms as described in equation (2.4), equation (2.5) and equation (2.6).

LISTING A.1: Discretization of the derivatives in Fourier space.

```
D    = np.linspace(-M//2,M//2,M,endpoint=False)
Dx   = np.fft.fftshift(np.multiply(D,(2*np.pi/L)))*np.sqrt(-1+0j)
Dxx  = -np.fft.fftshift(np.power(np.multiply(D,(2*np.pi/L)),2))
Dxxxx = np.fft.fftshift(np.power(np.multiply(D,(2*np.pi/L)),4))
```

Next, we evaluate each term of the Kuramoto-Sivashinsky equation by pointwise multiplying each diagonal with with the DFT of u, for this we define the following functions:

LISTING A.2: functions for the derivatives.

```
def u_x(u):
    return np.real(np.fft.ifft(np.multiply(Dx,np.fft.fft(u))))

def u_xx(u):
    return np.real(np.fft.ifft(np.multiply(Dxx,np.fft.fft(u))))

def u_xxxx(u):
    return nu*np.real(np.fft.ifft(np.multiply(Dxxxx,np.fft.fft(u))))

def Lu(u): # Linear part of the KS equation
    return (-u_xx(u) - u_xxxx(u))

def fu(u): # Nonlinear part of the KS equation
    return (-1)*np.multiply(u, u_x(u))
```

Here we make use of the Fast Fourier Transform (FFT) algorithm to take u into the Fourier space instead of DFT as described in section 2.1. We do this because applying this algorithm is more efficient ($\mathcal{O}(M \log M)$ rather than DFT ($\mathcal{O}(M^3)$). Now with the definition for each derivative defined, we calculate each new time-step as defined in section 2.2:

LISTING A.3: IMEX RK method to propagate the solution through time.

```
def u_half(u):
    k_lin = -Dxx - Dxxxx
    G     = np.fft.fft(np.add(np.multiply((0.5*dt),fu(u)), u))
    Uhat  = np.divide(G,(1-np.multiply((0.5*dt),k_lin)))

    return np.real(np.fft.ifft(Uhat))

def u_plus1(u, u_half):
    return u + np.multiply(dt,Lu(u_half))
             + np.multiply(dt,fu(u_half))
```

Note that we don't apply the method as described in equation (2.12) using matrix calculation to calculate the inverse, but rather we take the rest of the equation into Fourier space to make a division inside of Fourier space. This saves us on calculation time and other complications. With the functions derived to propagate through each timestep, we can now approximated the solution by applying a for loop from $t = 0$ to the maximal desired value tn.

A.2 Approximating the diagonal B_{ii}

By applying the method as described at the end of section 3.3, we can propagate a random defined Q_0 over time using the following definitions:

LISTING A.4: IMEX RK method to propagate the value of $q \in Q$ through time.

```
def q_half(u, q):
    k_lin = -Dxx - Dxxxx
    G = np.fft.fft(q + 0.5*dt*fu_dotq(u,q))
    Qhat = np.divide(G,(1-np.multiply((0.5*dt), k_lin)))

    return np.real(np.fft.ifft(Qhat))

def q_plus1(u_half, q, q_half):
    return q + np.multiply(dt,fu_dotq(u_half, q_half))
             + np.multiply(dt,Lu(q_half))
```

Here, we approximate fu_dotq as defined by equation 3.12 using the following code:

LISTING A.5: finite difference method to approximate f'(u) q.

```
def fu_dotq(u, q):
    return (fu(u+eps*q) - fu(u))/eps
```

Finally, we orthogonalize Y using QR factorization using the following implementation of the modified Gramm Schmidt method:

LISTING A.6: Implementation of the Modified Gramm Schmidt method where only the value of Q of the QR Decomposition is saved.

```
def MGS(Q):
    for m in range(len(Y[0])):
        for j in range(0, m):
            R_mj =  np.vdot(Q[:,j].transpose(),Q[:,m])
            Q[:,m] = Q[:,m] - R_mj*Q[:,j]

        R_mm   = np.linalg.norm(Q[:,m])
        Q[:,m] = Q[:,m]/R_mm

    return Q
```

Taking the functions listed above together with the functions in section A.1, we can propagate the solution of u and the approximation of Q over time using the following loop to calculate the diagonal of B_{ii} for each time step:

LISTING A.7: The main loop used for the propagation of solution u and the approximation of Q.

```
for i in range(nt-1):
        u_i = u[:,i]

        # IMEX RK for calculating u_{i+1}
        u_ihalf  = u_half(u_i)
        u[:,i+1] = u_plus1(u_i, u_ihalf)

        # Solution space
        A_iQ_i = np.zeros(Q.shape)

        # Iterate through each column in Q
        for j in range(len(Q[0])):
            q_j          = Q[:,j]
            q_ihalf      = q_half(u_i, q_j)
            A_iQ_i[:,j] = q_plus1(u_ihalf, q_j, q_ihalf)

        # Compute Q_{i+1} as the QR factorization of A_iQ_i
        Q = MGS(A_iQ_i)

        # Take the diagonal Bii for the next time step.
        B_ii[:,i] = np.dot(Q.transpose(),AQ(u[:,i+1], Q)).diagonal()
```

A.3 Approximating the Lyapunov exponents

With the implementation defined for the IMEX RK numerical method to approximate A and Q, we can now approximate the Lyapunov exponents as defined in equation (3.4). As we are working in a discrete system, we are working with a bounded maximum value for t in a discrete function. Thus, we will calculate the Riemann sum of this integral for a t_{max} using the following algorithm:

LISTING A.8: Approximate the Lyapunov exponents given the values of the diagonal B_{ii} over time t to tn.

```
def calc_Lyapunov(B_ii):

    # Solution space
    lambda_i = np.zeros(len(B_ii[:,0]))

    # Calculate the integral as a Riemann sum.
    for i in range(len(B_ii[:,0])):
        for t in range(len(B_ii[0])):
            lambda_i[i] = lambda_i[i] + B_ii[i][t]*(dt)

    # Divide by t
    lambda_i = lambda_i / (tn-t0)

    return lambda_i
```

A.4 Calculate the attractor dimension

With the Lyapunov exponents calculated, we can finally calculate the dimension using the following implementation of equation (1.2):

LISTING A.9: Algorithm for the calculation of the Lyapunov dimension, given a set of Lyapunov exponents.

```
def calc_D_L(lambda_i):

    # Sort the lyapunov exponents from the biggest to the smallest.
    lambda_sorted = -np.sort(-lambda_i)

    # Find the maximum k.
    k = 0
    for i in range(1, len(lambda_sorted)):
        if (sum(lambda_sorted[:i]) <= 0):
            break
        k=i

    # If the dimension can't be calculated, return -1.
    if (k == len(lambda_sorted)-1):
        D_L = -1
    else:
        # Calculate the Lyapunov dimension.
        D_L = k + (np.sum(lambda_sorted[:k])
                / abs(lambda_sorted[k+1]))
    return D_L
```

Bibliography

[1] Uri M. Ascher, Steven J. Ruuth, and Raymond J. Spiteri. "Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations". In: *Applied Numerical Mathematics* 25.2 (1997), pp. 151–167. URL: https://www.sciencedirect.com/science/article/pii/S0168927497000561.

[2] Luca Dieci, Michael Jolly, and Erik van Vleck. "Numerical Techniques for Approximating Lyapunov Exponents and Their Implementation". In: *Journal of Computational and Nonlinear Dynamics* 6 (Jan. 2011). URL: https://www.researchgate.net/publication/228613270_Numerical_Techniques_for_Approximating_Lyapunov_Exponents_and_Their_Implementation.

[3] Russell A. Edson et al. "Lyapunov exponents of the Kuramoto-Sivashinsky PDE". In: (2019). URL: https://arxiv.org/abs/1902.09651.

[4] M.S. Jolly, R. Témam, and C. Xiong. "Convergence of a chaotic attractor with increased spatial resolution of the Ginzburg-Landau equation". In: *Chaos, Solitons Fractals* 5.10 (1995). Structure Formation in Continuous Dynamical Systems, pp. 1833–1845. ISSN: 0960-0779. URL: https://www.sciencedirect.com/science/article/pii/096007799400197X.

[5] James L. Kaplan and James A. Yorke. "Ordinary differential equations which yield periodic solutions of differential delay equations". In: *Journal of Mathematical Analysis and Applications* 48.2 (1974), pp. 317–324. URL: https://www.sciencedirect.com/science/article/pii/0022247X74901620.

[6] Y. Kuramoto and T. Tsuzuki. "Persistent Propagation of Concentration Waves in Dissipative Media Far from Thermal Equilibrium". In: *Progress of Theoretical and Experimental Physics* (1976), 356–369. URL: https://academic.oup.com/ptp/article/55/2/356/1883646?login=false.

[7] A. Lyapunov. "Problème général de la stabilité du mouvement". In: *Annales de la Faculté des sciences de l'Université de Toulouse pour les sciences mathématiques et les sciences physiques* 9 (1907), pp. 203–474. URL: http://www.numdam.org/item/?id=AFST_1907_2_9__203_0.

[8] G.I. Sivashinsky. "Nonlinear analysis of hydrodynamic instability in laminar flames—I. Derivation of basic equations". In: *Acta Astronautica* 4.11 (1977), pp. 1177–1206. URL: https://www.sciencedirect.com/science/article/pii/0094576577900960.