Efficient algorithms for rigorous integration forward in time of dPDEs. Existence of globally attracting fixed points of viscous Burgers equation with constant forcing, a computer assisted proof

Jacek Cyranka
Institute of Computer Science, Jagiellonian University
prof. Stanisława Łojasiewicza 6, 30-348 Kraków, Poland
jacek.cyranka@ii.uj.edu.pl

December 4, 2012

Contents

0.1 Prologue 1

1 Efficient algorithms for rigorous integration forward in time of dPDEs. 2

1.1 Introduction 2
1.2 Fast Fourier Transform algorithm 3
1.2.1 Outline of approach 4
1.2.2 Some elementary facts about the trigonometric interpolation on uniform mesh 4
1.2.3 Interpolation with phase shift 6
1.2.4 Uniform matrix form of the FFT 7
1.2.5 Factorization into sparse matrices 9
1.2.6 Variants of the FFT algorithm 17
1.2.7 Complexity argument 23
1.3 Issues arising when the FFT algorithm is used 25
1.3.1 Aliasing problem 25
1.3.2 Overestimates 30
1.4 A rigorous DPDE integrator 35
1.4.1 Automatic differentiation 36
1.4.2 Jet transport 38
1.4.3 An algorithm combining the Automatic Differentiation with the FFT 39
1.4.4 Optimizations 43
1.4.5 Comment on Automatic Differentiation Convolutions . . . 48
1.5 Complexity discussion 49
1.5.1 Core complexity 49
1.5.2 Explicit operations count 50
1.6 Rigorous numeric tests 53
1.7 Notions on Software 62

2 Existence of globally attracting fixed points of viscous Burgers equation with constant forcing, a computer assisted proof 65

2.1 Introduction 65
2.2 The viscous Burgers equation 67
2.2.1 The viscous Burgers equation in the Fourier basis 67

2.3 Analytic arguments 68
2.3.1 Energy as Lyapunov function 68
2.3.2 A trapping region for (2.5a) 70
2.4 Global results 72
2.5 General method of self-consistent bounds. 77
2.5.1 Self-consistent bounds 78
2.6 Local existence and uniquenes 80
2.7 Proof of Theorem 2.1.1 81
2.8 Algorithm for constructing an absorbing set for any Galerkin projection of (2.5a) 83
2.9 Rigorous integration forward in time 85
2.9.1 Algorithm for integrating rigorously dPDEs 86
2.10 Algorithm for proving Theorem 2.1.1 92
2.11 Conclusion 94
.1 Data from the example proof 94
.2 Validate tail function in pseudo-code 96

Abstract

The dissertation is divided into two separate parts.

First part We propose an efficient and generic algorithm for rigorous integration forward in time of systems of equations originating from partial differential equations written in the Fourier basis. By rigorous integration we mean a procedure, which operates on sets, and return sets which are guaranteed to contain the exact solution. The algorithm is generating, in an efficient way, normalized derivatives, which then are used by the Lohner algorithm to produce a rigorous bound. Algorithm has been successfully tested on several PDEs including the Burgers equation, Kuramoto-Sivashinsky equation and Swift-Hohenberg equation. Problem of rigorous integration in time of partial differential equations is a problem of large computational complexity, and efficient algorithms are required to deal with PDEs on higher dimensional domains, like the Navier-Stokes equation.

Second part We present a computer assisted method for proving the existence of globally attracting fixed points of dissipative PDEs. An application to the viscous Burgers equation with periodic boundary conditions and a constant in time forcing function is presented as a case study. We establish the existence of a locally attracting fixed point by using computer techniques based on the method of self-consistent bounds. To prove that the fixed point is, in fact, globally attracting we introduce a technique relying on construction of an absorbing set, capturing after a finite time any sufficiently regular initial condition. Then the absorbing set is rigorously integrated forward in time to verify that any sufficiently regular initial condition is in the basin of attraction of the fixed point.

Keywords: computer assisted proof, dissipative PDE, rigorous numeric, automatic differentiation, fast Fourier transform, FFT, long time behavior, attractor, interval arithmetic, Burgers equation

AMS classification: Primary: 65M99, 35B40. Secondary: 35B41, 65G20, 65M70, 65Y20, 65T50

0.1 Prologue

The dissertation is divided into two separate parts, each of them being self-contained. Intersection of the two parts is that the results can be seen as a preparation to the rigorous numerical analysis for the Navier-Stokes equations and other higher dimensional PDEs.

Chapter 1

Efficient algorithms for rigorous integration forward in time of dPDEs.

1.1 Introduction

The main goal of this part of the dissertation is to present an efficient algorithm for rigorous integration forward in time of dissipative partial differential equations (dPDEs). We have added to the title the phrase "forward in time" because dPDEs cannot be integrated backward in time, due to blow-ups of solutions. The proposed approach combines several techniques. We present issues arising when an algorithm based on the proposed approach is used for rigorous calculations and we present methods of circumventing the issues. We have performed several rigorous numeric tests to show that the algorithm leads to an improvement in the efficiency compared to the existing approach. The algorithm successfully accomplished varied rigorous numeric tasks, and the obtained bounds were of the same order as in the existing approach, which indicates that the algorithm is suitable for the computer assited proofs.

The conclusion is that the algorithm is a proper approach to the task of the rigorous integration of higher dimensional dPDEs, including the Navier-Stokes equations.

We consider the following problem

$$\begin{aligned} u\colon [0,T)\times\mathbb{T}\to\mathbb{R}, \quad f,u_0\colon \mathbb{T}\to\mathbb{R},\\ u_t = L(u) + N(u,u_x,u_{xx},\dots) + f,\\ u(0,x) = u_0(x). \end{aligned} \tag{1.1}$$

Where L is a "Laplacian" operator (for instance $L(u)=\nu u_{xx}$ or $L(u)=-\nu u_{xxxx}-u_{xx}$), N is a proper polynomial of u and its partial derivatives, f is a constant in time forcing, T is a maximal time of existence, $\mathbb{T}$ is the one-dimensional torus.

Not any equation (1.1) is a dPDE. It is additionally required from (1.1) to be called a dPDE that the linear part L "dominates" in some sense the nonlinear part N, for the formal definition refer Section 2.5.

The Fourier basis is introduced and the problem of solving (1.1) is reduced to the problem of solving the following infinite dimensional dynamical system

$$\frac{da_k}{dt} = c_N k^r \sum_{\substack{k_1,\dots,k_n\in\mathbb{Z}\\ k_1+\cdots+k_n=k}} a_{k_1}\cdot a_{k_2}\cdots a_{k_n} + \lambda_k a_k + f_k, \quad k\in\mathbb{Z}. \tag{1.2}$$

Where λ_k are eigenvalues of the "Laplacian", c_N, n and r depends on N, $\{a_k\}_{k\in\mathbb{Z}} \subset \mathbb{C}$ describes the evolution of the Fourier modes. Let us mention a few important equations which fall into dPDE class: the Burgers equation, the Kuramoto-Sivashinsky equation, the Ginzburg-Landau equation, the Swift-Hohenberg equation and, last but not least, the Navier-Stokes equation.

Up to our knowledge, there exists two published theoretical approaches to the rigorous numeric for the **non stationary PDE** problem. The method of self-consistent bounds, presented in the series of papers [ZM], [Z2], [Z3], [ZAKS] and the method presented in [AK]. All mentioned methods have been successfully applied to the study of the Kuramoto-Sivashinsky equation dynamics (KS equation), indeed existence of some time-periodic solutions have been proven. The focus of the previous research on one particular equation comes from the interest in establishing the fist proof of existence of the chaotic dynamics in a partial differential equation.

Presented in Section 6 rigorous numeric tests consider concrete examples of (1.1), and aim at verifying if the presented algorithms can be used to establish new results, which have not been published in the existing literature. We also do a comparison of the presented algorithms with the existing approaches.

1.2 Fast Fourier Transform algorithm

The Fast Fourier Transform (FFT) algorithm is widely known and has proven to be extremely useful in many applications. Fields of applicability are for instance numerical solving of partial differential equations for weather prediction, signal analysis and image processing in computer graphics. Up to our knowledge the FFT algorithm was described for the first time in the paper [CT]. For an excellent review we refer to the papers [T] and [T2].

Definition 1.2.1. *To avoid ambiguities the symbol*

$$i$$

is used only to denote the imaginary unit.

As the orthogonal basis of the periodic functions we pick the trigonometric *Fourier basis functions* denoted by

$$e_k(x) := \exp(ikx), \quad k\in\mathbb{Z}.$$

We will use either of the symbols depending on the context.

Because in this paper we consider only one-dimensional PDEs, we present an one-dimensional version of the FFT algorithm in this section.

1.2.1 Outline of approach

When using a spectral method based on the Fourier series one often has to compute convolutions of a finite number of a finite series $\{u_k\}_{k=-N}^{N}$

$$(u * \cdots * u)_k = \sum_{\substack{k_1+k_2+k_3+\cdots+k_n=k \\ -N\le k_i\le N}} u_{k_1} u_{k_2} \cdots u_{k_n}, \qquad k = -N, \ldots, N. \tag{1.3}$$

Where n is the number of terms in the convolution $u * \cdots * u$. The origin of $(u * \cdots * u)_k$ is always the presence of the term u^n in the differential equation under consideration.

It is known in the numerical analysis community [CHQZ] that computation of (1.3) could be considerably speed-up by the use of the FFT algorithm.

The idea of the approach is based on the following facts

- the function $u(x) = \sum_{k=-N}^{N} u_k \exp(ikx)$ is in the unique way determined by $u(2\pi j/M)$, where $M \ge 2N+1$ and $j = 0, 1, \ldots, M-1$.
- Passing from the values at the nodes $u(2\pi j/M)$, $j = 0, \ldots, M-1$ to the Fourier coefficients u_k, $k = -N, \ldots, N$ and vice-versa is done and is done efficiently by using the FFT algorithm.

Therefore a possibly effective approach to computation of (1.3) is the following

0. assume that $u(x) = \sum_{j=-N}^{N} u_j \exp(ijx)$

1. choose M big enough, see the discussion of *the aliasing problem* in Section 1.3.1.

2. Let $x_j := \frac{2\pi j}{M}$, $j = 0, 1, \ldots, M-1$. Compute $\hat{u}_j = u(x_j)$ - using FFT.

3. Compute $\hat{q}_j = \hat{u}_j \cdots \hat{u}_j$ for $j = 0, 1, \ldots, M-1$. Modulo *the aliasing error*, we have $(u \cdots u)(x_j) = \hat{q}_j$

4. Using FFT applied to $\hat{q}_j$ we obtain the Fourier coefficients for u^n given by (1.3).

1.2.2 Some elementary facts about the trigonometric interpolation on uniform mesh

Let us fix $M \in \mathbb{Z}_+$.

Definition 1.2.2. *Let $\{x_j\}_{j=0}^{M-1}$ be the uniform grid of points in the interval $[0, 2\pi]$ such that $x_j = \frac{2\pi j}{M}$.*

Definition 1.2.3. *Let $u, v : \mathbb{R} \to \mathbb{C}$ be 2π-periodic functions. We define* a discrete L_2 scalar product *by*

$$(u, v)_D := \frac{1}{M} \sum_{j=0}^{M-1} u(x_j) \cdot \overline{v(x_j)}$$

The basis functions $\{e_k\}$ are orthogonal in the sense of the scalar product given by Definition 1.2.3. Namely we have the following lemma

Lemma 1.2.4. *For any $k, l \in \mathbb{Z}$ holds* $(e_k, e_l) = \begin{cases} 1 & k = l + sM \text{ for any } s \in \mathbb{Z} \\ 0 & \text{otherwise.} \end{cases}$,

Proof:

$$(e_k, e_l)_D = \frac{1}{M} \sum_{j=0}^{M-1} \exp(ijk2\pi/M) \exp(-ijl2\pi/M) =$$

$$\frac{1}{M} \sum_{j=0}^{M-1} \exp(ij(k-l)2\pi/M) = \begin{cases} 1, & k = l + sM, \quad s \in \mathbb{Z} \\ 0, & \text{otherwise} \end{cases} .$$

∎

The orthogonality property of e_k with respect the discrete scalar product $(\cdot, \cdot)_D$ allows us to compute the coefficients from the values of $u(x) = \sum_k u_k e_k(x)$ on the set $\{x_j\}$. Namely,

Lemma 1.2.5. *Assume that $u(x) = \sum_{k=-N}^{N} u_k e_k(x)$ and $v(x) = \sum_{k=0}^{M} v_k e_k(x)$. Let $\hat{u}_j = u(x_j)$ and $\hat{v}_j = v(x_j)$ for $j = 0, \ldots, M-1$.*

Then for any $k \in \mathbb{Z}$ holds

$$\sum_{\substack{s \in \mathbb{Z} \\ -N \le k+sM \le N}} u_{k+sM} = \frac{1}{M} \sum_{j=0}^{M-1} \hat{u}_j \exp(-i2\pi jk/M), \tag{1.4}$$

$$v_k = \frac{1}{M} \sum_{j=0}^{M-1} \hat{v}_j \exp(-i2\pi jk/M) \tag{1.5}$$

Proof: From Lemma 1.2.4 it follows immediately that

$$\sum_{\substack{s \in \mathbb{Z} \\ -N \le k+sM \le N}} u_{k+sM} = (u, e_k)_D = \frac{1}{M} \sum_{j=0}^{M-1} \hat{u}_j \overline{e_k(x_j)} =$$

$$\frac{1}{M} \sum_{j=0}^{M-1} \hat{u}_j \exp(-i2\pi jk/M).$$

The proof of the second equality is analogous. ∎

Remark 1.2.6. In the context of the above lemma we are interested in the situation when (1.4) gives us u_k for $k = -N, \ldots, N$. This guaranteed when $-N \leq k + sM \leq N$ is satisfied only with $s = 0$, this forces that $M \geq 2N + 1$. This has an obvious interpretation, the number of nodes need to retrieve to coefficients of trigonometric functions entering in the expression for u should be greater or equal than the number of functions. When this condition is not satisfied, then we have *the aliasing error*, see the discussion in Section 1.3.1.

1.2.3 Interpolation with phase shift

In the context of removing the aliasing error (see Section 1.3.1) it is useful to extend the fact from previous subsection to a shifted grid.

Let us fix $\Delta \in \mathbb{R}$ and $M \in \mathbb{Z}_+$.

Definition 1.2.7. *Let* $\{x_j^\Delta\}_{j=0}^{M-1}$ *be the uniform grid of points in the interval* $[0, 2\pi]$ *such that* $x_j = \left(\frac{2\pi j}{M} + \Delta\right) \mod 2\pi$.

Definition 1.2.8. *Let* $u, v : \mathbb{R} \to \mathbb{C}$ *be* 2π*-periodic functions. We define a discrete* L_2 *scalar product by*

$$(u, v)_D^\Delta := \frac{1}{M} \sum_{j=0}^{M-1} u(x_j^\Delta) \cdot \overline{v(x_j^\Delta)}$$

The basis functions $\{e_k\}$ are orthogonal in the sense of the scalar product given by Definition 1.2.8. Namely we have the following lemma

Lemma 1.2.9. *For any* $k, l \in \mathbb{Z}$ *holds*

$$(e_k, e_l) = \begin{cases} \exp(isM\Delta) & k = l + sM \text{ for any } s \in \mathbb{Z}\ , \\ 0 & \text{otherwise.} \end{cases} \tag{1.6}$$

Proof:

$$(e_k, e_l)_D^\Delta = \frac{1}{M} \sum_{j=0}^{M-1} \exp((ijk2\pi/M + ik\Delta) \exp(-ijl2\pi/M - ij\Delta) =$$

$$\exp(i(k-l)\Delta) \frac{1}{M} \sum_{j=0}^{M-1} \exp(ij(k-l)2\pi/M) =$$

$$\begin{cases} \exp(i(k-l)\Delta), & k = l + sM, \quad s \in \mathbb{Z} \\ 0, & \text{otherwise} \end{cases} .$$

∎

The orthogonality property of e_k with respect the discrete scalar product $(\cdot, \cdot)_D^\Delta$ allows us to compute the coefficients from the values of $u(x) = \sum_k u_k e_k(x)$ on the set $\{x_j^\Delta\}$. Namely,

Lemma 1.2.10. *Assume that* $u(x) = \sum_{k=-N}^{N} u_k e_k(x)$ *and* $v(x) = \sum_{k=0}^{M} v_k e_k(x)$. *Let* $\hat{u}_j^{\Delta} = u(x_j^{\Delta})$ *and* $\hat{v}_j^{\Delta} = v(x_j^{\Delta})$ *for* $j = 0, \ldots, M-1$.

Then for any $k \in \mathbb{Z}$ *holds*

$$\sum_{\substack{s\in\mathbb{Z} \\ -N\le k+sM\le N}} \exp(isM\Delta)u_{k+sM} = \frac{1}{M}\sum_{j=0}^{M-1} \hat{u}_j^{\Delta}\exp(-ik(2\pi j/M+\Delta)), \quad (1.7)$$

$$v_k = \frac{1}{M}\sum_{j=0}^{M-1} \hat{v}_j^{\Delta}\exp(-ik(2\pi j/M+\Delta)) \quad (1.8)$$

Proof: From Lemma 1.2.9 it follows immediately that

$$\sum_{\substack{s\in\mathbb{Z} \\ -N\le k+sM\le N}} \exp(isM\Delta)u_{k+sM} = (u,e_k)_D^{\Delta} = \frac{1}{M}\sum_{j=0}^{M-1} \hat{u}_j^{\Delta}\overline{e_k(x_j^{\Delta})} =$$

$$\frac{1}{M}\sum_{j=0}^{M-1} \hat{u}_j^{\Delta}\exp(-ik(2\pi k/M+\Delta)).$$

The proof of the second equality is analogous. ∎

Remark 1.2.11. Comparing the formulas under the sum over s obtained in Lemma 1.2.5 and in Lemma 1.2.10 is the appearance of the term $\exp(isM\Delta)$ in the latter case. The aliasing error may be eliminated by choosing a proper phase shift Δ.

1.2.4 Uniform matrix form of the FFT

Definition 1.2.12. *Let* $v : \mathbb{R} \to \mathbb{C}$ *be a* 2π*-periodic function such that* v *belongs to the subspace spanned by the functions* $\{e_{-N}, \ldots, e_N\}$, *i.e.* $v(x) = \sum_{k=-N}^{N} v_k e_k(x)$ *then we call*

$$\{v_k\}_{k=-N}^{N}$$

the $l_2(N)$ *coefficients of* v. *When it is clear from context we will drop the notation of* N, *and use simply the* l_2 *coefficients of* v.

Definition 1.2.13. *Let* $v\colon \mathbb{R} \to \mathbb{C}$ *be a* 2π*-periodic function such that* v *belongs to the subspace spanned by the functions* $\{e_{-N}, \ldots, e_N\}$, *i.e.* $v(x) = \sum_{k=-N}^{N} v_k e_k(x)$ *then we call*

$$\hat{v}_j = \sum_{k=-N}^{N} v_k e_k(x_j), \quad j = 0, \ldots, M-1$$

the $L_2(M)$ *coefficients of* v. *When it is clear from context we will drop the notation of* M, *and use simply the* L_2 *coefficients of* v.

By the $l_2(N) \to L_2(M)$ transform (later on called the $l_2 \to L_2$ transform) we mean simply the evaluation of $u(x_j)$, being in fact the Discrete Fourier Transform

$$\hat{u}_j = \sum_{k=-N}^{N} u_k \exp(ijk2\pi/M), \quad j = 0, \ldots, M-1. \tag{1.9}$$

By the $L_2(M) \to l_2(N)$ transform (later on called the $L_2 \to l_2$ transform) we mean the computation of the Fourier coefficients using (1.4), which since $M \geq 2N+1$ gives

$$u_k = \frac{1}{M} \sum_{j=0}^{M-1} \hat{u}_j \exp(-ijk2\pi/M), \quad k = -N, \ldots, N. \tag{1.10}$$

First step in explaining the FFT algorithm is to "unify" (1.9) and (1.10) in a matrix form. We would like to rewrite both transforms defined above in the matrix form. Let us fix N and M. We map the l_2 coefficients $\{u_k\}_{k=-N}^{N}$ and the L_2 coefficients $\{\hat{u}_j\}_{j=0}^{M-1}$ to a M dimensional vectors z and $\hat{z}$ in the following way

$$\begin{aligned} z_k &:= \begin{cases} u_k, & k = 0, \ldots, N, \\ u_{k-M}, & k = M-N, \ldots, M-1 \\ 0, & \text{otherwise} \end{cases}, \\ \hat{z}_j &:= \hat{u}_j, \quad j = 0, \ldots, M-1. \end{aligned} \tag{1.11}$$

Then the matrix form is given by the following Lemma

Lemma 1.2.14 (the FFT matrix form)**.** *Let* $M \geq 2N+1$

$$\hat{z} = W_M z, \quad z = \overline{W_M}\hat{z}, \tag{1.12}$$

where $W_M(j,k) := \exp(ijk2\pi/M)$ *and* $\overline{W_M}(j,k) := \frac{1}{M}\exp(-ijk2\pi/M)$.

Proof: From (1.9) and the definition of z_j, $\hat{z}_j$ it follows that

$$\begin{aligned} \hat{z}_j = \sum_{k=-N}^{N} u_k \exp(ijk2\pi/M) &= \sum_{k=0}^{N} u_k \exp(ijk2\pi/M) + \\ \sum_{k=-N}^{-1} u_k \exp(ijk2\pi/M) &= \sum_{k=0}^{N} z_k \exp(ijk2\pi/M) + \\ \sum_{k=M-N}^{M-1} u_{k-M} \exp(ij(k-M)2\pi/M) &= \sum_{k=0}^{N} z_k \exp(ijk2\pi/M) + \\ \sum_{k=M-N}^{M-1} z_k \exp(ijk2\pi/M) &= \sum_{k=0}^{M-1} z_k \exp(ijk2\pi/M). \end{aligned}$$

For the other equality we proceed as follows, from (1.10) and the definition of z_j, $\hat{z}_j$ we have

$$u_k = \frac{1}{M} \sum_{j=0}^{M-1} \hat{u}_j \exp(-ijk2\pi/M), \qquad k = -N, \ldots, N.$$

Consider first $k = 0, \ldots, N$. Then we have

$$z_k = u_k = \frac{1}{M} \sum_{j=0}^{M-1} \hat{z}_j \exp(-ijk2\pi/M).$$

For $k = M - N, \ldots, M - 1$ we have

$$z_k = u_{k-M} = \frac{1}{M} \sum_{j=0}^{M-1} \hat{u}_j \exp(-ij(k-M)2\pi/M) = \frac{1}{M} \sum_{j=0}^{M-1} \hat{z}_j \exp(-ijk2\pi/M).$$

It remains to show that $0 = z_k = (\overline{W_M}\hat{z})_k$ for $k = N+1, \ldots, M-N-1$.

Let us fix $k = N+1, \ldots, M-N-1$. We have from Lemma 1.2.4 and fact that u is spanned by $\{e_{-N}, \ldots, e_N\}$

$$(\overline{W_M}\hat{z})_k = \frac{1}{M} \sum_{j=0}^{M-1} \exp(-ijk2\pi/M)\hat{z}_j = \frac{1}{M} \sum_{j=0}^{M-1} \exp(-ijk2\pi/M)\hat{u}_j = (u, e_k)_D = 0.$$

∎

As the consequence of the orthogonality relations from Lemma 1.2.4 holds

$$W_M^{-1} = \overline{W_M}.$$

1.2.5 Factorization into sparse matrices

To evaluate (1.12) efficiently one does not perform the multiplication by W_M, which is a dense matrix but one factorizes W_M into *sparse matrices* and perform the multiplication by sparse matrices.

Definition 1.2.15. *By δ_{jk} we will denote* the Kronecker delta, *i.e.*

$$\delta_{jk} = \begin{cases} 1, & j = k \\ 0, & j \neq k \end{cases}.$$

All matrices that will appear from now on are square matrices.

Definition 1.2.16. *Let $A \in \mathbb{C}^{M \times M}$. The element at j-th row and k-th column of A is denoted by*

$$A(j,k) \text{ and } A_{jk},$$

where $j, k = 0, \ldots, M-1$.

Definition 1.2.17. *Let $A \in \mathbb{C}^{n\times n}$ and $B \in \mathbb{C}^{m\times m}$. By $A\times B \in \mathbb{C}^{(n+m)\times(n+m)}$ we denote* the Kronecker matrix product

$$A \times B = \begin{bmatrix} a_{00}B & a_{01}B & \dots & a_{0(n-1)}B \\ a_{10}B & a_{11}B & \dots & a_{1(n-1)}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{(n-1)0}B & a_{(n-1)1}B & \dots & a_{(n-1)(n-1)}B \end{bmatrix}$$

More precisely, we have

$$\begin{aligned}(A\times B)_{k_1m+k_2,j_1m+j_2} &= a_{k_1j_1}b_{k_2j_2}, \quad k_1,j_1 = 0,\dots,n-1,\\ &\qquad k_2,j_2 = 0,\dots,m-1.\end{aligned}$$

Observe that $A\times B$ is in fact a matrix of $A\bigotimes B$ expressed in the basis $\{h_j\}_{j=0,\dots,(n-1)(m-1)} = \{e_{p(j)}\bigotimes f_{q(j)}\}$, where $j = p(j)m+q(j)$, $q(j)\in\{0,\dots,m-1\}$, $p(j)\in\{0,\dots,n-1\}$

Lemma 1.2.18. *The Kronecker product satisfies the following rules*

$$A\times(B\times C) = (A\times B)\times C, \tag{1.13}$$

$$A\times(BC) = (A\times B)(A\times C), \tag{1.14}$$

$$(AB)\times C = (A\times C)(B\times C). \tag{1.15}$$

This properties are obvious consequences of the properties of the tensor product.

Definition 1.2.19. *For $p,q\in\mathbb{Z}_+$ we denote by $D_q^p \in \mathbb{C}^{pq\times pq}$ the following diagonal matrix*

$$D_q^p(j,k) := \begin{cases} \exp(ism2\pi/(pq)), & j=k=sq+m, \quad m=0,\dots,q-1,\ s=0,\dots,p-1\\ 0, & \textit{otherwise.}\end{cases}$$

Hence

$$D_q^p(j_1q+j_2,k_1q+k_2) = \delta_{j_1k_1}\delta_{j_2k_2}\exp(ij_1j_2 2\pi/(pq)),$$

where $j_2,k_2 = 0,\dots,q-1$ and $j_1,k_1 = 0,\dots,p-1$.

Definition 1.2.20. *For $p,q\in\mathbb{Z}_+$ we denote by $P_q^p \in \mathbb{C}^{pq\times pq}$ the following* permutation matrix

$$P_q^p(j,k) := \begin{cases} 1, & \textit{if } j = rp+s \textit{ and } k = sq+r,\\ 0, & \textit{otherwise.}\end{cases}$$

Hence

$$P_p^q(j_1p+j_2,k_1q+k_2) = \delta_{j_1k_2}\delta_{j_2k_1},$$

where $j_1,k_2 = 0,\dots,q-1$ and $j_2,k_1 = 0,\dots,p-1$.

Observe that

$$D_1^p = I_p, \quad P_1^p = I_p, \quad (P_q^p)^{-1} = (P_q^p)^T = P_p^q.$$

Lemma 1.2.21.

$$P_b^a D_b^a = D_a^b P_b^a$$

Proof:

$$D_b^a(j_1b+j_2,k_1b+k_2)=\delta_{j_1k_1}\delta_{j_2k_2}\exp(ij_1j_2 2\pi/(ab)),$$
$$P_b^aD_b^a(j_2a+j_1,k_1b+k_2)=\delta_{j_1k_1}\delta_{j_2k_2}\exp(ij_1j_2 2\pi/(ab)),$$

and

$$D_a^b(j_2a+j_1,k_2a+k_1)=\delta_{j_1k_1}\delta_{j_2k_2}\exp(ij_1j_2 2\pi/(ab)),$$
$$D_a^bP_b^a(j_2a+j_1,k_1b+k_2)=\delta_{j_1k_1}\delta_{j_2k_2}\exp(ij_1j_2 2\pi/(ab)).$$

∎

The following *matrix factorization Theorem* lies at heart of the FFT algorithm

Lemma 1.2.22. *Let $W_d \in \mathbb{C}^{d\times d}$ such that $W_d(j,k)=\exp(ijk2\pi/d)$, I be the identity matrix, D and P be the matrices given by Definition 1.2.19 and Definition 1.2.20, $\times$ be the Kronecker matrix product, then*

$$W_{pq}=(W_q\times I_p)\,P_q^pD_q^p\,(W_p\times I_q)$$

Proof: The proof was provided in [T], hereafter we provide another variant.

We have

$$(W_p\times I_q)_{kj}=(W_p)_{k_1j_1}\,\delta_{k_2j_2}=\omega^{qk_1j_1}\delta_{k_2j_2},\quad k=k_1q+k_2,\ j=j_1q+j_2. \quad (1.16)$$

The matrix D_q^p is diagonal, with $(D_q^p)_{kk}=\exp(ik_1k_2 2\pi/pq)=\omega^{k_1k_2}$, where k,k_1,k_2 are as above, and $\omega=\exp i2\pi/pq$. Hence the effect of the multiplication by D_q^p on the left, is the multiplication of k-th row by $\omega^{k_1k_2}$, hence

$$\left(D_q^p\,(W_p\times I_q)\right)_{kj}=\omega^{k_1k_2+qk_1j_1}\delta_{k_2j_2},\quad k=k_1q+k_2,\ j=j_1q+j_2. \quad (1.17)$$

The multiplication on the left by permutation matrix P_q^p exchanges rows. In particular, the k-th row, where $k=k_1q+k_2$ becomes now (k_2p+k_1)-th row. Hence we obtain

$$\left(P_q^pD_q^p\,(W_p\times I_q)\right)_{kj}=\omega^{k_1k_2+qk_1j_1}\delta_{k_2j_2},\quad k=k_2p+k_1,\ j=j_1q+j_2. \quad (1.18)$$

For the final multiplication by $W_q\times I_p$, let us observe that (notice different meaning of l_i and k_i below)

$$(W_q\times I_p)_{lk}=\omega^{pl_1k_2}\delta_{l_2k_1},\quad l=l_1p+l_2,\ k=k_2p+k_1. \quad (1.19)$$

By combining (1.18) and (1.19) we obtain (we use decompositions $l=l_1p+l_2$, $k=k_2p+k_1$, $j=j_1q+j_2$)

$$\sum_k (W_q\times I_p)_{lk}\left(P_q^pD_q^p\,(W_p\times I_q)\right)_{kj}=\sum_{k_1,k_2}\omega^{pl_1k_2}\delta_{l_2k_1}\omega^{k_1k_2+qk_1j_1}\delta_{k_2j_2}=$$
$$\omega^{pl_1j_2}\omega^{l_2j_2+ql_2j_1}=\omega^{(l_1p+l_2)(j_1q+j_2)-l_1j_1pq}=\omega^{lj}.$$

∎

We follow [T] to extend Lemma 1.2.22 to an arbitrary number of factors. First, to extract information how the general pattern looks like we use a three factor case

$$W_{pqr} = W_{p(qr)} = (W_{qr} \times I_p)\, P^p_{qr} D^p_{qr}\, (W_p \times I_{qr}) = \\ (W_r \times I_{pq})\, (P^q_r D^q_r \times I_p)\, (W_q \times I_{pr})\, P^p_{qr} D^p_{qr}\, (W_p \times I_{qr}),$$

we rewrite it in the following form

$$W_{pqr} = T_3 T_2 T_1,$$

where

$$T_1 = P^p_{qr} D^p_{qr}\, (W_p \times I_{qr}),$$
$$T_2 = (P^q_r D^q_r \times I_p)\, (W_q \times I_{pr}),$$
$$T_3 = (P^r_1 D^r_1 \times I_{pq})\, (W_r \times I_{pq}),$$

observe that $P^r_1 D^r_1 \times I_{pq} = I_{pqr}$.

The FFT algorithm is derived as a consequence of the following Theorem, providing explicit form of sparse matrices to compute the transformations given by (1.12).

Theorem 1.2.23 (Algorithm FFT-A [T]). *Let $W_d \in \mathbb{C}^{d\times d}$ such that $W_d(j,k) = \exp(ijk2\pi/d)$, I be the identity matrix, D and P be the matrices given by Definition 1.2.19 and Definition 1.2.20, $\times$ be the Kronecker matrix product. Assume that M is factorized in the following way $M = n_k n_{k-1} \dots n_1$, then*

$$W_M = T_k T_{k-1} \cdots T_1,$$
$$T_j = \left(P^{n_j}_{m_j} D^{n_j}_{m_j} \times I_{l_j}\right) \left(W_{n_j} \times I_{M/n_j}\right),$$

where $l_1 = 1$, $l_{j+1} = n_j l_j$ and $m_j = M/l_{j+1}$, $j = 1, \dots, k$.

Proof We proceed by induction. First, the two factor case is fulfilled by Lemma 1.2.22. Second, assuming that $W_{M/n_1} = \widetilde{T}_k \cdots \widetilde{T}_2$ we verify that $W_M = T_k T_{k-1} \cdots T_1$. Observe that the matrices T_i and $\widetilde{T}_i$ differ in dimensions.

It is easy to see that

$$l_1 = 1, \quad l_{j+1} = n_1 n_2 \cdots n_j, \quad m_j = n_{j+1} \cdots n_k.$$

$$W_M = W_{n_1(M/n_1)} = (W_{M/n_1} \times I_{n_1})(P^{n_1}_{M/n_1} D^{n_1}_{M/n_1})(W_{n_1} \times I_{M/n_1}) = \\ (W_{M/n_1} \times I_{n_1}) T_1$$

because

$$P^{n_1}_{M/n_1} D^{n_1}_{M/n_1} = (P^{n_1}_{M/n_1} D^{n_1}_{M/n_1}) \times I_1 = (P^{n_1}_{M/n_1} D^{n_1}_{M/n_1}) \times I_{l_1}.$$

Now we need to transform $W_{M/n_1} \times I_{n_1}$ to a desired form.

By induction hypothesis we know that $W_{M/n_1} = \widetilde{T}_k \cdots \widetilde{T}_2$, where

$$\widetilde{T}_j = \left(P^{n_j}_{\widetilde{m}_j} D^{n_j}_{\widetilde{m}_j} \times I_{\widetilde{l}_j}\right) \left(W_{n_j} \times I_{M/n_1 n_j}\right), \tag{1.20}$$

where

$$\widetilde{m}_j = M n_1 / l_{j+1} n_1 = M / l_{j+1} = m_j, \quad \widetilde{l}_j = l_j / n_1.$$

Therefore

$$\widetilde{T}_j \times I_{n_1} = \left(P^{n_j}_{\widetilde{m}_j} D^{n_j}_{\widetilde{m}_j} \times I_{\widetilde{l}_j n_1}\right) \left(W_{n_j} \times I_{(Mn_1)/n_1 n_j}\right) = T_j, \quad j = 2, \ldots, k. \tag{1.21}$$

This completes the proof. ∎

Observe that the factorization of M in Theorem 1.2.23 is arbitrary, in particular n_j's may not be prime, and appear in any order. W_{n_j} are often referred as *"small transforms"*, and it depends only on implementation what "small transforms" are provided. Convenient and efficient implementation uses a finite number of "small transforms" and therefore requires the input number M to be a product only of n_j's defining the small transforms.

```
Input: Vector z, Vector ω
l_j := 1;
/* Loop for each factor.                                          */
for j := 1, ..., k do
    m_j := M/l_j n_j;
    /* Two loops of LINEAR complexity (m_j · l_j = M/n_j).        */
    for m := 0, ..., m_j − 1 do
        for l := 0, ..., l_j − 1 do
            t := l + m · l_j;
            /* Here we multiply sub-vector composed by elements
               indexed by t, M/n_j + t, ..., (n_j − 1) · M/n_j + t of
               vector z by the "small transform" matrix W_{n_j}.   */
            z[t, M/n_j + t, ..., (n_j − 1) · M/n_j + t] :=
            W_{n_j} · z[t, M/n_j + t, ..., (n_j − 1) · M/n_j + t];
            /* Multiplication by phase factors ω and the
               permutation step                                    */
            for s := 0, ..., n_j − 1 do
                c[s · M/n_j + t] := ω[m · l_j · s] · z[l + n_j · m · l_j + s · l_j];
            end
        end
    end
    z := c;
    l_j := n_j l_j;
end
```

Algorithm 1: Algorithm FFT-A

To explain intuitively Theorem 1.2.23 we present some examples.

Example 1.2.24. Let $M = pq$, $W_p(j,k) = \omega^{jkq}$, $W_q = \omega^{jkp}$, $\omega := \exp(i2\pi/M)$

$$W_M = (W_q \times I_p)\, P_q^p D_q^p\, (W_p \times I_q)$$

$$W_q \times I_p = \left[\begin{array}{ccc|ccc|c|ccc}
\overbrace{1 & 0 & 0}^{p \text{ elements}} & & 1\ 0\ 0 & & & & 1\ 0\ 0 & \\
 & \ddots & & & \ddots & & \cdots & & \ddots & \\
0 & \ddots & 0 & & 0\ \ddots\ 0 & & & & 0\ \ddots\ 0 & \\
0 & 0 & 1 & & 0\ 0\ 1 & & & & 0\ 0\ 1 & \\
\hline
1 & 0 & 0 & \omega^{1\cdot 1\cdot p} & 0 & 0 & & \omega^{1\cdot(q-1)\cdot p} & 0 & 0 \\
 & \ddots & & & \ddots & & \cdots & & \ddots & \\
0 & \ddots & 0 & 0 & \ddots\ 0 & & & 0 & \ddots\ 0 & \\
0 & 0 & 1 & 0 & 0\ \omega^{1\cdot 1\cdot p} & & & 0 & 0\ \omega^{1\cdot(q-1)\cdot p} & \\
\hline
 & \vdots & & & \vdots & & \ddots & & \vdots & \\
\hline
1 & 0 & 0 & \omega^{(q-1)\cdot 1\cdot p} & 0 & 0 & & \omega^{(q-1)\cdot(q-1)\cdot p} & 0 & 0 \\
 & \ddots & & & \ddots & & \cdots & & \ddots & \\
0 & \ddots & 0 & 0 & \ddots\ 0 & & & 0 & \ddots\ 0 & \\
0 & 0 & 1 & 0 & 0\ \omega^{(q-1)\cdot 1\cdot p} & & & 0 & 0\ \omega^{(q-1)\cdot(q-1)\cdot p} &
\end{array}\right],$$

$$W_p \times I_q = \left[\begin{array}{ccc|ccc|c|ccc}
\overbrace{1 & 0 & 0}^{q \text{ elements}} & & 1\ 0\ 0 & & & & 1\ 0\ 0 & \\
 & \ddots & & & \ddots & & \cdots & & \ddots & \\
0 & \ddots & 0 & & 0\ \ddots\ 0 & & & & 0\ \ddots\ 0 & \\
0 & 0 & 1 & & 0\ 0\ 1 & & & & 0\ 0\ 1 & \\
\hline
1 & 0 & 0 & \omega^{1\cdot 1\cdot q} & 0 & 0 & & \omega^{1\cdot(p-1)\cdot q} & 0 & 0 \\
 & \ddots & & & \ddots & & \cdots & & \ddots & \\
0 & \ddots & 0 & 0 & \ddots\ 0 & & & 0 & \ddots\ 0 & \\
0 & 0 & 1 & 0 & 0\ \omega^{1\cdot 1\cdot q} & & & 0 & 0\ \omega^{1\cdot(p-1)\cdot q} & \\
\hline
 & \vdots & & & \vdots & & \ddots & & \vdots & \\
\hline
1 & 0 & 0 & \omega^{(p-1)\cdot 1\cdot q} & 0 & 0 & & \omega^{(p-1)\cdot(p-1)\cdot q} & 0 & 0 \\
 & \ddots & & & \ddots & & \cdots & & \ddots & \\
0 & \ddots & 0 & 0 & \ddots\ 0 & & & 0 & \ddots\ 0 & \\
0 & 0 & 1 & 0 & 0\ \omega^{(p-1)\cdot 1\cdot q} & & & 0 & 0\ \omega^{(p-1)\cdot(p-1)\cdot q} &
\end{array}\right].$$

Example 1.2.25. Let $M = 2^p$ then $W_2 = \left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$, $\omega := \exp(i2\pi/M)$

$$W_M = T_p T_{p-1} \dots T_1,$$
$$T_j = \left(P^2_{m_j} D^2_{m_j} \times I_{l_j}\right) \left(W_2 \times I_{M/2}\right),$$

$$W_2 \times I_{M/2} = \left[\begin{array}{ccc|ccc} \overbrace{1 & 0 & 0}^{M/2 \text{ elements}} & & & 1 & 0 & 0 \\ 0 & \ddots & 0 & & & 0 & \ddots & 0 \\ 0 & 0 & 1 & & & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & & & -1 & 0 & 0 \\ 0 & \ddots & 0 & & & 0 & \ddots & 0 \\ 0 & 0 & 1 & & & 0 & 0 & -1 \end{array}\right],$$

$$D^2_{m_j} \times I_{l_j} =$$

$$\left[\begin{array}{cccc|ccc|c|ccc} \overbrace{1 & 0 & 0 & 0}^{(m_j+1)\cdot l_j \text{ elements}} & & & & & & & & & & \\ 0 & \ddots & 0 & 0 & & & & & & & & & & \\ 0 & 0 & \ddots & 0 & & & 0 & & & 0 & & \cdots & & 0 & \\ 0 & 0 & 0 & 1 & & & & & & & & & & & \\ \hline & & & & \omega^1 & 0 & 0 & & & & & & & & \\ & 0 & & & 0 & \ddots & 0 & & & 0 & & \cdots & & 0 & \\ & & & & 0 & 0 & \omega^1 & & & & & & & & \\ \hline & & & & & & & \omega^2 & 0 & 0 & & & & & \\ & 0 & & & & 0 & & 0 & \ddots & 0 & \cdots & & & 0 & \\ & & & & & & & 0 & 0 & \omega^2 & & & & & \\ \hline & 0 & & & & 0 & & & 0 & & \ddots & & & 0 & \\ \hline & & & & & & & & & & & & \omega^{m_j-1} & 0 & 0 \\ & 0 & & & & 0 & & & 0 & & \cdots & & 0 & \ddots & 0 \\ & & & & & & & & & & & & 0 & 0 & \omega^{m_j-1} \end{array}\right].$$

Example 1.2.26. Let $M = 3^p$ then $W_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -0.5 - i\sqrt{3}/2 & -0.5 + i\sqrt{3}/2 \\ 1 & -0.5 + i\sqrt{3}/2 & -0.5 - i\sqrt{3}/2 \end{bmatrix}$

$$W_M = T_p T_{p-1} \dots T_1,$$
$$T_j = \left(P^3_{m_j} D^3_{m_j} \times I_{l_j} \right) \left(W_3 \times I_{M/3} \right),$$

$$W_3 \times I_{M/3} = \left[\begin{array}{ccc|ccc|ccc} \overbrace{1 & 0 & 0}^{M/3 \text{ elements}} & & & 1 & 0 & 0 & & & 1 & 0 & 0 \\ 0 & \ddots & 0 & & & 0 & \ddots & 0 & & & 0 & \ddots & 0 \\ 0 & 0 & 1 & & & 0 & 0 & 1 & & & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & -0.5 - i\sqrt{3}/2 & 0 & 0 & -0.5 + i\sqrt{3}/2 & 0 & 0 \\ 0 & \ddots & 0 & 0 & \ddots & 0 & 0 & \ddots & 0 \\ 0 & 0 & 1 & 0 & 0 & -0.5 - i\sqrt{3}/2 & 0 & 0 & -0.5 + i\sqrt{3}/2 \\ \hline 1 & 0 & 0 & -0.5 + i\sqrt{3}/2 & 0 & 0 & -0.5 - i\sqrt{3}/2 & 0 & 0 \\ 0 & \ddots & 0 & 0 & \ddots & 0 & 0 & \ddots & 0 \\ 0 & 0 & 1 & 0 & 0 & -0.5 + i\sqrt{3}/2 & 0 & 0 & -0.5 - i\sqrt{3}/2 \end{array}\right].$$

1.2.6 Variants of the FFT algorithm

As it was observed in [T] Algorithm FFT-A given in Theorem 1.2.23 is not the only possible form of the FFT algorithm. In this subsection, we derive two other forms that have been used in the actual program.

Execution of Algorithm FFT-A requires performing the multiplication by the $\left(P^{n_j}_{m_j} D^{n_j}_{m_j} \times I_{l_j}\right)$. Usually this operation is being split into the following product $\left(P^{n_j}_{m_j} \times I_{l_j}\right) \left(D^{n_j}_{m_j} \times I_{l_j}\right)$. We call the multiplication by $P^{n_j}_{m_j} \times I_{l_j}$ is called *the permutation step*. The permutation step is crucial only in some applications, in the problem of the integration of a dPDE the permutation step is completely unnecessary. The reason is as follows when permutation step is not executed during the $l_2 \to L_2$ transform then the L_2 coefficients are returned in a scrambled order, meaning that given a resulting vector of L_2 coefficients one cannot tell at which position the j-th L_2 coefficient is. Still, when one's interest lies only in multiplying L_2 coefficients and then $L_2 \to l_2$ transforming the multiplication result back, the exact order in which the L_2 coefficients appear does not have to be clear.

To get rid of the permutation matrix one cannot simply forget about this step during execution of Algorithm FFT-A. Another variants of the FFT algorithm, one in which the permutation step is interleaved to the end and one in which the permutation step is first to be executed has to be derived for the mentioned purpose.

Derivation of Algorithm FFT-B

Lemma 1.2.27 ([T]). *Let $W_d \in \mathbb{C}^{d \times d}$ such that $W_d(j,k) = \exp(ijk2\pi/d)$, I be the identity matrix, D and P be the matrices given by Definition 1.2.19 and*

Definition 1.2.20, $\times$ be the Kronecker matrix product. Then

$$(W_p \times I_q) P_p^q = P_p^q (I_q \times W_p). \tag{1.22}$$

and

$$P_q^p (W_p \times I_q) = (I_q \times W_p) P_q^p. \tag{1.23}$$

Proof In [T] Lemma 1.2.27 was stated without any proof, we would like to propose a proof here.

Let us consider first $\left((W_p \times I_q) P_p^q\right)_{lj}$. Observe that in the permutation matrix P_p^q the only nonzero element in the j-th column is equal 1 and appears in the row $k = sq + r$, where r, s are given by the condition

$$j = rp + s. \tag{1.24}$$

Moreover, the right-multiplication (multiplication from the right-hand side) by permutation matrix permutes columns, and the left-multiplication permutes rows. Let $l = aq + b$, where $b = 0, \ldots, q-1$. Then we have

$$\left((W_p \times I_q) P_p^q\right)_{lj} = (W_p \times I_q)_{aq+b,sq+r} = (W_p)_{as} (I_q)_{br}. \tag{1.25}$$

Hence

$$\left((W_p \times I_q) P_p^q\right)_{lj} = \omega^{as} \delta_{br}, \quad \text{where } l = aq + b,\ j = rp + s, \tag{1.26}$$

Now we compute $\left(P_p^q (I_q \times W_p)\right)_{lj}$. Let l, j are expressed by a, b, s, r as above. Then it is easy to see that

$$\left(P_p^q (I_q \times W_p)\right)_{lj} = (I_q \times W_p)_{bp+a,rp+s} = (I_q)_{br} (W_p)_{as} = \omega^{as} \delta_{br}$$

This finishes the proof. ∎

First, in order to derive a general pattern for this variant of the FFT algorithm, basing on the two-factor case

$$W_{qr} = (W_r \times I_q) P_r^q D_r^q (W_q \times I_r),$$

we analyze the three-factor case

$$\begin{gathered} W_{p(qr)} = (W_{qr} \times I_p) P_{qr}^p D_{qr}^p (W_p \times I_{qr}) = P_{qr}^p (I_p \times W_{qr}) D_{qr}^p (W_p \times I_{qr}) = \\ P_{qr}^p \left[(I_p \times W_r \times I_q) (I_p \times P_r^q) (I_p \times D_r^q) (I_p \times W_q \times I_r)\right] D_{qr}^p (W_p \times I_{qr}) = \\ P_{qr}^p (I_p \times P_r^q) (I_{pq} \times W_r \times I_1) (I_p \times D_r^q) (I_p \times W_q \times I_r) D_{qr}^p (I_1 \times W_p \times I_{qr}). \end{gathered}$$

In the above computations we used Lemmas 1.2.27 and 1.2.18 and the following observation

$$\begin{gathered} (I_p \times W_r \times I_q) (I_p \times P_r^q) = I_p \times ((W_r \times I_q) P_r^q) = \\ I_p \times (P_r^q (I_q \times W_r)) = (I_p \times P_r^q) (I_p \times I_q \times W_r) = \\ (I_p \times P_r^q) (I_{pq} \times W_r \times I_1) \end{gathered}$$

The three factor case provides enough information to "guess" the general pattern for this variant of the FFT algorithm.

$$W_{p(qr)} = P^p_{qr}\,(I_p \times P^q_r)\,(I_{pq} \times W_r \times I_1)\,(I_p \times D^q_r)\,(I_p \times W_q \times I_r)\,D^p_{qr}\,(I_1 \times W_p \times I_{qr}) =$$
$$\left(I_1 \times P^p_{qr}\right)(I_p \times P^q_r)\,(I_{pq} \times P^r_1)\cdot$$
$$\cdot\left(I_{pq} \times D^r_{pqr/(pqr)}\right)\left(I_{pq} \times W_r \times I_{pqr/(pqr)}\right)\cdot$$
$$\cdot\left(I_p \times D^q_{pqr/(pq)}\right)\left(I_p \times W_q \times I_{pqr/(pq)}\right)\cdot$$
$$\left(I_1 \times D^p_{pqr/p}\right)\left(I_1 \times W_p \times I_{pqr/p}\right).$$

Observe that the above equality can be rewritten in the following form

$$W_{pqr} = P_1P_2P_3T_3T_2T_1.$$

Theorem 1.2.28 (Algorithm FFT-B)**.** *Assume that M is factorized in the following way $M = n_kn_{k-1}\dots n_1$, then*

$$W_M = P_1P_2\dots P_kT_kT_{k-1}\dots T_1,$$
$$T_j = \left(I_{l_j} \times D^{n_j}_{m_j}\right)\left(I_{l_j} \times W_{n_j} \times I_{m_j}\right),$$
$$P_j = I_{l_j} \times P^{n_j}_{m_j},$$

where $l_1 = 1$, $l_{j+1} = n_jl_j$ and $m_j = M/l_{j+1}$, $j = 1,\dots,k$.

Proof We skip the proof because of the similarity to the proof of Theorem 1.2.23.

We provide Algorithm FFT-B in pseudo-code

Input: *Vector* z, *Vector* ω
$l_j := 1$;
/* Loop for each factor. */
for $j := 1, \ldots, k$ **do**
  $m_j := M/l_j n_j$;
  /* Two loops of LINEAR complexity ($m_j \cdot l_j = M/n_j$). */
  for $l := 0, \ldots, l_j - 1$ **do**
    for $m := 0, \ldots, m_j - 1$ **do**
      $t := m + l \cdot m_j \cdot n_j$;
      /* Here we multiply sub-vector composed by elements
        indexed by $t, m_j + t, \ldots, (n_j - 1) \cdot m_j + t$ of vector z
        by the "small transform" matrix W_{n_j}. */
      $z[t, m_j + t, \ldots, (n_j - 1) \cdot m_j + t] :=$
      $W_{n_j} \cdot z[t, m_j + t, \ldots, (n_j - 1) \cdot m_j + t]$;
      /* Multiplication by phase factors ω. */
      for $s := 0, \ldots, n_j - 1$ **do**
        $z[s \cdot m_j + t] := \omega[l_j \cdot m \cdot s] \cdot z[s \cdot m_j + t]$;
      end
    end
  end
  $l_j := n_j l_j$;
end

Algorithm 2: Multiplication by $T_k T_{k-1} \ldots T_1$ in Algorithm FFT-B

Input: *Vector* z
$l_j := M/n_k$;
for $j := k - 1, \ldots, 1$ **do**
  $m_j := M/l_j n_j$;
  for $l := 0, \ldots, l_j - 1$ **do**
    for $m := 1, \ldots, n_j m_j - 1$ **do**
      $t := \lfloor m/n_j \rfloor$;
      $s := m \mod n_j$;
      exchange$(z[m + l \cdot m_j \cdot n_j], z[m_j \cdot s + t + l \cdot m_j \cdot n_j])$;
    end
  end
  $l_j := l_j n_j$;
end

Algorithm 3: Multiplication by $P_1 P_2 \ldots P_k$ in Algorithm FFT-B

Derivation of Algorithm FFT-C

Here we provide a similar analysis as for the algorithms FFT-A and FFT-B presented earlier, but for the third variant of the FFT algorithm, which we call

the FFT-C algorithm.

$$
\begin{aligned}
W_{(pq)r} = (W_r \times I_{pq})\, P_r^{pq} D_r^{pq}\, (W_{pq} \times I_r) = (W_r \times I_{pq})\, D_{pq}^r\, (I_r \times W_{pq})\, P_r^{pq} = \\
(W_r \times I_{pq})\, D_{pq}^r \left[I_r \times \left((W_q \times I_p)\, P_q^p D_q^p\, (W_p \times I_q) \right) \right] P_r^{pq} = \\
(W_r \times I_{pq})\, D_{pq}^r \left[I_r \times \left((W_q \times I_p)\, D_p^q P_q^p\, (W_p \times I_q) \right) \right] P_r^{pq} = \\
(W_r \times I_{pq})\, D_{pq}^r \left[I_r \times \left((W_q \times I_p)\, D_p^q\, (I_q \times W_p)\, P_q^p \right) \right] P_r^{pq} = \\
(W_r \times I_{pq})\, D_{pq}^r\, (I_r \times W_q \times I_p) \left(I_r \times D_p^q \right) (I_{rq} \times W_p) \left(I_r \times P_q^p \right) P_r^{pq}.
\end{aligned}
$$

The second identity in the first line and the third line above is due to the Lemma 1.2.21, the fourth line is due to Lemma 1.2.27, the fifth line is due to the one of the properties given in Lemma 1.2.18.

Theorem 1.2.29 (Algorithm FFT-C)**.** *Assume that M is factorized in the following way $M = n'_k n'_{k-1} \dots n'_1$, then*

$$
\begin{aligned}
W_M &= T_k T_{k-1} \dots T_1 P'_1 P'_2 \dots P'_k, \\
T_j &= \left(I_{m_j} \times W_{n'_j} \times I_{l_j} \right) \left(I_{m_j} \times D_{l_j}^{n'_j} \right), \\
P'_j &= I_{m_j} \times P_{n'_j}^{l_j},
\end{aligned}
$$

where $l_1 = 1$, $l_{j+1} = n'_j l_j$ and $m_j = M/l_{j+1}$, $j = 1, \dots, k$.

```
Input: Vector z
l_j := M/n_k;
for j := k − 1, ..., 1 do
    m_j := M/l_j n_j;
    for m := 0, ..., m_j − 1 do
        for l := 1, ..., l_j n_j − 1 do
            t := ⌊l/n_j⌋;
            s := l mod n_j;
            exchange(z[l + m · l_j · n_j], z[s · l_j + t + m · l_j · n_j]);
        end
    end
    l_j := l_j n_j;
end
```

Algorithm 4: Multiplication by $P'_1 P'_2 \dots P'_k$ in Algorithm FFT-C

Input: *Vector* z, *Vector* ω
$l_j := 1$;
/* Loop for each factor. */
for $j := 1, \ldots, k$ **do**
  $m_j := M/l_j n_j$;
  /* Two loops of LINEAR complexity ($m_j \cdot l_j = N/n_j$). */
  for $m := 0, \ldots, m_j - 1$ **do**
    for $l := 0, \ldots, l_j - 1$ **do**
      $t := l + m \cdot n_j \cdot l_j$;
      /* First, multiplication by phase factors ω. */
      for $s := 0, \ldots, n_j - 1$ **do**
        $z[s \cdot l_j + t] := \omega[m_j \cdot l \cdot s] \cdot z[s \cdot l_j + t]$;
      end
      /* Here we multiply sub-vector composed by elements indexed by $t, m_j + t, \ldots, (n_j - 1) \cdot m_j + t$ of vector z by the "small transform" matrix W_{n_j}. */
      $z[t, l_j + t, \ldots, (n_j - 1) \cdot l_j + t] := W_{n_j} \cdot z[t, l_j + t, \ldots, (n_j - 1) \cdot l_j + t]$;
    end
  end
  $l_j := n_j l_j$;
end

Algorithm 5: Multiplication by $T_k T_{k-1} \ldots T_1$ in Algorithm FFT-C

Argument that the permutation step can be neglected

We recall that the permutation step is crucial only in some applications, in the problem of a PDE integration the permutation step is completely unnecessary, when the approach presented in Section 1.2.1 is taken. The L_2 coefficients are returned in a scrambled order from the $l_2 \to L_2$ transform, but it does not hurt. Algorithm FFT-B and Algorithm FFT-C executed in a right order permits to neglect the permutation step completely.

Now we argue that in fact, when $l_2 \to L_2$ transform is performed by the Algorithm FFT-C and $L_2 \to l_2$ transform is performed by the Algorithm FFT-B the permutation step can be dropped, because the permutation matrices cancel each other.

Lemma 1.2.30. *Let P_j and P'_j be the permutation matrices from Theorem 1.2.28 and Theorem 1.2.29. Let $M = n_k n_{k-1} \cdots n_1 = n'_k n'_{k-1} \cdots n'_1$. Where $n_j, n'_j \in$ "small transforms" for $j = 1, \ldots, k$.*
When the factors satisfies the following relation $n_j = n'_{k-j+1}$, for all $j = 1, \ldots, k$ (inversed order), then

$$P_j^{-1} = P'_{k-j+1}, \quad k = 1, \ldots, k.$$

Proof

$$P_1 = P_{M/n_1}^{n_1},\ P'_k = P_{n'_k}^{M/n'_k},$$

$$P_2 = I_{n_1} \times P_{M/n_1 n_2}^{n_2},\ P'_{k-1} = I_{n'_k} \times P_{n'_{k-1}}^{M/n'_k n'_{k-1}},$$

$$P_j = I_{l_j} \times P_{m_j}^{n_j} = I_{n_1 n_2 \ldots n_{j-1}} \times P_{n_{j+1} n_{j+2} \ldots n_k}^{n_j}, \quad \text{for } j > 1$$

$$P'_{k-j} = I_{m_{k-j}} \times P_{n'_{k-j}}^{l_{k-j}} = I_{n'_{k-j+1} n'_{k-j+2} \ldots n'_k} \times P_{n'_{k-j}}^{n'_1 n'_2 \ldots n'_{k-j-1}}, \quad \text{for } j > 1.$$

Now we use the fact that $n_j = n'_{k-j+1}$ for all $j = 1, \ldots, k$, then

$$P_j P'_{k-j+1} =$$

$$I_{n_1 n_2 \ldots n_{j-1}} \times P_{n_{j+1} n_{j+2} \ldots n_k}^{n_j} \cdot I_{n'_{k-j+2} n'_{k-j+3} \ldots n'_k} \times P_{n'_{k-j+1}}^{n'_1 n'_2 \ldots n'_{k-j}} = I, \quad j = 2, \ldots, k,$$

$$\text{and } P_1 P'_k = P_{M/n_1}^{n_1} P_{n'_k}^{M/n'_k} = I.$$

∎

1.2.7 Complexity argument

The point about the FFT algorithm that we would like to stress is that the complexity of the FFT algorithms is less than quadratic with respect to the dimension. The explanation here is provided for the algorithm FFT-B, but for the other mentioned variants it is analogous.

Now, we would like to argue, that the complexity of each loop corresponding to a factor of M is linear. First of all, observe that each row of the sparse matrix $\left(W_{n_j} \times I_{m_j}\right)$ has exactly n_j non-zero elements (complex, but not necessarily), compare Example 1.2.24, Example 1.2.25 and Example 1.2.26. The multiplication of a sub-vector of z by the matrix $D_{m_j}^{n_j} \left(W_{n_j} \times I_{m_j}\right)$ is presented in the following pseudo-code

```
for m := 0, ..., m_j − 1 do
    t := m + l · m_j · n_j;
    /* Here we multiply sub-vector composed by elements
       indexed by t, m_j + t, ..., (n_j − 1) · m_j + t of vector z by the
       "small transform" matrix W_{n_j}.                              */
    z[t, m_j + t, ..., (n_j − 1) · m_j + t] := W_{n_j} · z[t, m_j + t, ..., (n_j − 1) · m_j + t];
    /* Multiplication by phase factors ω.                             */
    for s := 0, ..., n_j − 1 do
        z[s · m_j + t] := ω[l_j · m · s] · z[s · m_j + t];
    end
end
```

The code comprises exactly $2 \cdot n_j \cdot m_j$ complex multiplications and $m_j \cdot (n_j - 1)$ complex additions.

But another external loop is needed, which corresponds to the multiplication by I_{l_j}. The multiplication by the matrix $\left(I_{l_j} \times D_{m_j}^{n_j}\right)\left(I_{l_j} \times W_{n_j} \times I_{m_j}\right)$ is presented in the following code

```
for l := 0, ..., l_j − 1 do
    for m := 0, ..., m_j − 1 do
        t := m + l · m_j · n_j;
        /* Here we multiply sub-vector composed by elements
           indexed by t, m_j + t, ..., (n_j − 1) · m_j + t of vector z by
           the "small transform" matrix W_{n_j}.                    */
        z[t, m_j+t, ..., (n_j−1)·m_j+t] := W_{n_j} · z[t, m_j+t, ..., (n_j−1)·m_j+t];
        /* Multiplication by phase factors ω.                       */
        for s := 0, ..., n_j − 1 do
            z[s · m_j + t] := ω[l_j · m · s] · z[s · m_j + t];
        end
    end
end
```

The code comprises exactly $2 \cdot n_j \cdot m_j \cdot l_j$ complex multiplications and $(n_j - 1) \cdot m_j \cdot l_j$ complex additions. Recall that

$$n_j \cdot m_j \cdot l_j = M$$

that shows a linear complexity of each loop corresponding to one of the factors of M.

Loop as above is executed for each factor of M, therefore an another external loop that iterates through all the factors $M = n_1 \cdots n_k$ is needed.

```
for j := 1, ..., k do
end
```

Therefore to obtain the total complexity of the FFT algorithm the number M is multiplied by the number of factors, which is

$$\sum_{n_j \in \text{“small transforms”}} p_j, \text{ where } M = n_1^{p_1} n_2^{p_2} \cdots,$$

what numbers exactly are provided in the set “small transforms” depends on the particular implementation (in ours the set “small transforms” contains the factors $2, 3, 4$ and 5).

The total, disregarding constants complexity of the FFT algorithm is

$$M \sum_{n_j \in \text{“small transforms”}} p_j.$$

Later on in the text to denote the logarithmic factor $\sum p_j$ we will use the simplified symbol

$$\log M.$$

1.3 Issues arising when the FFT algorithm is used

In this section we describe what *the aliasing error* is and provide techniques that allows to eliminate it.

We emphasize that two of the techniques presented in this section are apparently more efficient than the standard one, currently used in the rigorous numerics community, see [GLM] and references given there. Our approach is based on the technique introduced in [PO]. The authors of [PO] introduced new shifted discrete grids in order to eliminate the aliasing error in three dimensional convolutions arising in the Navier-Stokes equations. As we show here shifted discrete grids can be applied to the rigorous methods for dPDEs in 1D and apparently yield a substantial improvement.

1.3.1 Aliasing problem

When the convolutions like (1.3) are calculated using the approach outlined in Section 1.2.1 the so-called aliasing problem appears. First, in order to explain what aliasing problem is, let us focus on the particular case of the convolution of exactly two terms, i.e.

$$s_k := (u * u)_k = \sum_{\substack{k_1+k_2=k \\ -N\le k_i\le N}} u_{k_1} u_{k_2}, \quad k = -N, \dots, N,$$

First values of u at the grid points x_j are calculated using the $l_2 \to L_2$ transform

$$\hat{u}_j = \sum_{k=-N}^{N} u_k \exp(ikj2\pi/M), \quad j = 0, \dots, M-1,$$

Then the L_2 coefficients are multiplied

$$\hat{s}_j = \hat{u}_j \hat{u}_j = \sum_{k=-2N}^{2N} \sum_{k_1+k_2=k} u_{k_1} u_{k_2} \exp(ikj2\pi/M), \quad j = 0, \dots, M-1.$$

$L_2 \to l_2$ transforming $\hat{s}_j$ is expected to produce s_k's, but in general situation will produce an *"aliased"* result, which may not equal s_k. We discuss the aliasing problem and a solution below.

The coefficients $\hat{s}_j$ are the L_2 coefficients of the function

$$x \mapsto \sum_{k=-2N}^{2N} \left(\sum_{k_1+k_2=k} u_{k_1} u_{k_2} \right) \exp(ikx). \tag{1.27}$$

From Lemma 1.2.5 it follows that

$$\frac{1}{M}\sum_{j=0}^{M-1}\hat{s}_j\exp(-ikj2\pi/M)=\sum_{\substack{a\in\mathbb{Z}\\-2N\le k+aM\le 2N}}\left(\sum_{k_1+k_2=k+aM}u_{k_1}u_{k_2}\right)=$$

$$s_k+\sum_{\substack{a\in\mathbb{Z}\setminus\{0\}\\-2N\le k+aM\le 2N}}\left(\sum_{k_1+k_2=k+aM}u_{k_1}u_{k_2}\right)$$

The term

$$\sum_{\substack{a\in\mathbb{Z}\setminus\{0\}\\-2N\le k+sM\le 2N}}\left(\sum_{k_1+k_2=k+aM}u_{k_1}u_{k_2}\right)$$

is the aliasing error.

The aliasing error is not always present. In order to eliminate the aliasing error the following condition has to be satisfied

$$\forall k\in\{-N,\dots,N\}\quad \forall a\in\mathbb{Z}\setminus\{0\}\qquad k+aM>2N\quad \text{ or } k+sM<-2N.$$

It is easy to see that to find the bound on M, such that there are no terms contributing to the aliasing error, it is enough to take $k=\pm N$ and $a=\pm 1$. We obtain

$$N-M<-2N,\quad -N+M>2N,$$
$$M>3N.$$

Therefore when $M>3N$ the aliasing error is not present.

Convolution of n terms We set

$$s_k:=(u*\cdots*u)_k=\sum_{\substack{k_1+k_2+\cdots+k_n=k\\-N\le k_j\le N}}u_{k_1}u_{k_2}\cdots u_{k_n},\quad k=-N,\dots,N.$$

Consider now the convolution of n terms, $u*\cdots*u$. Then the result of the multiplication of the L_2 coefficients is the following

$$\hat{s}_j=\hat{u}_j\cdots\hat{u}_j.$$

Observe that $\hat{s}_j$ are the L_2-coefficients of the following function

$$x\mapsto\sum_{k=-nN}^{nN}\sum_{k_1+k_2+\cdots+k_n=k}u_{k_1}u_{k_2}\cdots u_{k_n}\exp(ikx2\pi/M),\quad j=0,\dots,M-1.$$

From Lemma 1.2.5 it follows that $L_2 \to l_2$ transform of $\hat{s}_j$ gives

$$\tilde{s}_k := \frac{1}{M} \sum_{j=0}^{M-1} \hat{s}_j \exp(-ikj2\pi/M) =$$

$$\sum_{\substack{a\in\mathbb{Z} \\ -nN\le k+aM\le nM}} \sum_{k_1+k_2+\cdots+k_n=k+aM} u_{k_1}\cdots u_{k_n} =$$

$$\sum_{k_1+k_2+\cdots+k_n=k} u_{k_1}\cdots u_{k_n} + \sum_{\substack{a\in\mathbb{Z}\setminus\{0\} \\ -nN\le k+aM\le nM}} \sum_{k_1+k_2+\cdots+k_n=k+aM} u_{k_1}\cdots u_{k_n}.$$

The term

$$\sum_{\substack{a\in\mathbb{Z}\setminus\{0\} \\ -nN\le k+aM\le nM}} \sum_{k_1+k_2+\cdots+k_n=k+aM} u_{k_1}\cdots u_{k_n}.$$

is the aliasing error. The the range of a appearing in the sum of the aliasing error is bounded, namely if $aM > (n+1)N$, then for all $k \in \{-N, \ldots, N-1, N\}$ holds $k + aM > nN$ and $k - aM < -nN$. Let $z := \max_{a\in\mathbb{N}}\{aM \le (n+1)N\}$.

Definition 1.3.1. *We call $\tilde{s}_k$'s as above* the aliased convolutions.

I Standard technique of aliasing error removal (padding) As we have demonstrated in the case of two factors, the aliasing error is eliminated simply by choosing the number of grid points M large enough. The following condition has to be satisfied in the general case

$$k + M > nN \quad \forall k \in \{-N, \ldots, N\}, \text{ and } k - M < -nN \quad \forall k \in \{-N, \ldots, N\},$$

we take the worst case condition and obtain

$$N - M < -nN, \quad -N + M > nN,$$
$$M > (n+1)N.$$

Therefore when $M > (n+1)N$ the aliasing error is not present.

II First technique based on phase shifts This technique is designed only for the functions, for which the Fourier coefficients $\{u_k\}$ are purely real or purely imaginary (for instance real odd or real even functions).

To remove aliasing error one may also use phase shift, that is calculate L_2 coefficients at the points belonging to the uniform grid shifted to the right by a constant value Δ

Let

$$\hat{u}_j^{\Delta} = \sum_{k=-N}^{N} u_k \exp(ik\,(j2\pi/M + \Delta)), \quad j = 0, \ldots, M-1,$$

$$\hat{s}_j^{\Delta} = \hat{u}_j \cdots \hat{u}_j.$$

It is easy to see that

$$\hat{s}_j^{\Delta} = \sum_{k=-nN}^{nN} \sum_{k_1+\cdots+k_n=k} u_{k_1}\cdots u_{k_n} \exp(ik\,(j2\pi/M+\Delta)), \quad j=0,\ldots,M-1.$$

Form Lemma 1.2.10 applied to function

$$x \mapsto \sum_{k=-nN}^{nN} \left(\sum_{k_1+\cdots+k_n=k} u_{k_1}\cdots u_{k_n} \right) \exp(ikx)$$

it follows that

$$\begin{aligned} s_k^{\Delta} := \frac{1}{M} \sum_{j=0}^{M-1} \hat{s}_j^{\Delta} \exp(-ik\,(j2\pi/M+\Delta)) = \\ \sum_{\substack{a\in\mathbb{Z} \\ -nN\le k+aM\le nN}} \exp(iaM\Delta) \sum_{k_1+k_2+\cdots+k_n=k+aM} u_{k_1}u_{k_2}\cdots u_{k_n} = \\ \sum_{k_1+k_2+\cdots+k_n=k} u_{k_1}u_{k_2}\cdots u_{k_n} + \\ \sum_{\substack{a\in\mathbb{Z}\setminus\{0\} \\ -nN\le k+aM\le nN}} \exp(iaM\Delta) \sum_{k_1+k_2+\cdots+k_n=k+aM} u_{k_1}u_{k_2}\cdots u_{k_n} \end{aligned} \tag{1.28}$$

The sum over a is giving rise to the aliasing error.

First we eliminate all terms in the aliasing error with $|a|>1$. For this the following condition has to be satisfied for all $k\in\{-N,\ldots,N\}$

$$k+2M > nN \quad \text{and } k-2M < -nN,$$

we take the worst case condition and obtain

$$N-2M < -nN, \quad -N+2M > nN,$$
$$M > \frac{(n+1)}{2}N.$$

Recall our assumption that the modes $\{u_k\}$ are purely real or purely imaginary. Hence all the sums $\sum_{k_1+k_2+\cdots+k_n=k+aM} u_{k_1}u_{k_2}\cdots u_{k_n}$ are either real or are pure imaginary. Assume for simplicity that it is real (for pure imaginary the reasoning is the same)

With the choice of $\Delta = \frac{\pi}{2M}$ we have

$$\exp(\pm i\Delta M) = \exp(\pm i\pi/2) = \pm i.$$

We see that terms in the aliasing error corresponding to $a=\pm 1$ are pure imaginary.

Hence we obtained following lemma

Lemma 1.3.2. *Let M be the number of the points on the grid. Let $u : \mathbb{R} \to \mathbb{C}$ be a 2π-periodic function such that u belongs to the subspace spanned by the functions $\{e_{-N}, \ldots, e_N\}$ and moreover is odd or even.*

Assume that $M > \frac{(n+1)}{2}N$, then $\mathrm{Re}\left(s_k^{\pi/(2M)}\right) = s_k$ (for u even or n even) or $\mathrm{Im}\left(s_k^{\pi/(2M)}\right) = s_k$ (for u odd and n odd), for all $-N \leq k \leq N$.

III Second technique based on phase shifts In this technique we eliminate the aliasing error using an even smaller number of the grid points M. But additional fee has to be paid, three transforms instead of one have to be evaluated in total.

We will use $s_k^{\pi/(2M)}$, $s_k^{\pi/M}$ and $\tilde{s}_k$ to infer from the value of s_k. Observe that

$$\exp(i\frac{\pi}{2M} \cdot M) = i, \quad \exp(i\frac{\pi}{M} \cdot M) = -1. \tag{1.29}$$

From Lemma 1.2.10 it follows that

$$\frac{1}{4}\left(2s_k^{\pi/(2M)} + s_k^{\pi/M} + \tilde{s}_k\right) = \sum_{\substack{a\in\mathbb{Z} \\ -nN\leq k+aM\leq nN}} \frac{1}{4}\left(2i^a + (-1)^a + 1\right) \sum_{k_1+k_2+\cdots+k_n=k+aM} u_{k_1}u_{k_2}\cdots u_{k_n} \tag{1.30}$$

Observe that for $|a| = 0, 1, 2, 3, 4$ the factor $\frac{1}{4}\left(2i^a + (-1)^a + 1\right)$ is equal to $1, \pm i/2, 0, \pm i/2, 1$ respectively.

Assume now that all u_k are real (the imaginary case is similar), then the term with $a = 0$ will give us s_k which is real, the term with $|a| = 1$ or $|a| = 3$ will contribute to the imaginary part and the term with $|a| = 2$ disappears. We remove all other terms by choosing M, such that for all $|a| \geq 4$ and all $k \in \{-N, \ldots, N\}$ holds $k + aM > nN$ and $k - aM < -nM$. This is implied by the following condition

$$-N + 4M > nN, \quad N - 4M < -nN$$
$$M > \frac{n+1}{4}N.$$

Therefore we have proved the following lemma

Lemma 1.3.3. *Let M be the number of the points on the grid. Let u: $\mathbb{R} \to \mathbb{C}$ be a 2π-periodic function such that u belongs to the subspace spanned by the functions $\{e_{-N}, \ldots, e_N\}$ and moreover is odd or even.*

Assume that $M > \frac{(n+1)}{4}N$, then $\mathrm{Re}\left(\frac{1}{4}\left(2s_k^{\pi/(2M)} + s_k^{\pi/M} + \tilde{s}_k\right)\right) = s_k$ (for u even or n even) or $\mathrm{Im}\left(\frac{1}{4}\left(2s_k^{\pi/(2M)} + s_k^{\pi/M} + \tilde{s}_k\right)\right) = s_k$ (for u odd and n odd), for all $-N \leq k \leq N$.

Now, let us put the presented techniques into the context of the FFT algorithm. Whatever aliasing elimination technique is used in the FFT approach for calculating convolutions, the $M \geq 2N+1$ relation has to be in any case satisfied to permit all modes (whose number is $2N+1$) be mapped by the mapping (1.11). First of all, when the required transforms are performed using the FFT algorithm, Technique III requires asymptotically more operations than Technique II. Namely, after simplifying constants

$$\text{Technique II requires } \frac{(n+1)}{2} N \log \frac{(n+1)}{2} N \text{ operations.}$$
$$\text{Technique III requires } 3\frac{(n+1)}{4} N \log \frac{(n+1)}{4} N \text{ operations.}$$

Still, first advantage of Technique III is when a limited number of "small transforms" is available, in such a situation meeting Technique II restrictions may require M to be a considerably larger number. Thus it may require less calculations to evaluate three smaller transforms in Technique III than one larger transform in Technique II. This is likely to happen when the set of "small transforms" available does not include the factor of 2.

Second advantage of Technique III is when M is a very large number, such that the structures used in calculations are barely stored in a computer's RAM memory, then further increase of M will likely result in a considerable efficiency drop, because parts of the structures will have to be moved into the computer's virtual memory.

1.3.2 Overestimates

As we are interested in rigorous computations, therefore all computations have to be performed using *the interval arithmetic*. In this subsection we address the question if the rigorous evaluation of the convolutions (1.3) using the FFT algorithm produces more or less overestimates than the direct evaluation. We will refer to *the interval FFT algorithm* as *the rigorous FFT algorithm*.

We have not found a comprehensive study of the overestimates that arise in the rigorous FFT algorithm in the existing literature. We have performed numerical tests in order to provide clear answer to the question if the rigorous evaluation of the convolutions (1.3) using the FFT algorithm produces less or more overestimates compared to the direct evaluation. Apparently, the rigorous FFT algorithm produces more overestimates, especially when sets of large diameter (of order > 1) are used. The considerably larger overestimates produced by the FFT compared to the direct method are probably, mainly due to several matrix-vector multiplication performed by the FFT algorithm, which we interpret as yet another instance of the wrapping effect, well-known among rigorous numeric community, for precise statement refer e.g. [ZLo].

In all the tests presented in this section the function u has been a real-valued and odd function, randomly picked for each single test. We have required that $\{u_k\}_{k=-N}^{N}$ satisfies $\operatorname{mid}[u_k] \in [-C/k^s, C/k^s]$, in order to mimic regular functions. $\operatorname{diam}(u_k)$ was the same for all the modes.

In each test we have used the techniques presented in Section 1.3.1 in order to eliminate the aliasing error. Each technique was executed using specific number of the grid points M, provided in each table. Each M has been factorized (e.g. $16 = 4 \cdot 4$) in order to indicate what "small transforms" were included in the FFT algorithm. We remark that in ours implementation of the FFT algorithm we have included the "small transforms" for the following factors: 5, 4, 3 and 2. Because two factors, which are not mutually prime (4 and 2) have been included, in ambiguous cases always the larger factor i.e. 4 has been chosen, because it involves less operations. We present the results in Figure 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12.

u_k diameter	maximum over coordinates of diameters s_k			
	direct eval	Algorithm FFT-B - FFT-C		
		I padding	II first phase shift	III second phase shift*
		$M = 24 = 4 \cdot 3 \cdot 2$	$M = 15 = 3 \cdot 4$	$M = 15 = 3 \cdot 4$
1e-05	2.24883e-05	0.000138663	0.000342648	0.000291918
0.0001	0.000224883	0.00138663	0.00342648	0.00291923
0.001	0.00224883	0.0138663	0.0342648	0.0291975
0.01	0.0225401	0.138663	0.344654	0.293096
0.1	0.269289	1.49388	4.65484	3.84513
1	8.12441	67.1403	302.976	244.252
2	30.2488	254.695	1177.64	947.817
5	180.622	1539.84	7231.75	5814.38

Figure 1.1: Tests for $N = 7$, $s_k := \sum_{k_1+k_2=k} u_{k_1} u_{k_2}$
*-in this case this technique has no advantages, because M is the same as in the first phase shift technique and two more convolution calculations are needed.

u_k diameter	maximum over coordinates of diameters s_k			
	direct eval	Algorithm FFT-B - FFT-C		
		I padding	II first phase shift	III second phase shift*
		$M = 72 = 4 \cdot 3 \cdot 3 \cdot 2$	$M = 45 = 5 \cdot 3 \cdot 3$	$M = 45 = 5 \cdot 3 \cdot 3$
1e-05	1.76984e-05	0.00045232	0.00121946	0.00110262
0.0001	0.000176984	0.0045232	0.0121946	0.0110272
0.001	0.00176984	0.045232	0.121946	0.110369
0.01	0.0184213	0.46557	1.42349	1.27813
0.1	0.340996	18.8056	82.8143	73.2078
1	21.8849	1677.01	7732.67	6824.61
2	85.7698	6662.83	30808.7	27188.2
5	529.425	41473.1	192097	169513

Figure 1.2: Tests for $N = 21$, $s_k := \sum_{k_1+k_2=k} u_{k_1} u_{k_2}$
*-in this case this technique has no advantages, because M is the same as in the first phase shift technique and two more convolution calculations are needed.

u_k diameter	maximum over coordinates of diameters s_k			
	direct eval	Algorithm FFT-B - FFT-C		
		I padding	II first phase shift	III second phase shift*
		$M = 180 = 5 \cdot 4 \cdot 3 \cdot 3$	$M = 120 = 5 \cdot 4 \cdot 3 \cdot 2$	$M = 120 = 5 \cdot 4 \cdot 3 \cdot 2$
1e-05	1.94003e-05	2.4104e-05	3.05321e-05	6.19881e-05
0.0001	0.000194003	0.000241041	0.000305322	0.000619884
0.001	0.00194014	0.00241108	0.00305427	0.00620226
0.01	0.0197268	0.0247851	0.0315998	0.0654602
0.1	0.674988	1.29889	2.10665	7.86464
1	2387.47	25275.7	46596.8	241910
2	56872.9	734152	1.36271e+06	7.30569e+06
5	4.59896e+06	6.76534e+07	1.26072e+08	6.89313e+08

Figure 1.3: Tests for $N = 59$, $s_k := \sum_{k_1+k_2=k} u_{k_1} u_{k_2}$
*-in this case this technique has no advantages, because M is the same as in the first phase shift technique and two more convolution calculations are needed.

u_k diameter	maximum over coordinates of diameters s_k			
	direct eval	Algorithm FFT-B - FFT-C		
		I padding	II first phase shift	III second phase shift*
		$M = 32 = 4 \cdot 4 \cdot 2$	$M = 15 = 5 \cdot 3$	$M = 15 = 5 \cdot 3$
1e-05	1.95128e-05	3.09146e-05	0.000107253	9.4862e-05
0.0001	0.000195128	0.000309146	0.00107254	0.000948623
0.001	0.00195129	0.00309164	0.010728	0.00948838
0.01	0.0197051	0.0311039	0.110178	0.0972894
0.1	0.290143	0.550873	4.52371	3.70483
1	45.757	235.82	2976.32	2304.46
2	309.917	1794.37	23243.3	17957
5	4347.81	27200.8	357952	276460

Figure 1.4: Tests for $N = 7$, $s_k := \sum_{k_1+k_2+k_3=k} u_{k_1} u_{k_2} u_{k_3}$
*-in this case this technique has no advantages, because M is the same as in the first phase shift technique and two more convolution calculations are needed.

u_k diameter	maximum over coordinates of diameters s_k			
	direct eval	Algorithm FFT-B - FFT-C		
		I padding	II first phase shift	III second phase shift*
		$M = 90 = 5 \cdot 3 \cdot 3 \cdot 2$	$M = 45 = 5 \cdot 3 \cdot 3$	$M = 45 = 5 \cdot 3 \cdot 3$
1e-05	6.87813e-05	0.00176142	0.00297904	0.00269565
0.0001	0.000687813	0.0176144	0.0297908	0.0269568
0.001	0.00688878	0.176266	0.298266	0.269876
0.01	0.0729481	1.88594	3.33923	3.00745
0.1	1.70614	165.085	430.783	373.935
1	406.774	121698	343497	296924
2	2883.3	956535	2.71295e+06	2.34458e+06
5	42922	1.48829e+07	4.225e+07	3.65114e+07

Figure 1.5: Tests for $N = 21$, $s_k := \sum_{k_1+k_2+k_3=k} u_{k_1} u_{k_2} u_{k_3}$
*-in this case this technique has no advantages, because M is the same as in the first phase shift technique and two more convolution calculations are needed.

u_k diameter	maximum over coordinates of diameters s_k			
	direct eval	Algorithm FFT-B - FFT-C		
		I padding	II first phase shift	III second phase shift*
		$M = 240 = 5 \cdot 4 \cdot 4 \cdot 3$	$M = 120 = 5 \cdot 4 \cdot 3 \cdot 2$	$M = 120 = 5 \cdot 4 \cdot 3 \cdot 2$
1e-05	0.000148806	0.00754909	0.0174181	0.014963
0.0001	0.00148841	0.075493	0.174187	0.149634
0.001	0.0150371	0.757057	1.74801	1.50125
0.01	0.180683	10.1283	24.216	20.3733
0.1	6.37341	2449.98	6742.48	5410.66
1	2884.45	2.12684e+06	5.94012e+06	4.74766e+06
2	21796.3	1.68787e+07	4.71802e+07	3.77002e+07
5	328737	2.6246e+08	7.34007e+08	5.86441e+08

Figure 1.6: Tests for $N = 59$, $s_k := \sum_{k_1+k_2+k_3=k} u_{k_1} u_{k_2} u_{k_3}$

*-in this case this technique has no advantages, because M is the same as in the first phase shift technique and two more convolution calculations are needed.

u_k diameter	maximum over coordinates of diameters s_k			
	direct eval	Algorithm FFT-B - FFT-C		
		I padding	II first phase shift	III second phase shift
		$M = 48 = 4 \cdot 4 \cdot 3$	$M = 24 = 4 \cdot 3 \cdot 2$	$15 = 5 \cdot 3$
1e-05	1.94003e-05	2.4104e-05	3.05321e-05	6.19881e-05
0.0001	0.000194003	0.000241041	0.000305322	0.000619884
0.001	0.00194014	0.00241108	0.00305427	0.00620226
0.01	0.0197268	0.0247851	0.0315998	0.0654602
0.1	0.674988	1.29889	2.10665	7.86464
1	2387.47	25275.7	46596.8	241910
2	56872.9	734152	1.36271e+06	7.30569e+06
5	4.59896e+06	6.76534e+07	1.26072e+08	6.89313e+08

Figure 1.7: Tests for $N = 7$, $s_k := \sum_{k_1+k_2+k_3+k_4+k_5=k} u_{k_1} u_{k_2} u_{k_3} u_{k_4} u_{k_5}$

u_k diameter	maximum over coordinates of diameters s_k			
	direct eval	Algorithm FFT-B - FFT-C		
		I padding	II first phase shift	III second phase shift
		$M = 144 = 4 \cdot 4 \cdot 3 \cdot 3$	$M = 72 = 4 \cdot 3 \cdot 3 \cdot 2$	$M = 45 = 5 \cdot 3 \cdot 3$
1e-05	7.15846e-05	0.000709162	0.001178	0.00160417
0.0001	0.000715878	0.00709179	0.0117803	0.0160426
0.001	0.00718731	0.0710905	0.118113	0.161287
0.01	0.752179	10.4391	18.3479	25.5228
0.1	36.2275	1696.94	2861.64	12889.4
1	184009	5.98891e+07	8.89027e+07	6.34577e+08
2	4.63014e+06	1.80823e+09	2.65626e+09	1.95288e+10
5	3.8902e+08	1.70558e+11	2.48932e+11	1.86307e+12

Figure 1.8: Tests for $N = 21$, $s_k := \sum_{k_1+k_2+k_3+k_4+k_5=k} u_{k_1} u_{k_2} u_{k_3} u_{k_4} u_{k_5}$

Remark 1.3.4. We would like to present some conclusions from the test we have performed.

- Apparently, the rigorous FFT algorithm produces overestimates larger by

u_k diameter	maximum over coordinates of diameters s_k			
	direct eval	Algorithm FFT-B - FFT-C		
		I padding	II first phase shift	III second phase shift
		$M = 360 = 5 \cdot 4 \cdot 3 \cdot 3 \cdot 2$	$M = 180 = 5 \cdot 4 \cdot 3 \cdot 3$	$120 = 5 \cdot 4 \cdot 3 \cdot 2$
1e-05	0.000530659	0.0292587	0.0512855	0.0349744
0.0001	0.0053115	0.292628	0.51297	0.34979
0.001	0.0549025	2.96643	5.24524	3.54432
0.01	0.906118	78.0069	219.8	90.0389
0.1	323.51	752838	3.92654e+06	776709
1	8.52507e+06	5.84354e+10	3.32134e+11	5.88342e+10
2	2.38477e+08	1.82509e+12	1.04585e+13	1.83279e+12
5	2.27425e+10	1.77844e+14	1.02007e+15	1.78557e+14

Figure 1.9: Tests for $N = 59$, $s_k := \sum_{k_1+k_2+k_3+k_4+k_5=k} u_{k_1} u_{k_2} u_{k_3} u_{k_4} u_{k_5}$

u_k diameter	maximum over coordinates of diameters s_k			
	direct eval	Algorithm FFT-B - FFT-C		
		I padding	II first phase shift	III second phase shift
		$M = 48 = 4 \cdot 4 \cdot 3$	$M = 24 = 4 \cdot 3 \cdot 2$	$15 = 5 \cdot 3$
1e-05	7.55469e-05	0.000107663	0.000125186	0.000321883
0.0001	0.000755525	0.00107663	0.00125186	0.00321885
0.001	0.00756184	0.0107685	0.0125219	0.0322085
0.01	0.0802333	0.109881	0.128471	0.342257
0.1	3.99075	3.96786	5.8809	46.2991
1	157234	449455	1.0882e+06	3.00292e+07
2	1.20857e+07	4.35324e+07	1.09262e+08	3.32159e+09
5	5.31801e+09	2.25963e+10	5.78319e+10	1.86328e+12

Figure 1.10: Tests for $N = 7$, $s_k := \sum_{k_1+\cdots+k_7=k} u_{k_1} \cdots u_{k_7}$

u_k diameter	maximum over coordinates of diameters s_k			
	direct eval	Algorithm FFT-B - FFT-C		
		I padding	II first phase shift	III second phase shift
		$M = 144 = 4 \cdot 4 \cdot 3 \cdot 3$	$M = 72 = 4 \cdot 3 \cdot 3 \cdot 2$	$M = 45 = 5 \cdot 3 \cdot 3$
1e-05	3.31821e-05	8.34889e-05	0.000334387	0.000283706
0.0001	0.00033196	0.000834932	0.00334441	0.00283755
0.001	0.00336929	0.00839214	0.0339857	0.0288688
0.01	0.0592131	0.131418	1.06813	0.997794
0.1	73.8556	2062.29	135328	198427
1	5.65913e+07	9.38222e+09	8.16458e+11	1.34344e+12
2	6.27298e+09	1.16532e+12	1.02332e+14	1.69511e+14
5	3.50596e+12	6.9871e+14	6.16814e+16	1.02581e+17

Figure 1.11: Tests for $N = 21$, $s_k := \sum_{k_1+\cdots+k_7=k} u_{k_1} \cdots u_{k_7}$

an order of magnitude compared to the direct method. The overestimates are becoming larger with the increase of the number of modes N, this is probably related to the matrix-vector multiplications. In the Lohner algorithm the related problem (the wrapping effect) is solved by avoiding the matrix-vector multiplications, here unfortunately one cannot proceed

u_k diameter	maximum over coordinates of diameters s_k			
	direct eval	Algorithm FFT-B - FFT-C		
		I padding	II first phase shift	III second phase shift
		$M = 360 = 5 \cdot 4 \cdot 3 \cdot 3 \cdot 2$	$M = 180 = 5 \cdot 4 \cdot 3 \cdot 3$	$120 = 5 \cdot 4 \cdot 3 \cdot 2$
1e-05	0.000413601	0.00769049	0.0136309	0.0151676
0.0001	0.0041454	0.0769299	0.136379	0.151763
0.001	0.0458016	0.625406	1.44914	1.26319
0.01	1.67176	36.6444	176.258	160.807
0.1	12669.5	1.11581e+07	1.04037e+08	1.0332e+08
1	2.61257e+10	7.56577e+13	7.71902e+14	7.65992e+14
2	3.02085e+12	9.47839e+15	9.72051e+16	9.64349e+16
5	1.76386e+15	5.75457e+18	5.8919e+19	5.86183e+19

Figure 1.12: Tests for $N = 59$, $s_k := \sum_{k_1+\cdots+k_7=k} u_{k_1} \cdots u_{k_7}$

like this, because the whole profit from using the FFT is from performing sparse matrix-vector multiplications. Another source of the issue is probably poor procedure (done in the straightforward way) of multiplying complex numbers. We are aware that there has been work investigating optimal ways of performing complex linear algebra, see [HK], which should be applied for our purposes. We are convinced that the overestimates can be reduced and we will investigate possibilities of reducing the overestimates arising in the FFT algorithm in our forthcoming work.

1.4 A rigorous DPDE integrator

To explain what does the term *rigorous integration* mean let us consider the following abstract Cauchy problem for a system of ordinary differential equations (ODEs)

$$\begin{cases} \dot{x}(t) &= f(x(t)), \\ x(0) &= x_0. \end{cases} \tag{1.31}$$

$x\colon [0,\mathcal{T}) \to \mathbb{R}^n$, $f\colon \mathbb{R}^n \to \mathbb{R}^n$, $f \in C^\infty$. The goal of a rigorous ODEs solver is to find a set $\mathbf{x_k} \subset \mathbb{R}^n$ compact and connected such that

$$\varphi(t_k, \mathbf{x_0}) \subset \mathbf{x_k}, \tag{1.32}$$

$t_k \in [0,\mathcal{T})$, $\quad \mathbf{x_0} \subset \mathbb{R}^n$, $\varphi(t_k, \mathbf{x_0})$ is a solution of (1.31) at time t_k.

Notation We denote by $[x]$ an *interval set* $[x] \subset \mathbb{R}^n$, $[x] = \Pi_{k=1}^n [x_k^-, x_k^+]$, $[x_k^-, x_k^+] \subset \mathbb{R}$, $-\infty < x_k^- \leq x_k^+ < \infty$, mid $([x])$ is the middle of an interval set $[x]$ and r $([x])$ is the rest, i.e. $[x] = \text{mid}\,([x]) + \text{r}\,([x])$. There exist a few algorithms that offer reliable computations of the solution trajectories for ODEs that are based on interval arithmetic. The approach used in this paper is based on the Lohner algorithm, introduced in [Lo] , see also [ZLo]. To obtain a rigorous bound for the solution of a system of ODEs, and avoid the so-called *wrapping effect* it is necessary to calculate the partial derivative with respect to the initial conditions.

Basically in the Lohner algorithm (see e.g. [ZLo]) instead of calculating directly the expression

$$\Phi_q([x],h) + R_{\Phi_q}([W]) \supset \varphi([x],h),$$

the mean value form is used

$$\Phi_q(\mathrm{mid}\,([x]),h) + \partial\Phi_q/\partial x\,([x],h)\,(\mathrm{mid}\,([x]) - [x]) + R_{\Phi_q}([W]) \supset \varphi([x],h),$$

$h > 0$ is a time step, Φ_q is a q-th order numerical method (for the purpose of this paper *the Taylor method*) for an equation (1.31), R_{Φ_q} is a remainder term calculated using a convex hull of an enclosure of the solution values over the time interval $[0,h]$, i.e. a set $[W] \subset \mathbb{R}^n$ such that $\varphi([0,h],[x]) \subset [W]$. Incorporating a higher order information has some advantages over the first order information of the Lohner approach, we do not discuss this topic in detail here, we refer the interested reader to the cited literature and references therein.

The bounds for the vector

$$\varphi([x],h), \tag{1.33}$$

and the bounds for the partial derivative with respect to initial condition

$$\partial\Phi_q/\partial x\,([x],h) \tag{1.34}$$

are both obtained by the Taylor method in our approach. Here we do not address in detail the Taylor method and the Lohner algorithm we refer the reader to [S] and [ZLo].

Obtaining Taylor coefficients of any order Now we address the question how to efficiently obtain bounds for the Taylor coefficients of any order.

The algorithms from [ZLo] were using some explicit formulas for the Taylor coefficients of any order. These explicit formulas were derived for the second degree polynomials only. This approach has been used in the following work considering rigorous dPDE integration, i.e. [ZM], [Z2], [Z3] and [ZAKS]. We found this approach awkward, mainly because of its troublesome implementation and limited applicability, apparently this approach is limited, and works only for dPDEs with the nonlinear term being a second degree polynomial.

Instead, we found more suitable to use an algorithm that combines *the jet transport* and *the automatic differentiation* techniques. These techniques have already proven to be extremely useful in many applications including a long-time integration of the solar system, see [JZ] and references given therein, or computer assisted proofs for ODEs.

1.4.1 Automatic differentiation

Now, let us address the question how to efficiently obtain the derivatives with respect to time of a sufficiently smooth function. Here we present a convenient and yet efficient approach called *the automatic differentiation*, see e.g. [G]. *The*

automatic differentiation has many applications besides the particular of our interest, it is a general technique of obtaining a fixed number of normalized derivatives of a general operation/procedure in a recursive way.

Definition 1.4.1. *Let $n > 0$, $a\colon [0, t_{max}) \to \mathbb{R}$ be a sufficiently smooth function, we call*

$$a^{[n]}(t) = \frac{1}{n!} a^{(n)}(t)$$

its n-th normalized derivative.

Let $a(t) = G(b(t), c(t))$, using the automatic differentiation the normalized derivatives of $a(t)$ can be obtained assuming that G is a composition of some basic operations for which the recursive evaluation formulas are known. Recursive formulas for basic operations (like multiplication, exponential, logarithm, trigonometric functions etc.) are easily derived, see for example [JZ].

Only particular systems originating from a dPDE with nonlinearity given by a polynomial are in the scope of this dissertation. Rules for addition and multiplication of two functions are required only. This will became apparent later on. Scalar multiplication being trivial is not presented here. Assume that $b^{[j]}, c^{[j]}$ are known for $j = 0, \ldots, r$ then

$$\text{if } a(t) = b(t) + c(t) \text{ then } a^{[r]}(t) = b^{[r]}(t) + c^{[r]}(t), \tag{1.35}$$

$$\text{if } a(t) = b(t) \cdot c(t) \text{ then } a^{[r]}(t) = \sum_{j=0}^{r} b^{[j]}(t) \cdot c^{[r-j]}(t). \tag{1.36}$$

The second rule can be proved by induction, starting with the following elementary rule of taking the derivative of a product of two functions

$$(fg)' = f'g + fg'.$$

Now let us consider a differential equation

$$\frac{dx}{dt} = F(x), \quad x \in \mathbb{R}^n, \quad F\colon \mathbb{R}^n \to \mathbb{R}^n. \tag{1.37}$$

Given $x^{[0]} = x^{(0)}$ only we are interested in calculating $x^{[j]}$ for $j = 1, \ldots, q-1$, this is done by the following recursive formula

$$x^{[j+1]} = \frac{1}{j+1} F^{[j]},$$

where $F^{[j]}$ is the j-th normalized derivative of the right-hand side function of (1.37).

1.4.2 Jet transport

Now, the question is how to obtain bounds for the partial derivative with respect to initial condition (1.34). The most convenient approach is to modify the automatic differentiation procedure to make it calculate (1.33) and (1.34) simultaneously. This is done by the *jet transport* technique.

Definition 1.4.2. *Let $n > 0$ be a fixed integer, $\hat{a} \in \mathbb{R}$, $a_j \in \mathbb{R}$, $j = 0, \ldots, n-1$. We call a first degree polynomial of n variables $\boldsymbol{\xi} = (\xi_0, \xi_1, \ldots, \xi_{n-1})$*

$$a\left(\boldsymbol{\xi}\right) := \check{a} + \sum_{j=0}^{n-1} a_j \xi_j$$

the first order jet.

An algebra of first order jets is formed with the addition and multiplication operations defined as follows

Definition 1.4.3. *Let $n > 0$, $a(\boldsymbol{\xi}), b(\boldsymbol{\xi})$ be first order jets, $\boldsymbol{\xi} = (\xi_0, \ldots, \xi_{n-1})$ are symbols. We define the addition $+$ of first order jets as follows*

$$a(\boldsymbol{\xi}) + b(\boldsymbol{\xi}) := \check{a} + \check{b} + \sum_{j=0}^{n-1} \left(a_j + b_j\right) \xi_j.$$

We define the multiplication $\cdot$ of first order jets as follows

$$a(\boldsymbol{\xi}) \cdot b(\boldsymbol{\xi}) := \check{a} \cdot \check{b} + \sum_{j=0}^{n-1} \left(\check{a} \cdot b_j + \check{b} \cdot a_j\right) \xi_j.$$

Definition 1.4.4. *Let $f \colon \mathbb{R}^n \to \mathbb{R}$, $t \geq 0$ be a time, $x_0 \in \mathbb{R}^n$ be a fixed value, $\boldsymbol{\xi} = (\xi_0, \ldots, \xi_{n-1})$ are symbols. We define the following first order jet for the function f*

$$J_f\left(\boldsymbol{\xi}\right) := f(x_0) + \sum_{l=0}^{n-1} \frac{\partial f}{\partial x_l}\left(x_0\right) \xi_l.$$

The definition above has a straightforward extension to the case when f is vector-valued. When f is vector-valued the vector of jets for f is denoted by $\mathbf{J_f}\left(\boldsymbol{\xi}\right)$. Using the definition above we define the first order jets for $\varphi(t, x_0)$ and calculate

$$\varphi^{[j]}(t, x_0), \qquad j = 1, \ldots, q-1$$

using the automatic diffrentation procedure described in Section 1.4.1. Where $\varphi(t, x_0)$ is the smooth solution of (1.37),

$$\varphi^{[j]}(t, x_0) := \frac{1}{j!} \frac{\partial^j \varphi(t, x_0)}{\partial t^i}, \qquad \left(\frac{\partial \varphi}{\partial x}\right)^{[j]}(t, x_0) := \frac{1}{j!} \frac{\partial^j}{\partial t^i} \frac{\partial \varphi(t, x_0)}{\partial x}, \qquad j = 1, \ldots, q-1,$$

Definition 1.4.5. *Let $f\colon \mathbb{R}^n \to \mathbb{R}^n$. We call the matrix*

$$\left(\frac{\partial f}{\partial x_0}(x_0), \ldots, \frac{\partial f}{\partial x_{n-1}}(x_0) \right)^T \in \mathbb{R}^{n \times n}$$

the variational part *of the jets* $\mathbf{J_f}(\boldsymbol{\xi})$. *Whereas the vector*

$$f(x_0) \in \mathbb{R}^n$$

the base part *of the jets* $\mathbf{J_f}(\boldsymbol{\xi})$.

In what follows in addition to the regular Taylor method (Φ_q)**, we are going to use an extended variant** (Φ_q^{jet}) **whose coefficients are** $\left\{\mathbf{J}_{\varphi^{[0]}}(\boldsymbol{\xi}), \ldots, \mathbf{J}_{\varphi^{[\mathbf{q}-1]}}(\boldsymbol{\xi})\right\}$**.** Φ_q^{jet} **works as follows the base part of** $\mathbf{J}_{\varphi^{[0]}}(\boldsymbol{\xi})$ **is set to the initial condition, the variational part of** $\mathbf{J}_{\varphi^{[0]}}(\boldsymbol{\xi})$ **is set to the identity. The higher order coefficients are calculated performing the same operations as in the regular method, but on first order jets, and operations are performed with the rules of the algebra defined in Definition 1.4.3. This allows, with the same code, solve** (1.37) **and the variational equations corresponding to it.**

Observe that when higher order variational data is required, the presented technique can be extended to obtain higher order variational data (for instance *the Hessian*). Higher order jets, being higher degree *homogeneous polynomials* could be used instead of first order jets, along with an algebra, which is defined analogously.

1.4.3 An algorithm combining the Automatic Differentiation with the FFT

In this section we propose an algorithm for calculating both the solution of systems of ODEs originating from a dPDE and a matrix of partial with respect to the initial condition derivatives.

We emphasize the fact that the output of the presented algorithm is not enough to integrate a system forward in time. This is not a stand-alone algorithm. But the output of the presented algorithm can be used as the input to the Lohner algorithm in order to rigorously integrate the system forward in time.

The algorithm has been designed for dPDEs written using the truncated Fourier basis. We consider the following abstract system

$$\frac{da_k}{dt} = N_k(a) + \lambda_k a_k = F_k(a), \quad k = -N, \ldots, N, \tag{1.38}$$

Notation $a := \{a_k\}_{k=-N}^{N} \subset \mathbb{C}$ is a finite set of the Fourier coefficients, $N_k(a)$ is a nonlinear term comprising a convolution ($\sum_{k_1+k_2=k} a_{k_1} a_{k_2}$ for instance). Let $\varphi([0,h],[x_0]) \subset \mathbb{C}^{2N+1}$ denotes all solution values with initial condition in $[x_0] \subset \mathbb{C}^{2N+1}$ and within the time interval $[0,h]$.

From now on let φ be a regular with respect to time solution of (1.38). Observe that the coefficients $\{a_k\}_{k=-N}^{N}$ are complex in this case, and $\varphi\colon [0, t_{max}) \times \mathbb{C}^{2N+1} \to \mathbb{C}^{2N+1}$.

First, let us comment on the approach to the first order jets of complex functions.

Definition 1.4.6. *Let $\{a_k\}_{k=-N}^{N} \subset \mathbb{C}$ satisfies $a_{-k} = \overline{a_k}$. We define $\Pi_{Re}\colon \mathbb{C}^{2N+1} \to \mathbb{R}^{2N+1}$ to be the following projection*

$$\Pi_{Re}\left(a_{-N}, \ldots, a_N\right) = \left(\mathrm{Re}\left(a_0\right), \mathrm{Im}\left(a_0\right), \ldots, \mathrm{Re}\left(a_N\right), \mathrm{Im}\left(a_N\right)\right).$$

Any function

$$\left(a_{-N}, \ldots, a_N\right) \mapsto f\left(a_{-N}, \ldots, a_N\right),$$

can be translated to a function of real variables

$$\left(\mathrm{Re}\left(a_0\right), \mathrm{Im}\left(a_0\right), \ldots, \mathrm{Re}\left(a_N\right), \mathrm{Im}\left(a_N\right)\right) \mapsto \widetilde{f}\left(\mathrm{Re}\left(a_0\right), \mathrm{Im}\left(a_0\right), \ldots, \mathrm{Re}\left(a_N\right), \mathrm{Im}\left(a_N\right)\right)$$

in such a way that if $a_{-k} = \overline{a_k}$ then

$$f\left(a_{-N}, \ldots, a_N\right) = \widetilde{f}\left(\Pi_{Re}\left(a_{-N}, \ldots, a_N\right)\right).$$

Then we may translate our system (1.38) and use real first order jets in order to calculate the partial derivative of the numerical method with respect to the initial condition (1.34).

Instead we use "complex" first order jets, which have the following form, let $\left(a_{-N}, \ldots, a_N\right) \mapsto f\left(a_{-N}, \ldots, a_N\right) \in \mathbb{C}$

$$J_f\left(\boldsymbol{\xi}\right) := f(x_0) + \sum_{l=-N}^{N} \begin{bmatrix} \frac{\partial \mathrm{Re}(f)}{\partial \mathrm{Re}(a_l)} + i\frac{\partial \mathrm{Im}(f)}{\partial \mathrm{Re}(a_l)} \\ \frac{\partial \mathrm{Re}(f)}{\partial \mathrm{Im}(a_l)} + i\frac{\partial \mathrm{Im}(f)}{\partial \mathrm{Im}(a_l)} \end{bmatrix} \xi_l. \tag{1.39}$$

Using this kind of jets greatly facilitates computations and permits certain optimizations. This approach is strictly of computational interest, we do not intend to give any exhaustive theory here. We provide below a sketch of a justification of such approach

Let $a(\xi)$ and $b(\xi)$ be arbitrary "complex" first order jets for two complex variables

$$a(\xi) = \check{c} + i\check{d} + \begin{bmatrix} c_1 + id_1 \\ c_2 + id_2 \end{bmatrix} \xi_1 + \begin{bmatrix} c_3 + id_3 \\ c_4 + id_4 \end{bmatrix} \xi_2,$$
$$b(\xi) = \check{e} + i\check{f} + \begin{bmatrix} e_1 + if_1 \\ e_2 + if_2 \end{bmatrix} \xi_1 + \begin{bmatrix} e_3 + if_3 \\ e_4 + if_4 \end{bmatrix} \xi_2,$$

Let us take the corresponding real first order jets $c = \mathrm{Re}\left(a\right)$, $d = \mathrm{Im}\left(a\right)$, $e = \mathrm{Re}\left(b\right)$ and $f = \mathrm{Im}\left(b\right)$

$$c(\xi) = \check{c} + c_1\xi_{11} + c_2\xi_{12} + c_3\xi_{21} + c_4\xi_{22},$$
$$d(\xi) = \check{d} + d_1\xi_{11} + d_2\xi_{12} + d_3\xi_{21} + d_4\xi_{22},$$
$$e(\xi) = \check{e} + e_1\xi_{11} + e_2\xi_{12} + e_3\xi_{21} + e_4\xi_{22},$$
$$f(\xi) = \check{f} + f_1\xi_{11} + f_2\xi_{12} + f_3\xi_{21} + f_4\xi_{22}.$$

If

$$h(\xi) = a(\xi) \cdot b(\xi),$$
$$h_1(\xi) = c(\xi) \cdot e(\xi) - d(\xi) \cdot f(\xi), \text{ and}$$
$$h_2(\xi) = c(\xi) \cdot f(\xi) + e(\xi) \cdot d(\xi),$$

then the following relation holds $h(\xi) = h_1(\xi) + ih_2(\xi)$.

We would like to present below an outline of the efficient algorithm for calculating a rigorous normalized derivatives of the solution along with the partial derivative with respect to initial condition for systems of the type (1.38). Let us comment what are the main ingredients of the described algorithm. First, the information about the partial with respect to initial condition derivative is obtained by introducing the first order jets in place of ordinary scalar values. Second, first order jets are being passed through the FFT algorithm in order to minimize the number of operations for calculating the convolutions. Third, in order to allow fast calculation of the automatic differentiation convolutions the L_2 coefficients of the normalized derivatives are calculated and stored at each step.

Notation In order to facilitate the explanation of the following algorithm we are going to use the following auxiliary symbols $a^{[j]}(\boldsymbol{\xi}) := \mathbf{J}_{\varphi[j]}(\boldsymbol{\xi})$, $N^{[j]}(\boldsymbol{\xi})$ denotes the first order jet for j-th normalized derivative of the nonlinear part of (1.38), $F^{[j]}(\boldsymbol{\xi})$ denotes the first order jet for the j-th normalized derivative of the right-hand side of (1.38). With hats $\hat{a}^{[j]}(\boldsymbol{\xi})$, $\hat{N}^{[j]}(\boldsymbol{\xi})$ and $\hat{F}^{[j]}(\boldsymbol{\xi})$ we indicate the corresponding L_2 coefficients.

Algorithm 1. Outline of an algorithm for efficient and rigorous calculation of the normalized derivatives, dedicated for any system of the type (1.38).

Input

- an interval set of initial conditions $[x_0] \subset \mathbb{C}^{2N+1}$,
- time step $h > 0$,
- order of the Taylor method $q > 0$.

Output

- the normalized derivatives

$$\left\{ \left(a_{-N}^{[1]}(\boldsymbol{\xi}), \ldots, a_N^{[1]}(\boldsymbol{\xi}) \right), \ldots, \left(a_{-N}^{[q]}(\boldsymbol{\xi}), \ldots, a_N^{[q]}(\boldsymbol{\xi}) \right) \right\},$$

  which comprise the Taylor coefficients and the partial derivatives with respect to initial condition of the coefficients.

begin

1. Calculate a rough-enclosure $[W] \subset \mathbb{C}^{2N+1}$, such that $\varphi([0,h],[x_0]) \subset [W]$ using the algorithm presented in [Z3].

2. Calculate $R_{\Phi_q}([W])$, the remainder of the Taylor method of order q,

3. Initialize calculations for Φ_q^{jet}. The base part of the jets $\left(a_{-N}^{[0]}(\boldsymbol{\xi}),\ldots,a_N^{[0]}(\boldsymbol{\xi})\right)$ is set to be equal to the provided initial condition $[x_0]$. The variational part of the jets $\left(a_{-N}^{[0]}(\boldsymbol{\xi}),\ldots,a_N^{[0]}(\boldsymbol{\xi})\right)$ is set to the identity, i.e.

$$\frac{\partial \operatorname{Re}\left(\varphi_k^{[0]}\right)}{\partial \operatorname{Re}(a_l)} := \begin{cases} 1 & \text{, for } k=l \\ 0 & \text{otherwise.} \end{cases}$$

$$\frac{\partial \operatorname{Im}\left(\varphi_k^{[0]}\right)}{\partial \operatorname{Im}(a_l)} := \begin{cases} 1 & \text{, for } k=l, \\ -1 & \text{, for } k=-l, \\ 0 & \text{otherwise.} \end{cases}$$

$$\frac{\partial \operatorname{Re}\left(\varphi_k^{[0]}\right)}{\partial \operatorname{Im}(a_l)}, \frac{\partial \operatorname{Im}\left(\varphi_k^{[0]}\right)}{\partial \operatorname{Re}(a_l)} := 0.$$

4. $l_2 \to L_2$ transform $\left(a_{-N}^{[0]}(\boldsymbol{\xi}),\ldots,a_N^{[0]}(\boldsymbol{\xi})\right)$ to obtain $\left(\hat{a}_0^{[0]}(\boldsymbol{\xi}),\ldots,\hat{a}_M^{[0]}(\boldsymbol{\xi})\right)$

   for $j=0,\ldots,q-1$

   (a) calculate $\left(\hat{N}_0^{[j]}(\boldsymbol{\xi}),\ldots,\hat{N}_M^{[j]}(\boldsymbol{\xi})\right)$ using the L_2 coefficients calculated in the previous steps $\left\{\left(\hat{a}_0^{[0]}(\boldsymbol{\xi}),\ldots,\hat{a}_M^{[0]}(\boldsymbol{\xi})\right),\ldots,\left(\hat{a}_0^{[j]}(\boldsymbol{\xi}),\ldots,\hat{a}_M^{[j]}(\boldsymbol{\xi})\right)\right\}$,

   (b) calculate $\left(N_{-N}^{[j]}(\boldsymbol{\xi}),\ldots,N_N^{[j]}(\boldsymbol{\xi})\right)$ by $L_2 \to l_2$ transforming $\left(\hat{N}_0^{[j]}(\boldsymbol{\xi}),\ldots,\hat{N}_M^{[j]}(\boldsymbol{\xi})\right)$,

   (c) add the linear contribution $\left(F_{-N}^{[j]}(\boldsymbol{\xi}),\ldots,F_N^{[j]}(\boldsymbol{\xi})\right) := \left(N_{-N}^{[j]}(\boldsymbol{\xi}),\ldots,N_N^{[j]}(\boldsymbol{\xi})\right) + \left(\lambda_{-N}a_{-N}^{[j]}(\boldsymbol{\xi}),\ldots,\lambda_N a_N^{[j]}(\boldsymbol{\xi})\right)$,

   (d) set $\left(a_{-N}^{[j+1]}(\boldsymbol{\xi}),\ldots,a_N^{[j+1]}(\boldsymbol{\xi})\right) := \frac{1}{j+1}\left(F_{-N}^{[j]}(\boldsymbol{\xi}),\ldots,F_N^{[j]}(\boldsymbol{\xi})\right)$,

   (e) calculate and store $\left(\hat{a}_0^{[j+1]}(\boldsymbol{\xi}),\ldots,\hat{a}_M^{[j+1]}(\boldsymbol{\xi})\right)$ by $l_2 \to L_2$ transforming $\left(a_{-N}^{[j+1]}(\boldsymbol{\xi}),\ldots,a_N^{[j+1]}(\boldsymbol{\xi})\right)$,

Notation Now we would like to comment on Step 4a of Algorithm 1, because that is where a lot of savings are being introduced by the FFT algorithm.

Basically the improvement of introducing the FFT algorithm into the standard procedure is as follows. At the given order j instead of calculating the double convolutions

$$\sum_{k_1+k_2+\cdots+k_n=k} \sum_{j_1+j_2+\cdots+j_n=j} a_{k_1}^{[j_1]}(\boldsymbol{\xi})\, a_{k_2}^{[j_2]}(\boldsymbol{\xi}) \cdots a_{k_n}^{[j_n]}(\boldsymbol{\xi}), \quad k = -N, \ldots, N \tag{1.40}$$

directly, the following operations, comprising steps 4a, 4b and 4e of Algorithm 1 are used, compare with the outline presented in Section 1.2.1

- one $l_2 \to L_2$ transform, and one $L_2 \to l_2$ transform,
- calculation of

$$\sum_{j_1+j_2+\cdots+j_n=j} \hat{a}_k^{[j_1]}(\boldsymbol{\xi})\, \hat{a}_k^{[j_2]}(\boldsymbol{\xi}) \cdots \hat{a}_k^{[j_n]}(\boldsymbol{\xi}), \qquad k = 0, \ldots, M. \tag{1.41}$$

Observe that in the FFT variant the loop over k is not present at all.

1.4.4 Optimizations

In this section we discuss what optimizations could be implemented in Algorithm 1.

Now, let us assume that the right-hand side of (1.38) contains the following convolution of two terms

$$\sum_{k_1+k_2=k} a_{k_1} a_{k_2}, \quad k = -N, \ldots, N.$$

We drop the notation of $(\boldsymbol{\xi})$ in the following description of the optimizations. In order to calculate the j-th normalized derivative of F (step 4b of Algorithm 1) the following sum is to be evaluated

$$\sum_{|k_1| \le N} \sum_{l=0}^{j} a_{k_1}^{[l]} a_{k-k_1}^{[j-l]}. \tag{1.42}$$

From now on we call the approach to the calculation of sums (1.42) in which everything is evaluated directly *the direct approach*. The approach in which the sum over k is not present, because the L_2 coefficients are used in place of the l_2 coefficients *the FFT approach*.

Now let us propose two possible optimizations reducing considerably the computational cost.

exploiting the $k_1 \to k - k_1$ symmetry of the sum

This optimization works for the direct approach only, cannot be applied for the FFT approach. It relies on exploiting the $k_1 \to k - k_1$ symmetry of the sum.

Lemma 1.4.7. *Let* $e(k) := \begin{cases} 1 & , k \text{ is divisible by } 2 \\ 0 & , \text{otherwise} \end{cases}$. *For* $k \geq 0$ *(for* $k < 0$ *analogously) we have*

$$\sum_{|k_1| \leq N} \sum_{l=0}^{j} a_{k_1}^{[l]} a_{k-k_1}^{[j-l]} = 2 \sum_{\frac{k}{2} < k_1 \leq N} \sum_{l=0}^{j} a_{k_1}^{[l]} a_{k-k_1}^{[j-l]} + e(k) \sum_{l=0}^{j} a_{k/2}^{[l]} a_{k/2}^{[j-l]}.$$

Proof: For $j = 0$ the above equality is just the $k_1 \to k - k_1$ symmetry of the sum

$$\sum_{|k_1| \leq N} a_{k_1} a_{k-k_1} = 2 \sum_{\frac{k}{2} < k_1 \leq N} a_{k_1} a_{k-k_1} + e(k) a_{k/2} a_{k/2}.$$

For $j > 0$ the $k_1 \to k - k_1$ symmetry of the sum still holds, because adding the sum over j does not destroy the $k_1 \to k - k_1$ symmetry

$$\sum_{l=0}^{j} a_{k_1}^{[l]} a_{k-k_1}^{[j-l]} = \sum_{l=0}^{j} a_{k-k_1}^{[l]} a_{k_1}^{[j-l]},$$

and therefore

$$\sum_{|k_1| \leq N} \sum_{l=0}^{j} a_{k_1}^{[l]} a_{k-k_1}^{[j-l]} = 2 \sum_{\frac{k}{2} < k_1 \leq N} \sum_{l=0}^{j} a_{k_1}^{[l]} a_{k-k_1}^{[j-l]} + e(k) \sum_{l=0}^{j} a_{k/2}^{[l]} a_{k/2}^{[j-l]}. \qquad \blacksquare$$

Exploiting the symmetry of the automatic differentiation sum

Lemma 1.4.8. *Let* $e(k) := \begin{cases} 1 & , k \text{ is divisible by } 2 \\ 0 & , \text{otherwise} \end{cases}$. *For* $k \geq 0$ *(for* $k < 0$ *analogously) we have*

$$\sum_{|k_1| \leq N} \sum_{l=0}^{j} a_{k_1}^{[l]} a_{k-k_1}^{[j-l]} = 2 \sum_{l=0}^{\lfloor (j-1)/2 \rfloor} \sum_{|k_1| \leq N} a_{k_1}^{[l]} a_{k-k_1}^{[j-l]} + e(j) \sum_{|k_1| \leq N} a_{k_1}^{[j/2]} a_{k-k_1}^{[j/2]}.$$

Proof:

$$\sum_{|k_1| \leq N} \sum_{l=0}^{j} a_{k_1}^{[l]} a_{k-k_1}^{[j-l]} = \sum_{l=0}^{j} \sum_{|k_1| \leq N} a_{k_1}^{[l]} a_{k-k_1}^{[j-l]} =$$

$$2 \sum_{l=0}^{\lfloor (j-1)/2 \rfloor} \sum_{|k_1| \leq N} a_{k_1}^{[l]} a_{k-k_1}^{[j-l]} + e(j) \sum_{|k_1| \leq N} a_{k_1}^{[j/2]} a_{k-k_1}^{[j/2]},$$

The second equality above is due to the following fact

$$\sum_{|k_1|,|k-k_1|\le N} a_{k_1} b_{k-k_1} = \sum_{|k_1|,|k-k_1|\le N} b_{k_1} a_{k-k_1}. \qquad \blacksquare$$

For any $\{a_k\}_{k=-N}^{N}$ and $\{b_k\}_{k=-N}^{N}$.

Observe that both of the optimizations presented in Section 1.4.4 and Section 1.4.4 cannot be combined, have to be used separately. The optimization has been presented for the direct approach only, but it is straightforward applicable to the FFT approach. Here we do not derive the optimization separately for the FFT approach, the only thing that has to be changed in the analysis presented above is to replace the convolutions of the l_2 coefficients (the sums over k) by the multiplication of the L_2 coefficients.

Grouping of expressions in case of convolution of $n > 2$ terms

Now let us consider the case when the right-hand side of (1.38) contains a convolution of three terms. The following explanation is an illustration to the optimizations that can be performed when the right-hand side of (1.38) contains a convolution of more than two terms

$$\sum_{k_1+k_2+k_3=k} a_{k_1} a_{k_2} a_{k_3}, \quad k = -N, \dots, N.$$

In order to calculate the j-th normalized derivative of F the following sum is to be evaluated

$$\sum_{k_1+k_2+k_3=k} \sum_{l_1+l_2+l_3=j} a_{k_1}^{[j_1]} a_{k_2}^{[j_2]} a_{k_3}^{[j_3]}. \tag{1.43}$$

This sum should be evaluated recursively, i.e. first the auxiliary result is calculated for $r = 0, \dots, j$

$$b_k^{[r]} := \sum_{l=0}^{r} \sum_{|k-k_1|,|k_1|\le N} a_{k_1}^{[l]} a_{k-k_1}^{[r-l]}, \quad k = -2N, \dots, 2N.$$

Observe that k spans from $-2N$ to $2N$, this is required to cover all cases when $k_1 + k_2 + k_3 = k$. And then

$$\sum_{l=0}^{j} \sum_{\substack{|k-k_1|\le N,\\ |k_1|\le 2N}} b_{k_1}^{[l]} a_{k-k_1}^{[j-l]}, \quad k = -N, \dots, N.$$

Remark 1.4.9. Observe that if at this point the values of $b^{[l]}$ for $l = 0, \dots, j-1$ are known (were calculated and stored in the previous steps) the complexity of (1.43) depends linearly on j. First, the $b_k^{[j]}$'s are calculated, and then $\sum_{l=0}^{j} \sum b_{k_1}^{[l]} a_{k-k_1}^{[j-l]}$, which gives only two loops (over j) in total. In general case of $n > 2$ terms the same holds, but more auxiliary results have to be calculated.

Now, let us consider the case when the right-hand side of (1.38) contains a convolution of $n > 2$ terms. In order to calculate the j-th normalized derivative of F the following sum is to be evaluated

$$\sum_{k_1+\cdots+k_n=k} \sum_{l_1+\cdots+l_n=j} a_{k_1}^{[l_1]} \cdots a_{k_n}^{[l_n]}. \tag{1.44}$$

First, let us group the expressions pairwise, and calculate the following auxiliary results

$$\begin{aligned}
a_k^{[r],2} &:= \textstyle\sum_{j_1=0}^{\lfloor (r-1)/2 \rfloor} \sum_{k_1} a_{k_1}^{[j_1]} a_{k-k_1}^{[j-j_1]} + e(r) \sum_{k_1} a_{k_1}^{[r/2]} a_{k-k_1}^{[r/2]} \ , \quad |k| \le N_2 \\
a_k^{[r],4} &:= \textstyle\sum_{j_1=0}^{\lfloor (r-1)/2 \rfloor} \sum_{k_1} a_{k_1}^{[j_1],2} a_{k-k_1}^{[j-j_1],2} + e(r) \sum_{k_1} a_{k_1}^{[r/2],2} a_{k-k_1}^{[r/2],2} \ , \quad |k| \le N_4 \\
a_k^{[r],8} &:= \textstyle\sum_{j_1=0}^{\lfloor (r-1)/2 \rfloor} \sum_{k_1} a_{k_1}^{[j_1],4} a_{k-k_1}^{[j-j_1],4} + e(r) \sum_{k_1} a_{k_1}^{[r/2],4} a_{k-k_1}^{[r/2],4} \ , \quad |k| \le N_8 \\
&\cdots \\
a_k^{[r],2^{max}} &:= \textstyle\sum_{j_1=0}^{\lfloor (r-1)/2 \rfloor} \sum_{k_1} a_{k_1}^{[j_1],2^{max-1}} a_{k-k_1}^{[j-j_1],2^{max-1}} + e(r) \sum_{k_1} a_{k_1}^{[r/2],2^{max-1}} a_{k-k_1}^{[r/2],2^{max-1}}, \ |k| \le N_{2^{max}}
\end{aligned}$$

for all $r = 0, \ldots, j$, or just $r = j$ if for the others $r = 0, \ldots, j-1$ have been calculated and stored in the previous steps the limits $N_2, N_4, \ldots$ should be calculated for particular case such that to cover all cases when $k_1 + \cdots + k_n = k$. Observe that in all of the calculations above the symmetry of the AD sum is used as presented in Section 1.4.4.

Using the binary representation

$$n = b_{max} 2^{max} + b_{max-1} 2^{max-1} + \cdots + b_2 2^2 + b_1 2 + b_0$$

the calculation of (1.44) finally reduces to the (recursive) evaluation of the following sum

$$\sum_{l_1+\cdots+l_{max}=j} \sum_{k_{max}+\cdots+k_1=k} \left(a_{k_{max}}^{[l_{max}],2^{max}} \right)^{b_{max}} \cdot \left(a_{k_{max-1}}^{[l_{max-1}],2^{max-1}} \right)^{b_{max-1}} \cdots \left(a_{k_1}^{[l_1],2} \right)^{b_1} \cdot \left(a_{k_0}^{[l_0],1} \right)^{b_0}. \tag{1.45}$$

Here, for the evaluation of (1.45), none of the optimizations presented in Section 1.4.4 and Section 1.4.4 can be used.

Optimizations for special cases of boundary conditions

We would like to comment on the optimizations that are performed when either of the periodic-odd or periodic-even boundary conditions are imposed. The optimization works for both the direct and the FFT approach. In those cases the sine or the cosine basis is used respectively

$$\{\sin kx\}_{k=1}^{\infty}, \quad \{\cos kx\}_{k=0}^{\infty}. \tag{1.46}$$

This optimization technique relies on exploiting the special structure of the following two by two real matrices

$$\begin{bmatrix} \frac{\partial \operatorname{Re}\left(\varphi_k^{[j]}\right)}{\partial \operatorname{Re}(a_l)} & \frac{\partial \operatorname{Re}\left(\varphi_k^{[j]}\right)}{\partial \operatorname{Im}(a_l)} \\ \frac{\partial \operatorname{Im}\left(\varphi_k^{[j]}\right)}{\partial \operatorname{Re}(a_l)} & \frac{\partial \operatorname{Im}\left(\varphi_k^{[j]}\right)}{\partial \operatorname{Im}(a_l)} \end{bmatrix}, \tag{1.47}$$

In case one of the basis (1.46) is imposed the only nonzero elements are $\frac{\partial \operatorname{Re}\left(\varphi_k^{[j]}\right)}{\partial \operatorname{Re}(a_l)}$ or $\frac{\partial \operatorname{Im}\left(\varphi_k^{[j]}\right)}{\partial \operatorname{Im}(a_l)}$. Still (1.47) are represented in the computer program in their entirety in order to keep the software generic, i.e. to allow the same code handle any kind of the periodic boundary conditions, without any modification of the code or providing different vector fields.

This optimization is based on the observation that when the complex first order jets are passed through the Algorithm 1, the matrices (1.47) take one of the following forms depending on the current stage of the algorithm

- $\begin{bmatrix} * & 0 \\ * & 0 \end{bmatrix}$ or $\begin{bmatrix} 0 & * \\ 0 & * \end{bmatrix}$, where $* \in \mathbb{R}$. After the multiplication by the complex phase shift factors or the complex phase factors ω (from the diagonal matrices, see e.g. matrix $D^2_{m_j} \times I_{l_j}$ from Example 1.2.25) all the purely real or purely imaginary coefficients become complex.

- $\begin{bmatrix} * & 0 \\ 0 & 0 \end{bmatrix}$ or $\begin{bmatrix} 0 & 0 \\ 0 & * \end{bmatrix}$, where $* \in \mathbb{R}$. Jets being the l_2 coefficients of a normalized derivative (denoted by $a^{[j]}(\boldsymbol{\xi})$ in the description of Algorithm 1) of any real-odd or real-even function. When either of the basis (1.46) is used only $\frac{\partial \operatorname{Re}\left(\varphi_k^{[j]}\right)}{\partial \operatorname{Re}(a_l)}$ or $\frac{\partial \operatorname{Im}\left(\varphi_k^{[j]}\right)}{\partial \operatorname{Im}(a_l)}$ can have a nonzero value.

- $\begin{bmatrix} * & * \\ 0 & 0 \end{bmatrix}$, where $* \in \mathbb{R}$. Jets being the L_2 coefficients of a normalized derivative (denoted by $\hat{a}^{[j]}(\boldsymbol{\xi})$ in the description of Algorithm 1) of any real-valued function (not necessarily odd or even). Recall that the L_2 coefficients are representing values at some points of a real-valued function.

The mentioned above knowledge about jets should be used to avoid redundant operations on zero elements, which in case of the rigorous calculations result in an efficiency drop. We have performed several numerical tests, which have indicated that when the interval arithmetic is used the multiplications by zero elements are not suppressed by *compile time optimizations* of *the gcc C++ compiler*. Other tests have indicated that when operations on complex numbers are introduced into the vector field then provided to *FadBad++ library* [BS] in order to perform an automatic differentiation a considerable efficiency drop was observed, which indicates that the FadBad++ library is not automatically performing this kind of optimizations. This is somehow expected, because realizing that such optimizations can be performed requires a global knowledge about the system. Other tests indicated that FadBad++ library tends to break down when a "moderately large" (by "moderately large" we mean ≈ 100 modes) vector field is provided, number of 100 modes is actually a small number when a two dimensional PDE is considered, non-rigorous computations deal with thousands of modes.

All of the code optimizations presented in this section has to be implemented manually. We have included them in the software package [Software] dedicated for rigorous integration of dPDEs. The software package [Software] is not relying on any of the available software libraries for automatic differentiation, only *the CAPD library* [CAPD] is used for the interval arithmetic and the Lohner algorithm. We have implemented the automatic differentiation to the extent required by systems originating from dPDEs with appropriate basis introduced and having a polynomial as the nonlinearity. We are convinced that this is the most appropriate approach to this problem since we are interested in problems related to higher dimensional dPDEs, like the Navier-Stokes equations, which require massive data to be processed, and the control over low-level implementation details is very important.

1.4.5 Comment on Automatic Differentiation Convolutions

Reader would notice that any of the sums of the type (1.41) is in fact a discrete convolution. We call a sum of the type (1.41) the automatic differentiation convolution. We address here the hypothesis stating that the automatic differentiation convolutions could be calculated efficiently using the FFT algorithm. Introducing the FFT here would beat the quadratic complexity with respect to the Taylor method order and reduce further the complexity of the procedure calculating the normalized derivatives, especially when a high order method is used.

Unfortunately a straightforward application of the FFT algorithm is not possible here due to the fact that the normalized derivatives are evaluated recursively. Now, let us illustrate the mentioned problem.

Let us fix $k \in \{0, \ldots, M\}$. Let us treat

$$\hat{a}_k^{[j]}, \quad j = 0, \ldots, q-1$$

as l_2 coefficients and try to calculate (required for $j+1$-th normalized derivative)

$$\sum_{j_1+j_2+\cdots+j_n=j} \hat{a}^{[j_1]}(\boldsymbol{\xi})\, \hat{a}^{[j_2]}(\boldsymbol{\xi}) \cdots \hat{a}^{[j_n]}(\boldsymbol{\xi}) \tag{1.48}$$

for $j = 0, \ldots, q-1$ using the procedure presented in the Section 1.2.1. Observe that the asymptotic complexity of calculating (1.48) directly is at least $O(q^2)$. Calculating (1.48) using the FFT algorithm would reduce the asymptotic complexity to $O(q \log q)$.

First, we emphasize that the procedure presented in the Section 1.2.1 assumes that all sums can be calculated simultaneously. In case of normalized derivatives that would mean that the normalized derivatives of the orders $1, \ldots, q$ can be calculated simultaneously. This is not the case here, because the normalized derivatives are evaluated recursively, and probably have to be evaluated recursively i.e. it is not possible to start calculating the j-th normalized derivative before the calculation of the $0, \ldots, j-1$-th normalized derivatives has been completed.

Second, we emphasize that applying the FFT algorithm to calculate the sum (1.48) just for a fixed j would not be beneficial. The complexity of the evaluation of (1.48) for a fixed j depends linearly with respect to j regardless the number of terms n, as was presented in Section 1.4.4.

1.5 Complexity discussion

The goal of this Section is not to give a thorough analysis of the complexity of the proposed algorithms, which can be a topic of research itself, but to provide an example illustrating the obvious advantages of the FFT algorithm introduced in the current context.

For the purpose of this section we take the following

$$a'_k = \sum_{k_1+k_2=k} a_{k_1} a_{k_2} = F_k, \quad -N \le k \le N. \tag{1.49}$$

as the model equation under investigation, observe that it does not include the linear term at all, this is due to the fact that the linear contribution is negligible in the overall operations count.

1.5.1 Core complexity

The goal of this subsection is to provide a heuristics explaining why we expect Algorithm 1 to be an improvement. We call *the "core" complexity* a complexity without explicitly given constants and without negligible terms. The "core" complexity of calculating the normalized derivatives of (1.49) using Algorithm 1 up to an order q is the following

$$\overbrace{c_1 q N \log N}^{\text{the FFT's}} + \overbrace{c_2 q(q+1) N}^{\text{multiplications of the } L_2 \text{ coefficients in FFT's}} . \tag{1.50}$$

Whereas the direct approach yields the following "core" complexity

$$\overbrace{c_3 q(q+1) N^2}^{\text{"double" convolution}} . \tag{1.51}$$

The factor $q(q+1)$ originates from the fact that in order to calculate the j-th normalized derivative j sums have to be evaluated, and summing everything up gives $1 + 2 + \cdots + q = \frac{1}{2} q(q+1)$, compare (1.40) and (1.41).

Where $q \ge 1$ is the Taylor method order.

The equation (1.50) contains a term which is quadratic only with respect to the order q. The equation (1.51) contains a term which is "double" quadratic, i.e. quadratic with respect to the order q and the dimension N.

1.5.2 Explicit operations count

Observe that the "core" complexity of Algorithm 1 is given in terms of a number of operations on jets. In this section we address the question what is the number of elementary operations required by Algorithm 1 applied to the particular case of the system (1.49). The calculations does not take into account the linear contribution, but the cost of the linear contribution is negligible compared to the nonlinear contribution and moreover is the same for both of the algorithms. M and N have been chosen to satisfy $M = 2^p$, and $M = 3N + 1$ (the smallest value of M for which Technique I (padding) from Section 1.3.1 eliminates the aliasing error)

Number of elementary operations for calculating the $0 < j$-th normalized derivative using the FFT approach

Definition 1.5.1. *We call* the elementary operation *either the addition of two complex numbers* (the complex addition) *or the multiplication of two complex numbers* (the complex multiplication).

We assume that the normalized derivatives for orders $0, \ldots, j-1$ have already been calculated.

FFT Cost of the FFT transform (either $l_2 \to L_2$ or $L_2 \to l_2$) is Mp Jet additions and $\frac{M}{2}p$ Jet-Scalar multiplications. See Example 1.2.24, observe that the given cost is valid only for $M = 2^p$, when M contains other factors the cost may differ by a constant. The cost is given for either FFT-B or FFT-C variant, for which the permutation step is not considered. See Section 1.2.7 for an argument that the number of Jet additions and Jet-Scalar multiplications for each $j \in \{1, \ldots, q\}$ is a linear function of M, and especially for the case $M = 2^p$ half of the Jet-Scalar multiplications are the multiplications by 1.

L_2 coefficients multiplication In fact Step 3 in the outline presented in Section 1.2.1. Calculated Cost of the multiplication of all the L_2 coefficients is $M\lceil \frac{j}{2} \rceil$ Jet multiplications and $M\left(\lceil \frac{j}{2} \rceil - 1\right)$ Jet additions and M Jet-Scalar multiplications (related to the multiplication by 2). The given cost takes into account the second optimization presented in Section 1.4.4.

Multiplication by $1/j$ Requires $2N + 1$ (the number of modes) Jet-Scalar multiplications.

Total cost The total cost of calculating the j-th order normalized derivative using the FFT algorithm is $2Mp + M\left(\lceil \frac{j}{2} \rceil - 1\right)$ Jet additions and $M\lceil \frac{j}{2} \rceil$ Jet multiplications and $Mp + M + 2N + 1$ Jet-Scalar multiplications.

Observe that the computational costs given above are expressed in terms of operations on the first order jets, see Section 1.4.2. Now we provide the explicit number of the elementary operations required to perform the mentioned operations on jets.

Let $a(\boldsymbol{\xi}), b(\boldsymbol{\xi})$ be "complex" first order jets (1.39).

Jet addition requires $4N+3$ complex additions. Refer (1.39) and Definition 1.4.3. The operation $\check{a}+\check{b}$ requires 1 complex addition. The operations $\{a_j+b_j\}_{-N}^{N}$ requires $2(2N+1)$ complex additions in total, because number of modes is $2N+1$ and the partial derivative has two complex components: partial derivative with respect to the real part and the partial derivative with respect to the imaginary part.

Jet-Scalar multiplication requires $4N+3$ complex multiplications. Refer (1.39). Let $s \in \mathbb{C}$ be a scalar, $s \cdot b(\boldsymbol{\xi}) = s \cdot \check{b} + \sum_{-N}^{N} s \cdot b_j$. The operation $s \cdot \check{b}$ requires 1 complex multiplication. The operations $\{s \cdot b_j\}_{-N}^{N}$ requires $2(2N+1)$ complex multiplications in total, because number of modes is $2N+1$ and the partial derivative has two complex components: partial derivative with respect to the real part and the partial derivative with respect to the imaginary part.

Jet multiplication requires $8N+5$ complex multiplications and $4N+2$ complex additions. Refer (1.39) and Definition 1.4.3. The operation $\check{a} \cdot \check{b}$ requires 1 complex multiplication. The operations $\left\{\check{a} \cdot b_j + \check{b} \cdot a_j\right\}$ requires $2 \cdot 2(2N+1)$ complex multiplications and $2(2N+1)$ complex additions in total, because the number of modes is $2N+1$ and the partial derivative has two complex components: partial derivative with respect to the real part and the partial derivative with respect to the imaginary part.

The total number of elementary operations required for calculating the j-th normalized derivative using the FFT algorithm is

- **complex additions** $\left[2Mp + M\left(\lceil \frac{j}{2} \rceil - 1\right)\right](4N+3) + M\lceil \frac{j}{2} \rceil (4N+2)$
- **complex multiplications** $M\lceil \frac{j}{2} \rceil (8N+5) + (Mp+M+2N+1)(4N+3)$

Number of elementary operations for calculating the $0 < j$-th normalized derivative using the direct approach

The total number of operations needed to evaluate the following expression depends on k and is expressed in the following list

$$N_k^{[j-1]}(\boldsymbol{\xi}) := 2 \sum_{k_1=\lfloor \frac{k}{2} \rfloor + 1}^{N} \sum_{r=0}^{j-1} a_{k_1}^{[r]} a_{k-k_1}^{[j-r]} + e(k) \sum_{r=0}^{j-1} a_{k/2}^{[r]} a_{k/2}^{[j-r]}, \qquad k = 0, \ldots, N. \tag{1.52}$$

- $k = 0$: Nj Jet multiplications + $Nj - 1$ Jet additions + j Jet multiplications + j Jet additions + 1 Jet-Scalar multiplication.
- $k = 1$: Nj Jet multiplications + $Nj - 1$ Jet additions + 1 Jet-Scalar multiplication.
- $k = 2$: $(N-1)j$ Jet multiplications + $(N-1)j - 1$ Jet additions + j Jet multiplications + j Jet additions + 1 Jet-Scalar multiplication.
- $k = 3$: $(N-1)j$ Jet multiplications + $(N-1)j - 1$ Jet additions + 1 Jet-Scalar multiplication.
- ...
- $k = N-1$: $\lceil \frac{N+1}{2} \rceil j$ Jet multiplications + $\lceil \frac{N+1}{2} \rceil j - 1$ Jet additions + j Jet multiplications + j Jet additions + 1 Jet-Scalar multiplication.
- $k = N$: $\lfloor \frac{N+1}{2} \rfloor j$ Jet multiplications + $\lfloor \frac{N+1}{2} \rfloor j - 1$ Jet additions + 1 Jet-Scalar multiplication.

The case $-N \leq k < 0$ is analogous.

Total cost of direct evaluation of the j-th normalized derivative

- **Jet multiplications** $\left[\frac{3}{2}N^2 - N + O(1) + 2\lfloor \frac{N}{2} \rfloor\right] j$,
- **Jet additions** $\left[\frac{3}{2}N^2 - N + O(1) + 2\lfloor \frac{N}{2} \rfloor\right] j - 2N$,
- **Jet-Scalar multiplications** $2(2N+1)$.

The term $2\lfloor \frac{N}{2} \rfloor$ contains the extra additions and multiplications in even cases, related to the term $e(k)a_{k/2}^2$. The Jet-Scalar multiplications are related to the multiplication by 2 and the multiplication by $1/j$.

The total number of elementary operations required for direct evaluation of the j-th order normalized derivative is as follows. We drop the $O(j)$ term, because of its irrelevance.

- **complex additions**

$$\left[\left(\frac{3}{2}N^2 - N + 2\left\lfloor \frac{N}{2} \right\rfloor\right)(4N+3) + \left(\frac{3}{2}N^2 - N + 2\left\lfloor \frac{N}{2} \right\rfloor\right)(4N+2)\right] j - 2N(4N+3),$$

- **complex multiplications**

$$\left[\left(\frac{3}{2}N^2 - N + 2\left\lfloor \frac{N}{2} \right\rfloor\right)(8N+5)\right] j + 2\,(2N+1)\,(4N+3)\,.$$

N	M	FFT	Direct
5	16	19 360	23 489
10	32	86 400	186 950
20	64	387 200	1 468 400
40	128	1 730 560	11 634 800

Figure 1.13: Number of complex additions required to calculate the normalized derivatives of the orders $\{1, 2, 3, 4, 5\}$

N	M	FFT	Direct
5	16	16 945	27 169
10	32	70 275	200 280
20	64	297 975	1 519 030
40	128	1 274 975	11 832 030

Figure 1.14: Number of complex multiplications required to calculate the normalized derivatives of the orders $\{1, 2, 3, 4, 5\}$

Remark 1.5.2. For real-valued odd and real-valued even solutions the number of elementary operations is different, the advantage of the FFT algorithm is reduced. This is related to the fact that the FFT algorithm operates only on complex numbers, whereas the system of equations can be written using real coefficients in those cases, e.g. see [ZM]. Still, the FFT approach proved to be faster, see the results presented in Section 1.6.

1.6 Rigorous numeric tests

We present below a report from various tests we have performed with the developed methods and algorithms. We stress that by tests we mean rigorous numerics tests, and in particular what is presented here is not in any case a computer assisted proof. Still, the purpose of the developed methods and algorithms are the computer assisted proofs for dPDEs.

In the tests either a finite truncation (a Galerkin projection) or the full infinite system (giving a differential inclusion) was considered. We have specified this for each test separately by stating *"Galerkin projection"* or *"differential inclusion"* respectively. In order to integrate the full infinite system several additional steps have to be performed, the Lohner algorithm for differential inclusions is used, refer Section 2.9.1 or [Z3].

In this section we present results from all the tests performed in the form of tables. Whenever a diameter of a finite or infinite dimensional set is provided in a table, the meaning of this number is in fact the maximum over coordinates of all diameters. The labels *"Direct approach"*, *"FFT approach"* and *"FadBad++"* appearing in the tables indicate different implementations of the Lohner algo-

rithm (or the Lohner algorithm for differential inclusions) that has been used. "Direct approach" indicates that the normalized derivatives has been obtained by the direct approach, whereas "FFT approach" indicates that the normalized derivatives has been obtained by the FFT approach presented previously, to remove the aliasing error technique "II First technique based on phase shifts" was used, unless otherwise stated. "FadBad++" indicates that the normalized derivatives has been obtained by using the FadBad++ software library for automatic differentiation [BS], without using the FFT at any point, the vector field provided as the input to the FadBad++ library was written using not complex but real coefficients, whenever odd or even boundary conditions were used, and properly optimized. The part of the Lohner algorithm regarding the Lohner representation of the sets was the same for both. For the thorough description of the Lohner algorithm and the issues related to the wrapping effect and representation of sets refer [ZLo].

The meaning of the symbol M depends on the context, in Chapter 1 it is used to denote the number of points of the discrete grid used by the FFT, and in Chapter 2 it is used to denote the dimension of the so-called finite tail, when the full infinite dimensional system is considered. To avoid the ambiguities in this section we denote the number of points of the discrete grid, used by the FFT, by M_{FFT}.

Burgers equation fixed point test (Galerkin projection) When periodic boundary conditions are used and the forcing function is not present the Burgers equation exhibits a globally attracting fixed point at zero. This is proven by the Energy exponential decay, see Section 2.3.1.

With this test we address the question what is, for the overall Lohner algorithm, the consequence of the overestimates produced by the FFT approach which are considerable, see the results from Section 1.3.2. This question is hard to answer without performing actual calculations due to the sophisticated nature of the Lohner type rigorous integrator.

The setting

$$\begin{aligned} &u_t + u \cdot u_x - \nu u_{xx} = 0, \\ &u(x,0) = u_0(x), \quad x \in \mathbb{R}, \\ &u(x,t) = u(x + 2k\pi, t), \quad x \in \mathbb{R},\ t \in [0, \mathcal{T}),\ k \in \mathbb{Z}, \end{aligned}$$

The uniform zero-centered box (with the fixed point at the center) was rigorously integrated forward in time on the fixed time interval with the fixed time step. The diameter of the initial box was the parameter of the tests. The variant of the FFT algorithm used was Algorithm FFT-B - FFT-C ($l_2 \to L_2$ by FFT-C without the permutation step, and then $L_2 \to l_2$ by FFT-B without the permutation step), the aliasing error was removed by using the Technique I - padding. In figure 1.16 we present a report from the performed tests. The diameter of the result was measured, which in fact was the diameter of the set

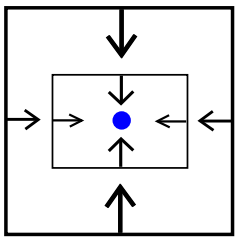

Figure 1.15: The setting for the Burgers equation

at the time 10 (1000 constant time steps of 0.001 have been executed). Due to the fact that the fixed point is attracting, the initial box is expected to decrease in diameter, specifically we have been interested if the diameter of the result is considerably larger when the FFT approach is used than when the direct approach is used. This kind of behavior is predicted by the results presented in Section 1.3.2.

Parameters that were used for the tests were $\nu = 0.1$, order of the Taylor method was 6, $N = 19$, $M_{FFT} = 40$.

#	Initial condition diam.	Set at the end diam.	
		direct approach	FFT approach
0	$1e-08$	$9.04837e-09$	$9.04838e-09$
1	$1e-07$	$9.04838e-08$	$9.0484e-08$
2	$1e-06$	$9.0484e-07$	$9.04867e-07$
3	$1e-05$	$9.04862e-06$	$9.05131e-06$
4	0.0001	$9.05088e-05$	$9.0789e-05$
5	0.001	0.000907367	0.000958336
6	0.01	0.00932532	0.00932849*

Total running time	
FadBad++	FFT approach
1849 sec	424 sec

(1.53)

Figure 1.16: Fixed point for Burgers equation test results

* in this test a blow-up of the set was experienced for the FFT approach. By a blow-up we mean that the rigorous bounds escaped the range of the representable numbers, such bounds are then represented in the software by the unbounded interval ($[-\infty, \infty]$). As soon as the unbounded interval appear in the calculation process the further calculations are meaningless. We have found a simple method circumventing this problem by a little additional computational cost. This issue is solved when the first order normalized derivative and only the first order normalized derivative (a one convolution) is calculated directly,

and the higher normalized derivatives are calculated using the FFT approach. The cost of calculating the first order normalized derivative is negligible for the overall cost of calculating the normalized derivatives up to a reasonable fixed order.

Kuramoto-Shivasinsky equation attracting periodic orbit test

The setting

$$
\begin{aligned}
&u_t = -\nu u_{xxxx} - u_{xx} + (u^2)_x, \\
&u(x,0) = u_0(x), \quad x \in \mathbb{R}, \\
&u(x,t) = -u(-x,t), \quad u(x,t) = u(x + 2k\pi, t), \quad x \in \mathbb{R},\ t \in [0, \mathcal{T}),\ k \in \mathbb{Z},
\end{aligned}
$$

One of the periodic orbits proven to exists in [Z3] has been picked (the attracting periodic orbit for $\nu = 0.032$). We have taken a box around an initial condition from [Z3] for this periodic orbit and then the box has been rigorously propagated forward in time using two implementations of the Lohner algorithm. In the tests the equations were integrated until the time of the full revolution of the orbit was reached, which is approximately 0.4091. The results of our tests are contained in Figures 1.17, 1.18 and 1.19.

I test - efficiency (Galerkin projection) All of the techniques presented previously in this part of the dissertation can be replaced by applying any available software library for automatic differentiation, for instance FadBad++ [BS]. With this tests we would like to provide an argument that we have not done a redundant work developing this approach. The Lohner algorithm implementation based upon the FFT approach and the optimizations discussed in this dissertation is apparently more efficient.

With this test we compared the efficiency of the Lohner algorithm implementation based upon the presented FFT approach with the implementation based on the direct approach, and using the external software library for automatic differentiation FadBad++ [BS] (the approach used to perform the computer assisted proof presented in Chapter 2).

A Galerkin projection of the infinite dimensional system was considered in this tests in order to isolate this part of the overall algorithm for which the developed techniques contribute. The initial condition was taken from [Z3], the diameter of the initial condition was $8e-05$ in each direction (the same as in [Z3]).

Observe that the time step had to be adjusted for each test, because of the stiffness problem exhibited by the Kuramoto-Shivasinsky equation.

Observe that in this case the algorithm based on the FFT approach appears to be less efficient than in the case of the setting used to produce the result in the table (1.53) (the Burgers equation with pure periodic bd. cond.). This is related to the fact that the FFT algorithm is necessary complex, and thus it is more beneficial when modes are complex (when the pure periodic boundary conditions

	Parameters used				Total running time	
#	Galerkin proj. dim.	M_{FFT}	Time step	Taylor m. ord.	FadBad++	**FFT approach**
1	23	48	10^{-4}	5	139 sec	**107 sec**
2	23	48	10^{-4}	8	297 sec	**175 sec**
3	23	48	10^{-4}	10	438 sec	**229 sec**
4	28	60	5^{-5}	5	496 sec	**343 sec**
5	34	72	2^{-5}	5	2147 sec	**1295 sec**

Figure 1.17: Total running time of programs propagating a box along orbit for the KS equation. The machine used was Linux 32-bit Intel Core i5-2430M CPU @ 2.40GHz x 4.

are used modes are complex, when either the periodic-odd or the periodic-even bd. cond. are used modes are purely imaginary or purely real resp.). The FFT algorithm is expected to be the most beneficial in a setting in which complex-valued solutions are looked for, this is motivated by the elementary operations count provided in Figure 1.13 and Figure 1.14. The results of this test are contained in Fig. 1.17.

II test - overestimates (Galerkin projection) The diameter of the the propagated set was measured after the full revolution along the orbit. For this test a finite Galerkin projection of the infinite dimensional system was used. Parameters that were used for the tests were $\nu = 0.032$, order of the Taylor method was 5, $N = 23$, $M_{FFT} = 48$. The results of this test are contained in Fig. 1.18.

#	Initial condition diam.	Set at the end diam.	
		Direct approach	FFT approach
0	$1e-08$	$7.7879e-05$	0.00012947
1	$1e-07$	0.000122268	0.000130407
2	$1e-06$	$9.39151e-05$	0.000144392
3	$1e-05$	0.00026486	0.000235396*
4	0.0001	0.00315496	0.00408689*

Figure 1.18: Diameter (max over coordinates) of boxes obtained by programs propagating a box along orbit for KS equation. A finite truncation (a Galerkin projection) was considered.

* in this test a blow-up of the set was experienced for the FFT approach. By a blow-up we mean that the rigorous bounds escaped the range of the representable numbers, such bounds are then represented in the software by the unbounded interval ($[-\infty, \infty]$). As soon as the unbounded interval appear in the calculation process the further calculations are meaningless.We have found a simple method circumventing this problem by a little additional computational cost. This issue is solved when the first order normalized derivative and only the first order normalized derivative (a one convolution) is calculated directly, and the higher normalized derivatives are calculated using the FFT approach. The cost of calculating the first order normalized derivative is negligible for the overall cost of calculating the normalized derivatives up to a reasonable fixed order.

III test - overestimates (differential inclusion) The diameter of the the propagated set was measured after the full revolution along the orbit. The full rigorous integrator (the Lohner algorithm for differential inclusions) of the infinite dimensional system was used in the test.

Parameters that have been used for the tests were as follows $\nu = 0.032$, order of the Taylor method was 5, $N = 23$, $M = 92$ (the dimension of the finite tail, for the exact meaning of this number refer Section 2.5). The results of this test are contained in Fig. 1.19.

Swift-Hohenberg equation a connection between fixed points test (differential inclusion)

#	Initial condition diam.	Set at the end diam.	
		Direct approach	FFT approach
0	$1e-08$	$7.7879e-05$	0.00012947
1	$1e-07$	$9.55487e-05$	0.000134767
2	$1e-06$	0.000174207	0.00025008
3	$1e-05$	0.000231317	0.00309667
4	0.0001	0.00134516	0.00221152*

Figure 1.19: Diameter (max over coordinates) of boxes obtained by programs propagating a box along orbit for KS equation. A the full infinite dimensional system was considered (differential inclusion).

The setting

$$u_t = \left(\nu - \left(1 + \frac{\partial^2}{\partial x^2}\right)^2\right) u - u^3, \tag{1.54}$$
$$u(x,0) = u_0(x), \quad x \in \mathbb{R},$$
$$u(x,t) = u(-x,t), \quad u(x,t) = u(x+2k\pi,t), \quad x \in \mathbb{R},\ t \in [0,\mathcal{T}),\ k \in \mathbb{Z},$$

Looking at the eigenvalues of the linear part of (1.54)

$$\lambda_k = \nu - (1-k^2)^2$$

it is immediately verified that zero is an unstable fixed point for (1.54), for a sufficiently large ν.

overestimates (differential inclusion) As the initial condition we took a small ball of functions close to zero, i.e.

$$u_0 = 2 \cdot \left(10^{-10} + [-10^{-11}, 10^{-11}]\right) + 2 \cdot \left(10^{-10} + [-10^{-11}, 10^{-11}]\right) \cos x.$$

This set have been rigorously integrated until it was being caught in the basin of attraction of an another fixed point for (1.54), being a attracting fixed point. The process of integration of the set was stopped as soon as the set has stabilized at an attracting fixed point.

Order of the Taylor method that was used for the tests was 5. The results of this test are contained in Fig. 1.21.

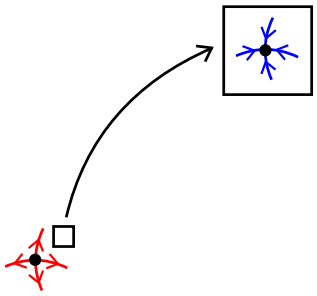

Figure 1.20: The setting for the Swift-Hohenberg equation

#	Parameters used					Set at end diam.	
	ν	Number of steps	Time step	N	The finite tail dim.	Direct approach	FFT approach
0	25	15000	0.0001	11	33	$2.31318e-05$	$4.62783e-05^{*}$
1	50	10000	0.0001	11	33	$1.28577e-06$	$3.19921e-05^{*}$
2	65	6000	0.0001	12	50	0.000153006	0.00339236^{**}

Figure 1.21: Diameter (max over coordinates) of boxes obtained by programs propagating a box along connection between fixed points for SH equation

* in this test a blow-up of the set was experienced for the FFT approach. By a blow-up we mean that the rigorous bounds escaped the range of the representable numbers, such bounds are then represented in the software by the unbounded interval ($[-\infty, \infty]$). As soon as the unbounded interval appear in the calculation process the further calculations are meaningless. We have found a simple method circumventing this problem by a little additional computational cost. This issue is solved when the first order normalized derivative and only the first order normalized derivative (a one convolution) is calculated directly, and the higher normalized derivatives are calculated using the FFT approach. The cost of calculating the first order normalized derivative is negligible for the overall cost of calculating the normalized derivatives up to a reasonable fixed order.

** in this test a blow-up was experienced, it was not eliminated when only the technique * was used, like in other cases, it was eliminated when furthermore instead of the phase-shift aliasing error elimination technique the padding technique was used.

Numerical data from all the performed test is available on-line [Software].

Concluding remarks from performed tests

Now, we would like to present some concluding remarks from the results presented above

- **The diameters of the sets obtained in the tests, see Figure 1.16, 1.18, 1.19 and 1.21 indicates that, surprisingly, the Lohner algorithm is immune to the considerable overestimates produced by the FFT approach. Consequently we are convinced that the Lohner algorithm based on the here developed FFT approach is the most adequate approach to the problem of rigorous integration of higher dimensional PDEs, including the Navier-Stokes equations.**

- **The different test cases have demonstrated that the software based on the developed techniques has improved the efficiency of the rigorous integration task, without severe consequences. All the test cases were successfully finished by the new approach (no blow-ups were experienced), and moreover considerable quality drop (measured by diameter of the propagated sets) have not been observed in most of the cases. We consider this as a significant achievement in the context of eventual computer assisted proofs for higher dimensional PDEs. The task of rigorous integration of higher dimensional PDEs like the Navier-Stokes equation, which is our goal, will require a large amount of computing time. Any reduction of the time spend on computations is of great importance here, because this could allow some things to be actually proven.**

- Different equation and different boundary conditions were used in each of the tests (the Burgers equation with pure periodic bd.cond., the KS equation with periodic-odd bd.cond. and the SH equation with periodic-even bd.cond.) in order to demonstrate that the developed software is generic and robust. The software can be applied for many problems involving dPDEs, for a description of the implementation details refer Section 1.7.

- The goal of achieving both generic and efficient software for rigorous integration of dPDEs has been achieved in some extent. This is a hard task in general. We have been able of achieving this for a periodic boundary conditions in 1D. Still such a software did not existed, and all the results published so far (e.g. [AK], [Z3]) focused on the particular case of the Kuramoto-Shivasinsky equation.

- Regarding the quality of results, sometimes the usage of the FFT approach resulted in blow-ups, which have blocked the further calculations. The tests in which a blow-up was experienced has been marked by *. We have found a simple method circumventing this problem by a little

additional computational cost. This issue is solved when the first order normalized derivative and only the first order normalized derivative (a one convolution) is calculated directly, and the higher normalized derivatives are calculated using the FFT approach. The cost of calculating the first order normalized derivative is negligible for the overall cost of calculating the normalized derivatives up to a reasonable fixed order.

The tests performed in this section are in principle repeatable, files with data from the presented proofs and all the source code used are available on-line [Software].

1.7 Notions on Software

In this section we reveal some of the implementation details of the developed software package. It has been written in the C++ programming language. The software in principle allows any dPDE having as nonlinearity a second order polynomial or a third order polynomial to be rigorously integrated. The source codes package has been published on-line [Software]. Here we provide only an overview of the used ideas to show that the software is in fact generic, for the details refer the source codes.

Definition of an equation is provided using a template class, which inherits from `capd::jaco::DPDEDefinition<PolyBdT>` for instance the following

```
template<class PolyBdT>
class Burgers : public PolyBdT::SubspaceType, public capd::jaco::DPDEDefinition<PolyBdT>{
public:
  RealType nu;
  int m_p; ///<the maximal order of the derivatives in the Laplacian
  int m_d; ///<dimension of the domain (currently only 1 is supported)
  int m_r; ///<order of the derivative in the nonlinear part
  int m_sufficientlyLarge; ///<stores the value equal to m_d+m_p+1
  ComplexType m_N_coeff; ///<constant value in front of the nonlinear part
  static const double S_DISSIPATIVE = -1; ///<used by the rough enclosure procedure
  Burgers(RealType nu_); ///<a constructor
  //Eigenvalues of a dPDE satisfies \lambda_k=-\nu(|k|)|k|^p
  RealType ni(int k) const; ///<returns \nu(|k|), where k is an index in the array
                            ///<storing modes
  RealType ni(IndexType& i) const; ///<returns \nu(|k|), where i is the index of a mode
  bool isDissipative(int k); ///<used by the rough enclosure procedure
  RealType lambda(int k) const; ///<returns an eigenvalue of the linear part, where k is
                                ///<an index in the array storing modes
  RealType lambda_k(int i) const; ///<returns an eigenvalue of the linear part, where i
                                  ///<is the index of a mode
  RealType lambda_k(IndexType& i) const; ///<returns an eigenvalue of the linear part,
                                         ///<where i is the index of a mode
  RealType V(int K) const; ///<function returns V(K)=\{ \inf{v(|k|} | |k|>=K} \}, see
                           ///<definition of dissipative PDE in the paper.
  RealType V(IndexType& i) const; ///<function returns V(K)=\{ \inf{v(|k|} | |k|>=K} \},
                                  ///<see definition of dissipative PDE in the paper.

  /**returns smallest integer K larger than k, such that f(k) is monotonously non
```

```
     * increasing function for i>K. Where f(k) = e^{h\lambda_k} k^r */
    int maximumPoint(const RealType& h, int r, int k) const;
  };
```

Then this class is provided as a template parameter to either of the `class DPDE2` (a second degree polynomial in the nonlinear part) or `class DPDE3` (a third degree polynomial in the nonlinear part), refer the sources with examples for the details.

The boundary conditions are defined during the obligatory initialization of the first order jets

```
capd::jaco::DPDEContainer container;
//2.set here the subspace of the initial condition e.g. setToRealValuedOdd means that
container.setToRealValuedOdd(); ///<the initial condition is real valued odd
FirstOrderJet1D::initialize(ModesContainer1D::modes2arraySizeStatic(m), container);
```

After this step the software is automatically performing optimizations for the provided type of the boundary conditions.

First order jets (`class FirstOrderJet`) are in fact structures composed of a complex number and a vector composed of `class Pair` objects, where an instance of `class Pair` is a pair of complex numbers representing partial derivatives with respect to the real part and the imaginary part, see (1.39). The operators $*$ and $+$ has been implemented like in Definition 1.4.3 using the operator overloading technique.

```
template< class DerivativeT, int D>
class FirstOrderJet : public capd::jaco::DPDEContainer{
public:
  static int dim;
  ScalarType val; ///< the base part
  capd::vectalg::Vector<DerivativeT, D> ksi; ///< first order coefficients (an array),
                                             ///see Definition 4.7
  static FirstOrderJet* buffer;

  friend const FirstOrderJet& operator*(const FirstOrderJet& foj1, const FirstOrderJet& foj2){
    (DPDEContainer&)*buffer = (DPDEContainer&)foj1 * (DPDEContainer&)foj2;
    int i;
    for(i=0; i < dim; ++i){
      //the following line is in fact optimized using DPDEContainer
      buffer->ksi[i] = foj1.val * foj2.ksi[i] + foj2.val * foj1.ksi[i];
    }
    buffer->val = foj1->val * foj2->val;
  }
};
```

The optimization presented in Section 1.4.4 has been implemented by introducing `class DPDEContainer`, which is a base class for `class FirstOrderJet`, and its purpose is to keep the information about zero elements of each jet. The operators $*$ and $+$ has been implemented for `class DPDEContainer` in order to propagate the information about zero elements of jets which are results of arithmetic operations.

```
class DPDEContainer{
public:
  int subspaceType;
  int solutionType;
  int solutionType2;
  bool baseReZero; ///<if the real part of the base is zero
  bool baseImZero; ///<if the imag part of the base is zero
  bool partialReReZero; ///<if the real part of partial derivative with resp to Re is zero
  bool partialReImZero; ///<if the imag part of partial derivative with resp to Re is zero
  bool partialImReZero; ///<if the real part of partial derivative with resp to Im is zero
  bool partialImImZero; ///<if the imag part of partial derivative with resp to Im is zero
};
```

There are several possible ways of implementing the automatic differentation in C++. In ours implementation results of an arithmetic operations on instances of `class FirstOrderJet` are being accumulated in a dedicated buffer in order to avoid surplus construction of instances of `class FirstOrderJet`, which may be large objects requiring a considerable time to create. We are aware that this implementation is not the optimal one, and is limited (obviously one buffer sometimes is not enough), there exists implementations exploiting the full potential of the automatic code optimizations performed by the gcc C++ compiler, refer e.g. [PP]. In the context of the execution time comparison given as I test - efficiency (Galerkin projection) in Section 1.6, we claim that when the automatic differentiation procedure in [Software] is further optimized (e.g. using the techniques presented in [PP], using grouping of expressions in cases when more jets are multiplied) then the efficiency benefit of using the presented software package will even be greater. Still with the current implementation of the automatic differentiation the efficiency benefit is already visible, compare the efficiency tests in Section 1.6.

We have not mentioned here all programing techniques we have used. Our software relies heavily on using *the C++ template classes*. The information about other classes we have implemented can be found in the source code comments of [Package].

Chapter 2

Existence of globally attracting fixed points of viscous Burgers equation with constant forcing, a computer assisted proof

2.1 Introduction

The field of computer assisted proofs for ordinary differential equations (ODEs) is a quite well established and analyzed topic. It seems to us that the development of methods for investigating the dynamics of PDEs by performing rigorous computer assisted proofs is at pioneering stage.

Up to our knowledge, there exists two such methods which are dedicated for a nonstationary PDE problem. The method of self-consistent bounds, presented in the series of papers [ZM], [Z2], [Z3], [ZAKS] and the method presented in [AK]. Both of them have been applied to the Kuramoto-Sivashinsky equation.

In the present paper we develop a computer aided method which is interesting by two main reasons. First is that it provides not only local, but global perspective on the dynamics. Second, it allows to establish results which have not been achieved using known analytical techniques. As a case study we present the forced viscous Burgers equation, where the forcing is constant in time and periodic in space. More specifically, we consider initial value problem with periodic boundary conditions for the equation

$$u_t + u \cdot u_x - \nu u_{xx} = f(x). \tag{2.1}$$

The case of the viscous Burgers equation with zero forcing has already served as an illustration of a computer aided technique in [FTKS]. In the present paper

we deal with the case of nonzero forcing, which is not anymore reductible to a linear PDE by the Hopf-Cole transform.

It has been shown that (2.1) belongs to the class of dissipative PDEs (dPDEs) possessing inertial manifolds [V]. Using our technique we have shown that the global attractor exhibited by (2.1) is in fact a unique stable fixed point. To establish the existence of an attracting fixed point locally, we use the computer techniques from [ZAKS]. We construct a small neighbourhood of a candidate for the fixed point and prove the existence and uniqueness of a fixed point within the neighbourhood by calculating an explicit upper bound for the logarithmic norm. In case the logarithmic norm is negative, we claim that there exists a locally attracting fixed point. On the other hand we show the fact of global existence of solutions by constructing trapping regions, inspired by the analogical sets constructed for the Navier-Stokes equations [MS], [ES], see also [ZNS].

We link those results by constructing an absorbing set, which captures any initial condition after a finite time. Then we rigorously integrate the obtained absorbing set forward in time until it is mapped into a small region with established existence of an attracting fixed point within. By doing so, we verify that any initial condition is in the basin of attraction of the fixed point. To perform a rigorous integration of the viscous Burgers equation forward in time, we use an algorithm based on the one from [Z3]. However, to make it more efficient and applicable to other equations, we propose a modified algorithm that deals with tails of inhomogeneous dimensions and adaptively finds a proper dimension. A description of the algorithm is also provided. The aforementioned elements all together give an original technique that allows to extend the property of attractiveness obtained locally on a small region to a global fact. We would like to stress that our method concerns the evolution in time of dPDEs, not only the stationary problem. Moreover, it is worth pointing out that we do not lay any restrictions in terms of integral conditions like $\int_Q u(t,x)\,dx = 0$ on a domain Q, that have been extensively used in previous works.

An example result obtained with the presented method is the following

Theorem 2.1.1. *For any* $\nu \in [2, 2.1]$ *and* $f \in 1.6\cos 2x - 2\sin 3x + \sum_{k=1}^{3} [-0.03, 0.03]\sin kx + [-0.03, 0.03]\cos kx$ *there exists a fixed point - steady state solution of* (2.1)*, which is unique and globally attracting any initial data* u_0 *satisfying* $u_0 \in C^4$ *and* $\int_0^{2\pi} u_0(x)\,dx = 2\pi$.

Other examples are given in Section 2.7. The expression $[-a, a]\sin kx$ denotes a set of functions $\{c \cdot \sin kx \mid c \in [-a, a]\}$. The function $1.6\cos 2x - 2\sin 3x$, added to the forcing, has been chosen as an example to show that our method is not limited to the simpler case of low energy forcings. Note that Theorem 2.1.1 covers a whole set of forcing functions within a "ball" $\sum_{k=1}^{3} [-0.03, 0.03]\sin kx + [-0.03, 0.03]\cos kx$. To achieve this we have used the interval arithmetic in a way to be explained later.

As this paper is dependent on [Z3] and [ZAKS], we recall only crucial definitions and results from the previous works and focus on the new elements. Proper references are always provided whenever necessary. We are convinced

that the presented techniques are applicable to higher dimensional dPDEs, including the Navier-Stokes equations, and we will address this problem in our forthcoming papers.

We organize the paper in the following way, the first part comprises the theory, it is concluded by the proof of Theorem 2.1.1 in Section 2.7. A presentation and discussions of the algorithms follows.

2.2 The viscous Burgers equation

As the viscous Burgers equation we consider following PDE

$$\frac{\partial u}{\partial t} + u \cdot \frac{\partial u}{\partial x} - \nu \triangle u = 0 \quad \text{in } \Omega, \quad t > 0,$$

where ν is a positive *viscosity constant*. The equation was proposed by Burgers (1948) as a mathematical model of the turbulence. Later on it has been successfully showed that the Burgers equation models certain gas dynamics (Lighthill (1956)) and acoustic (Blackstock (1966)) phenomena, see [CHQZ] and [Wh]. For our purposes we define this equation on the real line $\Omega := \mathbb{R}$, add *a constant in time forcing* f to the right-hand side and consider initial value problem with periodic boundary conditions

$$u \colon \mathbb{R} \times [0, \mathcal{T}) \to \mathbb{R},$$

$$f \colon \mathbb{R} \to \mathbb{R},$$

$$u_t + u \cdot u_x - \nu u_{xx} = f(x), \tag{2.2a}$$

$$u(x, 0) = u_0(x), \quad x \in \mathbb{R}, \tag{2.2b}$$

$$u(x, t) = u(x + 2k\pi, t), \quad x \in \mathbb{R},\ t \in [0, \mathcal{T}),\ k \in \mathbb{Z}, \tag{2.2c}$$

$$f(x) = f(x + 2k\pi), \quad x \in \mathbb{R},\ k \in \mathbb{Z}. \tag{2.2d}$$

2.2.1 The viscous Burgers equation in the Fourier basis

In this section we rewrite (2.2a) using *the Fourier basis* of 2π periodic functions $\{e^{ikx}\}_{i \in \mathbb{Z}}$. From now on we assume that all functions we use are sufficiently regular to be expanded in the Fourier basis and all necessary Fourier series converge.

Definition 2.2.1. *Let $u(x) \colon \mathbb{R} \to \mathbb{R}$ be a 2π periodic function. We refer to $\{a_k\}_{k \in \mathbb{Z}}$ as* the Fourier modes *of u. Where $a_k \in \mathbb{C}$ satisfies*

$$a_k = \frac{1}{2\pi} \int_0^{2\pi} u(x) e^{-ikx} \, dx, \tag{2.3}$$

moreover

$$u(x) = \sum_{k \in \mathbb{Z}} a_k e^{ikx}, \quad x \in \mathbb{R}. \tag{2.4}$$

Lemma 2.2.2. *Assume that* $|a_k| \leq \frac{M}{|k|^\gamma}$ *for* $k \in \mathbb{Z}$. *If* $n \in \mathbb{N}$ *is such that* $\gamma - n > 1$, *then the function* $u(x) = \sum_{k\in\mathbb{Z}} a_k e^{ikx}$ *belongs to* C^n. *The series*

$$\frac{\partial^s u}{\partial x^s} = \sum_{k\in\mathbb{Z}} a_k \frac{\partial^s}{\partial x^s} e^{ikx}$$

converges uniformly for $0 \leq s \leq n$.

Lemma 2.2.3. *Let* u_0 *be an initial value for the problem* (2.2), f *be a forcing. Then* (2.2) *rewritten in the Fourier basis become*

$$\frac{da_k}{dt} = -i\frac{k}{2} \sum_{k_1\in\mathbb{Z}} a_{k_1} \cdot a_{k-k_1} - \nu k^2 a_k + f_k, \quad k \in \mathbb{Z}, \tag{2.5a}$$

$$a_k(0) = \frac{1}{2\pi} \int_0^{2\pi} u_0(x) e^{-ikx}\, dx, \quad k \in \mathbb{Z}, \tag{2.5b}$$

$$f_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx}\, dx, \quad k \in \mathbb{Z}. \tag{2.5c}$$

For the proof refer [SuppMat].

Definition 2.2.4. *For any given number* $m > 0$ the m-th Galerkin projection *of* (2.5a) *is*

$$\frac{da_k}{dt} = -i\frac{k}{2} \sum_{\substack{|k-k_1|\leq m \\ |k_1|\leq m}} a_{k_1} \cdot a_{k-k_1} - \nu k^2 a_k + f_k, \quad |k| \leq m. \tag{2.6}$$

Note that in our case $\{a_k\}_{k\in\mathbb{Z}}$ are not independent. Solution u of (2.2) is real valued, which imposes that

$$a_k = \overline{a_{-k}}. \tag{2.7}$$

(2.7) is invariant under all Galerkin projections (2.6) as long as $f_k = \overline{f_{-k}}$.

2.3 Analytic arguments

In this section we provide some analytic arguments that we use in proving global existence and regularity results for solutions of (2.2).

2.3.1 Energy as Lyapunov function

Definition 2.3.1. *Energy of (2.5a) is given by the formula*

$$E(\{a_k\}) = \sum_{k\in\mathbb{Z}} |a_k|^2. \tag{2.8}$$

The following lemma provides an argument for the statement that *the energy* of (2.5a) is being absorbed by a ball whose radius depends on the forcing and the viscosity constant. Basing on this argument, later on, we will construct a trapping region for any Galerkin projection of (2.5a) . In particular, any trapping region constructed encloses the absorbing ball.

Lemma 2.3.2. *For any solution of* (2.5a) *or a Galerkin projection of* (2.5a) *such that* $a_{-k} = \overline{a_k}$

$$\frac{dE}{dt} = -2\nu \sum_{k\in\mathbb{Z}} k^2 |a_k|^2 + \sum_{k\in\mathbb{Z}} f_{-k} \cdot a_k + \sum_{k\in\mathbb{Z}} f_k \cdot a_{-k}.$$

Proof Using the symmetry of index in (2.8) we rewrite $\frac{dE}{dt} = \sum_{k\in\mathbb{Z}} (\frac{da_k}{dt} \cdot a_{-k}) + (\frac{da_{-k}}{dt} \cdot a_k)$.

$$\sum_{k\in\mathbb{Z}} (\frac{da_k}{dt} \cdot a_{-k}) + (\frac{da_{-k}}{dt} \cdot a_k) = \sum_{k\in\mathbb{Z}} -i \sum_{k_1\in\mathbb{Z}} (k-k_1) a_{k_1} \cdot a_{k-k_1} \cdot a_{-k}$$
$$+\sum_{k\in\mathbb{Z}} -i \sum_{k_1\in\mathbb{Z}} (-k-k_1) a_{k_1} \cdot a_{-k-k_1} \cdot a_k - 2\nu \sum_{k\in\mathbb{Z}} k^2 a_k \cdot a_{-k} + \sum_{k\in\mathbb{Z}} f_{-k} \cdot a_k + \sum_{k\in\mathbb{Z}} f_k \cdot a_{-k}$$
$$= 2\sum_{k\in\mathbb{Z}} -ik \sum_{k_1\in\mathbb{Z}} a_{k_1} \cdot a_{k-k_1} \cdot a_{-k} - 2\nu \sum_{k\in\mathbb{Z}} k^2 a_k \cdot a_{-k} + \sum_{k\in\mathbb{Z}} f_{-k} \cdot a_k + \sum_{k\in\mathbb{Z}} f_k \cdot a_{-k}.$$

We want to show that $\sum_{|k|\le N} k \sum_{\substack{|k_1|\le N \\ |k-k_1|\le N}} a_{k_1} \cdot a_{k-k_1} \cdot a_{-k} = 0$. In order to facilitate the proof explanation we denote $S_{N,k} := \sum_{\substack{|l|\le N \\ |k-l|\le N}} a_{k-l} \cdot a_l$ and $S_N := \sum_{|k|\le N} k \sum_{\substack{|k_1|\le N \\ |k-k_1|\le N}} a_{k_1} \cdot a_{k-k_1} \cdot a_{-k}$.

We proceed by induction, firstly we check if for $N = 1$ if thesis is fulfilled.

$$S_{1,k} = \begin{cases} a_{-1} \cdot a_0 + a_0 \cdot a_{-1} & , k = -1, \\ a_{-1} \cdot a_1 + a_0 \cdot a_0 + a_1 \cdot a_{-1} & , k = 0, \\ a_1 \cdot a_0 + a_0 \cdot a_1 & , k = 1, \end{cases}$$

$$S_1 = -1(a_{-1} \cdot a_0 \cdot a_1 + a_0 \cdot a_{-1} \cdot a_1) + 1(a_1 \cdot a_0 \cdot a_{-1} + a_0 \cdot a_1 \cdot a_{-1}) = 0.$$

We verify the induction step $S_{N-1} = 0 \Rightarrow S_N = 0$

$$S_N = S_{N-1}$$
$$+\sum \begin{cases} a_N \cdot a_{-N+k} \cdot a_{-k} 2k & , 0 < k < N, \quad (S_I) \\ a_{-N} \cdot a_{k+N} \cdot a_{-k} 2k & , -N < k < 0, \quad (S_{II}) \\ (a_{-N} \cdot a_0 + a_{-N+1} \cdot a_{-1} + \cdots + a_0 \cdot a_{-N}) \cdot a_N (-N) & , k = -N, \quad (S_{III}) \\ (a_N \cdot a_0 + a_{N-1} \cdot a_1 + \cdots + a_0 \cdot a_N) \cdot a_{-N} N & , k = N, \quad (S_{IV}) \end{cases}$$

we match elements with the same modes from (S_I) and (S_{III})

$$\sum_{0<k<N} a_N \cdot a_{k-N} \cdot a_{-k}(2k-N)$$
$$= \sum_{0<k<\frac{N}{2}} a_N \cdot a_{k-N} \cdot a_{-k}(2k-N+N-2k) + e(N) a_N \cdot a^2_{-\frac{N}{2}}(N-N) = 0.$$

we match elements with the same modes from (S_{II}) with and (S_{IV})

$$\sum_{0<k<N} a_{-N} \cdot a_{N-k} \cdot a_k(N-2k)$$
$$= \sum_{0<k<\frac{N}{2}} a_{-N} \cdot a_{N-k} \cdot a_k(N-2k+2k-N) + e(N) a_{-N} \cdot a^2_{\frac{N}{2}}(N-N) = 0.$$

Where $e(N) = 1$ for N even and $e(N) = 0$ for N odd. After substitution all that is left is

$$S_N = S_{N-1} + 2N a_N \cdot a_0 \cdot a_{-N} - 2N a_{-N} \cdot a_0 \cdot a_N = 0. \quad \blacksquare$$

2.3.2 A trapping region for (2.5a)

In this section we provide a forward invariant set for each Galerkin projection of (2.5a), called *the trapping region* . If we consider an arbitrary initial condition that is inside a trapping region, then the corresponding trajectory remain in this set in the future. This is an argument for the existence of solutions of each Galerkin projection of (2.5a) within a trapping region. Moreover, due to the existence of a trapping region, the solution of (2.5a), obtained by passing to the limit, conserves the initial regularity. We use this fact to argue that a solution of (2.5) with sufficiently regular initial data at any time exists, is unique, and is a classical solution of (2.2). Calculations performed in this section were inspired by the trapping regions built for the Navier-Stokes equations, see [MS] and [ES].

Notation For the purpose of simplifying the notation we "redefine" the absolute value of 0, from now on $|0| := 1$.

Lemma 2.3.3. *Let $\{a_k\}_{k\in\mathbb{Z}} \in l_2$, $N_k := -i\frac{k}{2}\sum_{k_1\in\mathbb{Z}} a_{k_1} \cdot a_{k-k_1}$. Assume that there exists $C > 0$ and $s > 0.5$ such that $\{a_k\}_{k\in\mathbb{Z}}$ satisfy $|a_k| \leq \frac{C}{|k|^s}$, $k \in \mathbb{Z}$. Then*

$$|N_k| \leq \frac{2^{s-\frac{1}{2}}\sqrt{E}C + 2^{s-1}\sqrt{E}C\frac{1}{\sqrt{2s-1}}}{|k|^{s-\frac{3}{2}}}, \quad k \in \mathbb{Z}.$$

Proof We start with the following easy estimate

$$|N_k| \leq \frac{1}{2}|k| \sum_{k_1\in\mathbb{Z}} |a_{k_1}||a_{k-k_1}|,$$

for $k \in \mathbb{Z}$. We split into two sub-cases

Case 1 $N_k^I = -i\frac{k}{2}\sum_{k_1} a_{k_1}\cdot a_{k-k_1}, \quad |k_1| \le \frac{1}{2}|k|$

$$|N_k^I| \le \sum_{|k_1|\le\frac{1}{2}|k|} \frac{1}{2}|k||a_{k_1}||a_{k-k_1}| \le \sum_{|k_1|\le\frac{1}{2}|k|} \frac{1}{2}|k|\frac{C}{|k-k_1|^s}|a_{k_1}|$$
$$\le \frac{2^{s-1}C}{|k|^{s-1}}\sqrt{\sum_{|k_1|\le\frac{1}{2}|k|}|a_{k_1}|^2}\sqrt{\sum_{|k_1|\le\frac{1}{2}|k|}1} \le \frac{2^{s-1}\sqrt{E}C\sqrt{2}}{|k|^{s-\frac{3}{2}}}.$$

Case 2 $N_k^{II} = -i\frac{k}{2}\sum_{k_1} a_{k_1}\cdot a_{k-k_1}, \quad |k_1| > \frac{1}{2}|k|$

$$|N_k^{II}| \le \sum_{|k_1|>\frac{1}{2}|k|} \frac{1}{2}|k||a_{k_1}||a_{k-k_1}| \le \frac{1}{2}|k|C\sum_{|k_1|>\frac{1}{2}|k|}\frac{1}{|k_1|^s}|a_{k-k_1}|$$
$$\le \frac{1}{2}|k|C\sqrt{\sum_{|k_1|>\frac{1}{2}|k|}\frac{1}{|k_1|^{2s}}}\sqrt{\sum_{|k_1|>\frac{1}{2}|k|}|a_{k-k_1}|^2}$$
$$\le \frac{1}{2}|k|\sqrt{E}C\sqrt{\frac{2^{2s}}{(2s-1)|k|^{2s-1}}} = \frac{2^{s-1}\sqrt{E}C\frac{1}{\sqrt{2s-1}}}{|k|^{s-\frac{3}{2}}}.$$

We used the estimation, the first equality is obvious for $|k|$ even, for $|k|$ odd it holds, because of the convexity functions

$$\sum_{|k_1|>\frac{1}{2}|k|}\frac{1}{|k_1|^{2s}} < 2\int_{\frac{1}{2}|k|}^{\infty}\frac{1}{r^{2s}}\,dr = 2\left[-\frac{1}{(2s-1)r^{2s-1}}\right]_{\frac{1}{2}|k|}^{\infty} = \frac{2^{2s}}{(2s-1)|k|^{2s-1}}.$$

After summing together Case 1 and Case 2, for any $k\in\mathbb{Z}$

$$|N_k| \le |N_k^I| + |N_k^{II}| = \frac{2^{s-\frac{1}{2}}\sqrt{E}C + 2^{s-1}\sqrt{E}C\frac{1}{\sqrt{2s-1}}}{|k|^{s-\frac{3}{2}}}.$$

holds. ∎

Theorem 2.3.4. *Let* $J>0$, $s>0.5$, $E_0=\frac{E_f}{\nu^2}$, $E>E_0$, $D=2^{s-\frac{1}{2}}+\frac{2^{s-1}}{\sqrt{2s-1}}$, $C>\sqrt{E}N^s$, $N>\max\left\{J,\left(\frac{\sqrt{E}D}{\nu}\right)^2\right\}$ *and* $f_k=0$ *for* $|k|>J$, *then*

$$W_0(E,N,C,s) = \{\{a_k\} \mid E(\{a_k\})\le E,\ |a_k|\le\frac{C}{|k|^s},\ |k|>N\}$$

is a trapping region for each Galerkin projection of (2.5a).

Proof First, we verify condition under which the energy is decreasing

$$2\nu\sum_{k\in\mathbb{Z}}k^2a_k\cdot a_{-k}\geq 2\nu\sum_{k\in\mathbb{Z}}|a_k|^2>\sum_{k\in\mathbb{Z}}|f_{-k}||a_k|+|f_k||a_{-k}|\geq\sum_{k\in\mathbb{Z}}f_{-k}|a_k|+f_k|a_{-k}|,$$

$$2\nu\sqrt{E}>2\sqrt{E_f},$$

$$E>\frac{E_f}{\nu^2}.$$

Therefore

$$\text{if } E>E_0=\frac{E_f}{\nu^2} \text{ then } \frac{dE}{dt}<0. \tag{2.9}$$

We observe that the condition $|a_k|\leq\frac{C}{|k|^s}$ is satisfied for all $k\in\mathbb{Z}$. Since $E(\{a_k\})\leq E$ and $|a_k|\leq\sqrt{E}$

$$|a_k|\leq\sqrt{E}\leq\frac{C}{|k|^s} \text{ because } C>\sqrt{E}N^s.$$

Now, we shall check if on ∂N_0 vector field is pointing inwards. For points at the boundary of W_0 such that $E(\{a_k\})=E$ and $E>\frac{E_f}{\nu^2}$ vector field points inwards from (2.9). For points such that $|a_k|=\frac{C}{|k|^s}$ for some $|k|>N$ we perform calculations to check if the diminution condition $\frac{d|a_k|}{dt}<0$ holds. We use Lemma 2.3.3.

$$\frac{d|a_k|}{dt}<-\nu|k|^2\frac{C}{|k|^s}+\frac{D\sqrt{E}C}{|k|^{s-\frac{3}{2}}}<0,$$

$$\nu|k|^2\frac{C}{|k|^s}>\frac{D\sqrt{E}C}{|k|^{s-\frac{3}{2}}},$$

$$\nu\sqrt{|k|}>D\sqrt{E},$$

$$|k|>\left(\frac{D\sqrt{E}}{\nu}\right)^2,$$

$\frac{d|a_k|}{dt}<0$ holds if $|k|>\frac{D^2E}{\nu^2}$. Proof is completed because $|k|>N>\frac{D^2E}{\nu^2}$. ∎

2.4 Global results

Let $\overline{H}\subset H$ be a subspace of sufficiently regular L^2 functions whose Fourier coefficients $\{a_k\}_{k\in\mathbb{Z}}$ satisfy $|a_k|\leq\frac{C}{|k|^s}$ with $s\geq 4$.

Notation Let $l>0$, from now on by $\varphi^l(t,x)$ we denote the solution of l-th Galerkin projection of (2.5a) at a time $t>0$, with an initial value $x\in P_l(H)$, we use $\{a_k\}_{|k|\leq l}$ to denote an initial condition $x\in P_l(H)$. Let $P_l(H)\ni\{a_k^l(t)\}_{|k|\leq l}:=\varphi^l(t,\{a_k\}_{|k|\leq l})$, $t>0$, $l>0$. $\{a_k^l(t)\}_{|k|\leq l}$ is well defined, because

solutions for each Galerkin projection of (2.5a) at any time $t > 0$ exists due to Theorem 2.3.4 (existence of a trapping region) and are unique due to the fact that (2.6) is a finite system of ODEs with a locally Lipschitz right-hand side. When it is known from context or irrelevant in this context we drop the index l.

Lemma 2.4.1. *Let $M_1 \geq 0$, $E_0 = \frac{E_f}{\nu^2}$, $\widetilde{E} > E_0$, $W \subset H$, $P_l(W) \subset W$ be a trapping region for l-th Galerkin projection of* (2.5a) *for all $l > M_1$.*
There exists a finite time $t_1 \geq 0$ such that $E\left(\varphi^l(t_1, P_l(\{a_k\}_{k\in\mathbb{Z}}))\right) \leq \widetilde{E}$ for all $\{a_k\}_{k\in\mathbb{Z}} \in W$ and $l > M_1$.

Proof Let $\{a_k\}_{k\in\mathbb{Z}} \in W$, let $E\left(\{a_k\}_{k\in\mathbb{Z}}\right) = E_I$ be an initial energy. It is enough to take either $t_1 = 0$ if $E_I \leq \widetilde{E}$ or $t_1 = \frac{1}{2\nu\varepsilon} \ln \frac{E_I}{\widetilde{E}}$ if $E_I > \widetilde{E}$, where $\varepsilon = \left(1 - \sqrt{\frac{E_0}{\widetilde{E}}}\right)$.
To see this, we calculate in similar fashion as in the proof of Theorem 2.3.4. Let $E_I > \widetilde{E}$, by Lemma 2.3.2 we have

$$\frac{dE}{dt} \leq -2\nu E + 2\sqrt{E}\sqrt{E_f} = -2\nu E\left(1 - \frac{\sqrt{E_f}}{\nu\sqrt{E}}\right) \leq -2\nu E\left(1 - \sqrt{\frac{E_0}{\widetilde{E}}}\right),$$

therefore

$$E(t) \leq e^{-2\nu t\left(1-\sqrt{\frac{E_0}{\widetilde{E}}}\right)} E_I.$$

We set $t_1 := t$, where t satisfies $e^{-2\nu\varepsilon t} E_I = \widetilde{E}$. ∎

Lemma 2.4.2. *Let $M_1 \geq 0$, $i \in \mathbb{Z}$, $W \subset H$, $P_l(W) \subset W$ be a trapping region for l-th Galerkin projection of* (2.5a) *for all $l > M_1$. Assume that $N_i^{\pm} \in \mathbb{R}^2$ are bounds such that*

$$N_i(\{a_k\}_{k\in\mathbb{Z}}) \in [N_{i,1}^-, N_{i,1}^+] \times [N_{i,2}^-, N_{i,2}^+]$$

for all $\{a_k\}_{k\in\mathbb{Z}} \in W$.
Then for any $\varepsilon > 0$ there exists a finite time $\hat{t} > 0$ such that for all $l > \max\{M_1, |i|\}$ and $t \geq \hat{t}$ $a_i^l(t)$ with any initial condition in $P_l(W)$ satisfies

$$a_i^l(t) \in \left[b_{i,1}^-, b_{i,1}^+\right] \times \left[b_{i,2}^-, b_{i,2}^+\right] + \left[-\varepsilon, \varepsilon\right]^2,$$

where $b_{i,j}^{\pm} = \frac{N_{i,j}^{\pm} + f_{i,j}}{-\lambda_i}$, $j = 1, 2$.

Proof In the calculations we drop the index l denoting the Galerkin projection dimension and the index j denoting the coordinate for better clarification, for instance instead of $a_{i,j}^l$ we write a_i. We perform the calculations for the first and the second component simultaneously, thus finally we obtain

two values $t_{i,1} > 0$ and $t_{i,2} > 0$. For any Galerkin projection of (2.5a) from $\frac{da_i}{dt} \leq -\nu i^2 a_i + N_i^+ + f_i$, $\frac{da_i}{dt} \geq -\nu i^2 a_i + N_i^- + f_i$ it follows that

$$a_i(t) \geq \left(a_i^- - b_i^-\right) e^{-\nu i^2 t} + b_i^-, \quad a_i(t) \leq \left(a_i^+ - b_i^+\right) e^{-\nu i^2 t} + b_i^+.$$

Where $a_i^{\pm}$ are bounds such that $a_i \in [a_{i,1}^-, a_{i,1}^+] \times [a_{i,2}^-, a_{i,2}^+]$, which exist because the initial condition is contained in a compact trapping region.
Because $-\nu i^2 t < 0$ for any $t > 0$ and $i \in \mathbb{Z}$ it follows that for a sufficiently large time $t_i > 0$ we have $\left(\left|a_i^+ - b_i^+\right| + \left|a_i^- - b_i^-\right|\right) e^{-\nu i^2 t} \leq \varepsilon$ for any $t \geq t_i$. It is enough to take

$$t_i := -\lg \frac{\varepsilon}{\left(\left|a_i^+ - b_i^+\right| + \left|a_i^- - b_i^-\right|\right)} / \left(\nu i^2\right).$$

Finally $\hat{t} := \max\{t_{i,1}, t_{i,2}\}$. ∎

Lemma 2.4.3. *Let $M_1 \geq 0$, $W \subset H$, $P_l(W) \subset W$ be a trapping region for l-th Galerkin projection of* (2.5a) *for all $l > M_1$.*
Assume that C_N, s_N are numbers such that for all $\{a_k\}_{k\in\mathbb{Z}} \in W$

$$|N_k(\{a_k\}_{k\in\mathbb{Z}})| \leq \frac{C_N}{|k|^{s_N}} \text{ for } |k| > M_1.$$

Then for any $\varepsilon > 0$ there exists a finite time $\hat{t} \geq 0$ such that for all $l > M_1$ and $t \geq \hat{t}$ $\left\{a_k^l(t)\right\}_{|k|\leq l}$ with any initial condition in $P_l(W)$ satisfy

$$|a_k^l(t)| \leq \frac{C_b + \varepsilon}{|k|^{s_b}} \text{ for } |k| > M_1,$$

where $C_b = \left(C_N + \max_{|k|>M_1}\left\{|f_k||k|^{s_N}\right\}\right)/\nu$, $s_b = s_N + 2$.

Proof We use the formula (2.37)

$$|a_k^l(t)| \leq \frac{C_a (k_{max})^{s_b - s_a} e^{-\nu (k_{max})^2 t} + C_b}{|k|^{s_b}}, \quad |k| > M_1$$

for all $l > M_1$, where $C_b = \left(C_N + \max_{|k|>M_1}\left\{|f_k||k|^{s_N}\right\}\right)/\nu$, $s_b = s_N + 2$, C_a and s_a are numbers such that $|a_k| \leq \frac{C_a}{|k|^{s_a}}$. Analogically, for a sufficiently large time $t_{\mathcal{F}} > 0$

$$C_a (k_{max})^{s_b - s_a} e^{-\nu (k_{max})^2 t} \leq \varepsilon, \quad t \geq t_{\mathcal{F}},$$

therefore

$$|a_k^l(t)| \leq \frac{C_b + \varepsilon}{|k|^{s_b}}, \quad t \geq t_{\mathcal{F}},\ l > M_1.$$

Finally, the obtained time $t_{\mathcal{F}}$ is uniform with respect to the projection dimension l. ∎

Lemma 2.4.4. *Let $E > 0$. The following estimate holds*

$$|N_k(\{a_k\}_{k\in\mathbb{Z}})| \leq \frac{1}{2}|k|E$$

for all $\{a_k\}_{k\in\mathbb{Z}} \in \{\{a_k\}_{k\in\mathbb{Z}} \in H \mid E(\{a_k\}_{k\in\mathbb{Z}}) \leq E\}$.

Proof Let $\{a_k\}_{k\in\mathbb{Z}} \in \{\{a_k\}_{k\in\mathbb{Z}} \in H \mid E(\{a_k\}_{k\in\mathbb{Z}}) \leq E\}$

$$|N_k(\{a_k\}_{k\in\mathbb{Z}})| \leq \frac{1}{2}|k| \sum_{k_1\in\mathbb{Z}} |a_{k-k_1}||a_{k_1}|,$$

$$|N_k(\{a_k\}_{k\in\mathbb{Z}})| \leq \frac{1}{2}|k| \sqrt{\sum_{k_1\in\mathbb{Z}} |a_{k_1}|^2} \sqrt{\sum_{k_1\in\mathbb{Z}} |a_{k-k_1}|^2},$$

$$|N_k(\{a_k\}_{k\in\mathbb{Z}})| \leq \frac{1}{2}|k|E. \qquad \blacksquare$$

Now, we shall introduce *the absorbing sets*. For any initial condition there exists a finite time after which the solutions of Galerkin projections are trapped in an absorbing set. We use absorbing sets as a tool for studying the global dynamics of (2.5a).

Definition 2.4.5. *Let $M_1 > 0$. Set $A \subset H$ is called* the absorbing set *for any Galerkin projection of* (2.5a)*, if for any initial condition $\{a_k\}_{k\in\mathbb{Z}} \in H$ there exists a finite time $t_1 \geq 0$ such that for all $l > M_1$ and $t \geq t_1$ $\varphi^l\left(t, P_l(\{a_k\}_{k\in\mathbb{Z}})\right) \in P_l(A)$.*

In the next result, to show the existence, we construct analytically an absorbing set. Furthermore, we construct absorbing sets with any order of polynomial decay. Later on, in the context of a computer assisted proof of the main theorem, we will construct an absorbing set using *the interval arithmetic*. Accomplishing this task requires the established existence of an absorbing set with a sufficiently large order of polynomial decay.

Lemma 2.4.6. *Let $\varepsilon > 0$, $E_0 = \frac{E_f}{\nu^2}$, $\widetilde{E} > E_0$, $M_1 > 0$: $f_k = 0$ for $|k| > M_1$,*

$$H \supset W_i\big(\widetilde{E}\big) := \left\{ \{a_k\}_{k\in\mathbb{Z}} \mid E(\{a_k\}_{k\in\mathbb{Z}}) \leq \widetilde{E},\ |a_k| \leq \frac{C_i}{|k|^{s_i}} \text{ for } |k| > M_1 \right\},$$

where $C_2 = \varepsilon + \left(\frac{1}{2}\widetilde{E} + \max_{|k|\leq J} \frac{|f_k|}{|k|}\right)/\nu$, $D = 2^{s-\frac{1}{2}} + \frac{2^{s-1}}{\sqrt{2s-1}}$ and
$s_i = i/2$, $C_i = \varepsilon + \left(C_{i-1}\sqrt{\widetilde{E}}D\right)/\nu$ for $i > 2$.
Then for all $i \geq 2$ $W_i\big(\widetilde{E}\big)$ is an absorbing set for any Galerkin projection of (2.5a).

Proof Let $\widetilde{E} > E_0$, $\{\widehat{a_k}\}_{k\in\mathbb{Z}}$ be an arbitrary initial condition for (2.5), $E\left(\{\widehat{a_k}\}_{k\in\mathbb{Z}}\right) =: E$. Let

$$W_0 := \left\{ \{a_k\}_{k\in\mathbb{Z}} \mid E(\{a_k\}_{k\in\mathbb{Z}}) \leq E,\ |a_k| \leq \frac{C_0}{|k|^{s_0}} \right\} \tag{2.10}$$

be a trapping region for each Galerkin projection of (2.5a) enclosing $\{\widehat{a_k}\}_{k\in\mathbb{Z}}$, which exists due to Theorem 2.3.4. Note that a trapping region can be scaled

to make it enclose an arbitrary initial condition. It follows from Lemma 2.4.1 that there exists a finite time $t_1 \geq 0$ such that for all $\{a_k\}_{k\in\mathbb{Z}} \in W_0$ and $l > M_1$

$$E(\varphi^l(t_1, P_l(\{a_k\}_{k\in\mathbb{Z}}))) \leq \widetilde{E}. \tag{2.11}$$

We define $W_1 := W_0 \cap \left\{\{a_k\}_{k\in\mathbb{Z}} \mid E(\{a_k\}_{k\in\mathbb{Z}}) \leq \widetilde{E}\right\}$. From (2.11) and that W_0, W_1 are trapping regions we immediately have that $\varphi^l(t, P_l(\{a_k\}_{k\in\mathbb{Z}}) \in W_1$ for all $\{a_k\}_{k\in\mathbb{Z}} \in W_0$, $t \geq t_1$ and $l > M_1$. Using Lemma 2.4.4 we bound the nonlinear part

$$|N_k(\{a_k\}_{k\in\mathbb{Z}})| \leq \frac{1}{2}|k|\widetilde{E} \text{ for all } \{a_k\}_{k\in\mathbb{Z}} \in W_1. \tag{2.12}$$

It follows from Lemma 2.4.3 that there exists a finite time $t_2 \geq t_1$ such that for all $t \geq t_2$ and $l > M_1$, $\{a_k^l(t)\}_{|k|\leq l}$ with any initial condition in $P_l(W_1)$ satisfy

$$|a_k^l(t)| \leq \frac{C_2}{|k|} \text{ for } |k| > M_1, \tag{2.13}$$

where $C_2 = \varepsilon + \left(\frac{1}{2}\widetilde{E} + \max_{|k|\leq J} \frac{|f_k|}{|k|}\right)/\nu$. It is important to start with the energy estimate (2.12) to bound the nonlinear part N_k because the goal is to estimate $|a_k|$ uniformly with respect to C_0 and s_0 (2.10). We emphasize that C_2 from (2.13) does not depend on C_0 and s_0. Having the bound (2.13), we construct the following absorbing set

$$W_2 := \left\{\{a_k\}_{k\in\mathbb{Z}} \mid E(\{a_k\}_{k\in\mathbb{Z}}) \leq \widetilde{E},\ |a_k| \leq \frac{C_2}{|k|}\right\}.$$

Due to Lemma 2.3.3 the following estimate holds

$$|N_k(\{a_k\}_{k\in\mathbb{Z}})| \leq \frac{C_2\sqrt{\widetilde{E}}D}{|k|^{-\frac{1}{2}}},$$

for all $\{a_k\}_{k\in\mathbb{Z}} \in W_2$. Due to Lemma 2.4.3 again there exists a finite time $t_3 \geq t_2$ such that for all $t \geq t_3$ and $l > M_1$, $\{a_k^l(t)\}_{|k|\leq l}$ with any initial condition in $P_l(W_2)$ satisfy

$$|a_k^l(t)| \leq \frac{C_3}{|k|^{\frac{3}{2}}} \text{ for } |k| > M_1,$$

where $C_3 = \varepsilon + \left(C_2\sqrt{\widetilde{E}}D\right)/\nu$, note that $f_k = 0$ for $|k| > M_1$. Having this bound, we construct the following absorbing set

$$W_3 := \left\{\{a_k\}_{k\in\mathbb{Z}} \mid E(\{a_k\}_{k\in\mathbb{Z}}) \leq \widetilde{E},\ |a_k| \leq \frac{C_3}{|k|^{\frac{3}{2}}} \text{ for } |k| > M_1\right\}.$$

Note the gain of $\frac{1}{2}$ in the order of polynomial decay of $\{a_k\}_{k\in\mathbb{Z}}$ in W_3 comparing to W_2. From applying Lemma 2.3.3 and Lemma 2.4.3 further we obtain a sequence of times $t_3 < t_4 < \cdots < t_n < \ldots$ such that

$$|a_k^l(t_3)| \leq \frac{C_3}{|k|^{s_3}}, \ldots,\ |a_k^l(t_n)| \leq \frac{C_n}{|k|^{s_n}}, \ldots, \text{ for } |k| > M_1,$$

with $s_i = i/2$, $C_i = \varepsilon + \left(C_{i-1}\sqrt{\widetilde{E}}D\right)/\nu$. Obtained W_i, $i \geq 2$ are absorbing sets for any Galerkin projection of (2.5a), which follows from the construction and that C_i for all $i \geq 2$ depend only on the energy $\widetilde{E}$. ∎

Remark 2.4.7. The same assumptions as in Lemma 2.4.6. Then the following holds $W_i(\widetilde{E}) \subset \overline{H}$ for $i \geq 8$.

Lemma 2.4.8. *Let $i \in \mathbb{Z}$, $\varepsilon > 0$, $A \subset H$ be an absorbing set for any Galerkin projection of* (2.5a). *Let $N_i^{\pm} \in \mathbb{R}^2$ be bounds such that $N_i(\{a_k\}_{k\in\mathbb{Z}}) \in [N_{i,1}^-, N_{i,1}^+] \times [N_{i,2}^-, N_{i,2}^+]$ for all $\{a_k\}_{k\in\mathbb{Z}} \in A$.*
Then $A \cap \left\{\{a_k\}_{k\in\mathbb{Z}} \mid a_i \in \left[b_{i,1}^-, b_{i,1}^+\right] \times \left[b_{i,2}^-, b_{i,2}^+\right] + [-\varepsilon,\varepsilon]^2\right\}$ is also an absorbing set for any Galerkin projection of (2.5a), *$b_{i,j}^{\pm} = \frac{N_{i,j}^{\pm} + f_{i,j}}{-\lambda_i}$, $j = 1, 2$.*

Proof Immediate consequence of Lemma 2.4.2. ∎

2.5 General method of self-consistent bounds.

For the purpose of the presented work we call a dissipative PDE a PDE of the following type

$$\frac{du}{dt} = Lu + N(u, Du, \ldots, D^r u) + f = F(u), \tag{2.14}$$

where $u \in \mathbb{R}^n$, $x \in \mathbb{T}^d$, ($\mathbb{T}^d = (\mathbb{R}/2\pi)^d$ is an d-dimensional torus), L is a linear operator, N a polynomial and by $D^s u$ we denote the collection of s-th order partial derivatives of u. Right-hand side contain a constant in time forcing function f. We require that L is diagonal in the Fourier basis $\{e^{ikx}\}_{k\in\mathbb{Z}^d}$

$$Le^{ikx} = \lambda_k e^{ikx}$$

and the *eigenvalues* λ_k satisfy

$$\lambda_k = -\nu(|k|)|k|^p, \tag{2.15a}$$
$$0 < \nu_0 \leq \nu(|k|) \leq \nu_1, \quad \text{for } |k| > K_-, \tag{2.15b}$$
$$p > r. \tag{2.15c}$$

We start our approach by replacing a sufficiently regular u and f in (2.14) by the Fourier series, i.e. $u(x,t) = \sum_{k\in\mathbb{Z}^d} a_k(t)e^{ikx}$ and $f(x) = \sum_{k\in\mathbb{Z}^d} f_k e^{ikx}$. We obtain a system of ODEs describing the evolution of the coefficients $\{a_k\}_{k\in\mathbb{Z}^d}$, where a_k is the coefficient corresponding to e^{ikx}.

$$\frac{da_k}{dt} = F_k(a) = L_k(a) + N_k(a) + f_k = \lambda_k a_k + N_k(a) + f_k, \quad k \in \mathbb{Z}^d. \tag{2.16}$$

Lemma 2.5.1. *Let ν be the viscosity constant in* (2.2a), *then* (2.2a) *satisfies the conditions* (2.15) *with $d = 1$, $\lambda_k = -\nu k^2$, $p = 2$, $r = 1$, $\nu(k) = \nu$.*

2.5.1 Self-consistent bounds

We recall, in context of dPDEs, the definition of self-consistent bounds from [Z3]. Let H be a Hilbert space, actually L_2 or one of its subspaces in the context of dPDEs. We assume that domain of F, right hand side of (2.14), is dense in H. By a solution of (2.14) we understand a function $u\colon [0,\mathcal{T}) \to \operatorname{dom}(F)$ such that u is differentiable and (2.14) is satisfied for all $t \in [0,\mathcal{T})$, $\mathcal{T}$ is a maximal time of existence. We assume that there is a set $I \subset \mathbb{Z}^d$ and a sequence of subspaces $H_k \subset H$ such that $\dim H_k = d_1 < \infty$, H_k and $H_{k'}$ are mutually orthogonal for $k \neq k'$ and $H = \overline{\oplus_{k\in I} H_k}$. For $n > 0$ we set $X_n = \oplus_{|k|\leq n, k\in I} H_k$, $Y_n = X_n^{\perp}$. By $P_n\colon H \to X_n$ and $Q_n\colon H \to Y_n$ we denote the orthogonal projections onto X_n and Y_n respectively.

In our approach we solve the system of equations (2.16) instead of (2.14). (2.16) is defined on the $l_2 = \left\{\{a_k\}\colon \sum |a_k|^2 < \infty\right\}$ space or one of its subspaces. Domain of (2.16) is dense in l_2. We associate a_k with the coefficient corresponding to e^{ikx} in the Fourier expansion of u. It is equivalent to solve either (2.14) or (2.16), assuming that the initial condition $u_0 \in H$ is sufficiently regular. In our approach we expand u_0 in the Fourier basis to get the initial value for all the variables $\{a_k(0)\}_{k\in\mathbb{Z}}$. We argue that the solution of (2.16) is defined at any time $t > 0$, moreover, the solution conserves its initial regularity due to existence of *trapping regions* and in fact is a classical solution of (2.14). For the details refer Section 2.6 and Section 2.7.

Definition 2.5.2. [Z3, Def. 1] *We say that* $F\colon H \supset \operatorname{dom}(F) \to H$ *is admissible if the following conditions are satisfied for any* $i \in \mathbb{R}$ *such that* $\dim X_i > 0$

- $X_i \subset \operatorname{dom}(F)$
- $P_i F\colon X_i \to X_i$ *is a* C^1 *function*

Definition 2.5.3. [Z3, Def. 2] *Assume* F *is an admissible function. Let* $m,\ M \in \mathbb{R}$ *with* $m \leq M$. *Consider an object consisting of: a compact set* $W \subset X_m$ *and a sequence of compact sets* $B_k \subset H_k$ *for* $|k| > m,\ k \in I$. *We define the conditions* ***C1, C2, C3, C4a*** *as follows:*

C1 *For* $|k| > M,\ k \in I$ *holds* $0 \in B_k$.

C2 *Let* $\hat{a}_k \colon = \max_{a\in B_k} \|a\|$ *for* $|k| > m,\ k \in I$ *and then* $\sum_{|k|>m, k\in I} \hat{a}_k^2 < \infty$. *In particular*

$$W \oplus \Pi_{|k|>m} B_k \subset H$$

and for every $u \in W \oplus \Pi_{k\in I, |k|>m} B_k$ *holds ,* $\|Q_n u\|^2 \leq \sum_{|k|>n, k\in I} \hat{a}_k^2$.

C3 *The function* $u \mapsto F(u)$ *is continuous on* $W \oplus \Pi_{k\in I, |k|>m} B_k \subset H$. *Moreover, if we define for* $k \in I$, $f_k = \max_{u\in W\oplus\Pi_{k\in I,|k|>m} B_k} |F_k(u)|$, *then* $\sum f_k^2 < \infty$.

C4a *For $|k| > m,\ k \in I$ B_k is given by* (2.17) *or* (2.18)

$$B_k = \overline{B(c_k, r_k)}, \quad r_k > 0 \tag{2.17}$$

$$B_k = \Pi_{s=1}^{d_1} \left[a_s^-, a_s^+\right], \quad a_s^- < a_s^+ \tag{2.18}$$

Let $u \in W \oplus \Pi_{|k|>m} B_k$. Then for $|k| > m$ holds:

- *if B_k is given by* (2.17) *then*

$$u_k \in \partial_{H_k} B_k \Rightarrow (u_k - c_k | F_k(u)) < 0.$$

- *if B_k is given by* (2.18) *then*

$$u_{k,s} = a_{k,s}^- \Rightarrow F_{k,s}(u) > 0,$$
$$u_{k,s} = a_{k,s}^+ \Rightarrow F_{k,s}(u) < 0.$$

Definition 2.5.4. [Z3, Def. 4] *Assume F is an admissible function. Let $m, M \in \mathbb{R}$ with $m \leq M$. Consider an object consisting of: a compact set $W \subset X_m$ and a sequence of compacts $B_k \subset H_k$ for $|k| > m, k \in I$. We say that set $W \oplus \Pi_{k\in I, |k|>m} B_k$ forms* self-consistent bounds for F *if conditions **C1**, **C2**, **C3** are satisfied.*
*If additionally condition **C4a** holds, then we say that $W \oplus \Pi_{k\in I, |k|>m} B_k$ forms* topologically self-consistent bounds for F.

To establish notation in the next sections we provide

Definition 2.5.5. *Given an object $W \oplus \Pi_{|k|>m} a_k$, $W \subset X_m$ and a sequence of compact sets $a_k \subset H_k$ for $|k| > m$, $m, M \in \mathbb{R}_+$, $m \leq M$*

- *W is called* the finite part,
- *$\Pi_{|k|>m} a_k$ is called* the tail *and denoted by T,*
- *$\Pi_{m<|k|\leq M} a_k$ is called* the near tail *and denoted by $T_{\mathcal{N}}$,*
- *$\Pi_{|k|>M} a_k$ is called* the far tail *and denoted by $T_{\mathcal{F}}$.*

$T_{\mathcal{N}}$ is the finite part of a tail, whereas $T_{\mathcal{F}}$ is the infinite part of a tail, in fact in our approach

$$T_{\mathcal{F}} := \prod_{|k|>M} \overline{B(0, C/|k|^s)}, \quad C \in \mathbb{R}_+,\ s \geq d + p + 1. \tag{2.19}$$

We recall that a set $W \oplus T \subset H$ with $T_{\mathcal{F}}$ as defined in (2.19) satisfies conditions C1, C2 and C3 of Definition 2.5.3 with $I = \mathbb{Z}^d$, namely forms a self-consistent bounds for (2.16) and equivalently forms a self-consistent bounds for (2.14). It is allowable to associate the finite part W with the near tail $T_{\mathcal{N}}$, but we keep the distinction because of the different treatment of both in the algorithm.

We do not address here question if solutions of a general dPDE (2.14) exists and are unique, it has been thoroughly answered in [Z3], see [Z3, Theorem 11].

2.6 Local existence and uniquenes

Regarding local existence and uniqueness we rely on results from [ZAKS]. For the sake of completeness we recall main theorems. The same symbols as in the preceding part are used in general context.

Definition 2.6.1. *Let $R \subset H$, R is convex, $l > 0$, $x \in X_l$, $\varphi^l(t,x)$ be the local flow inducted by the l-th Galerkin projection of* (2.16). *We call $P_l(R)$* a trapping region *for the l-th Galerkin projection of* (2.16) *if $\varphi^l(t, P_l(R)) \subset P_l(R)$ for all $t > 0$ or equivalently the vector field on the boundary of $P_l(R)$ is pointing inwards.*

Theorem 2.6.2. [ZAKS, Thm. 7] *Assume that $R \subset H$, R is convex and F satisfies conditions C1, C2, C3. Assume that we have a block decomposition of H, such that condition **Db** holds*

Db *there exists $l \in \mathbb{R}$ such that for any (i) and $x \in R$*

$$\mu\left(\frac{\partial F_{(i)}}{\partial x_{(i)}}(x)\right) + \sum_{(k),(k)\neq(i)} \left|\frac{\partial F_{(i)}}{\partial x_{(k)}}(x)\right| \leq l \tag{2.20}$$

Assume that $P_n(R)$ is a trapping region for the n-dimensional Galerkin projection of (2.16) *for all $n > M_1$. Then*

1. ***Uniform convergence and existence**. For a fixed $x_0 \in R$, let $x_n \colon [0,\infty] \to P_n(R)$ be a solution of $x' = P_n(F(x))$, $x(0) = P_n x_0$. Then x_n converges uniformly in a max-infinity norm on compact intervals to a function $x^* \colon [0,\infty] \to R$, which is a solution of* (2.16) *and $x^*(0) = x_0$. The convergence of x_n on compact time intervals is uniform with respect to $x_0 \in R$.*

2. ***Uniqueness within** R. There exists only one solution of the initial value problem* (2.16), *$x(0) = x_0$ for any $x_0 \in R$ such that $x(t) \in R$ for $t > 0$.*

3. ***Lipschitz constant**. Let $x \colon [0,\infty] \to R$ and $y \colon [0,\infty] \to R$ be solutions of* (2.16), *then*

$$|y(t) - x(t)|_{b,\infty} \leq e^{lt} |x(0) - y(0)|_{b,\infty}$$

4. ***Semidynamical system**. The map $\varphi \colon \mathbb{R}_+ \times R \to R$, where $\varphi(\cdot, x_0)$ is a unique solution of equation* (2.16) *such that $\varphi(0, x_0) = x_0$ defines a semidynamical system on R, namely*

   - *φ is continuous,*
   - *$\varphi(0,x) = x$,*
   - *$\varphi(t, \varphi(s,x)) = \varphi(t+s, x)$.*

The following Theorem is the main tool used to prove the existence of a locally attracting fixed point.

Theorem 2.6.3. [ZAKS, Thm. 8] *The same assumptions on R, F and a block decomposition H as in Theorem 2.6.2. Assume that $l < 0$.*

Then there exists a fixed point for (2.16) $x^* \in R$, *unique in R, such that for every $y \in R$*

$$|\varphi(t,y) - x^*|_{b,\infty} \leq e^{lt} |y - x^*|_{b,\infty}, \quad \text{for } t \geq 0,$$
$$\lim_{t\to\infty} \varphi(t,y) = x^*.$$

2.7 Proof of Theorem 2.1.1

Now we are ready to prove Theorem 2.1.1. The complete algorithm that we have used to prove Theorem 2.1.1 and other results (2.23) is demonstrated in Section 2.10. This proof is a prototype for any other result that is obtained using the algorithm, however each case requires construction of different sets. The sets and all the relevant numbers used in the proof of Theorem 2.1.1 are presented in Appendix .1.

Proof of Theorem 2.1.1 Let $u_0 \in C^4$ be an arbitrary initial condition, $\{a_k\}_{k\in\mathbb{Z}}$ be the Fourier coefficients of u_0, i.e. $u_0 = \sum a_k e^{ikx}$. Let $A \subset \overline{H}$ be an absorbing set for any Galerkin projection of (2.5a), which exists due to Lemma 2.4.6 (for instance W_8). Firstly, the existence of a locally attracting fixed point for (2.5) is established by constructing, using interval arithmetic, a set $\widetilde{W \oplus T} \subset \overline{H}$ satisfying the assumptions of Theorem 2.6.3. Namely $\widetilde{W \oplus T}$ is a trapping region for m-th Galerkin projection of (2.5a) for all $m > \widehat{m}$ and the logarithmic norm (2.20) is bounded from above by $l < 0$. By Theorem 2.6.3 within $\widetilde{W \oplus T}$ there exists a locally attracting fixed point for (2.5a).

Then, $V \oplus \Theta \subset \overline{H}$, an absorbing set for any Galerkin projection of (2.5a) satisfying

$$\varphi^m\left(t, P_m(V \oplus \Theta)\right) \subset P_m\left(\widetilde{W \oplus T}\right), \tag{2.21}$$

for all $t \geq \widehat{t}$ and $m > \widehat{m}$ is constructed. $V \oplus \Theta$ forms self-consistent bounds for (2.5a) and thus (2.21) is verified by rigorous integration of $V \oplus \Theta$ forward in time using Algorithm 9 presented hereafter. From (2.21) and that $V \oplus \Theta$ is an absorbing set for any Galerkin projection of (2.5a) it follows that

$$\varphi^m\left(t, P_m\left(\{a_k\}_{k\in\mathbb{Z}}\right)\right) \in P_m\left(\widetilde{W \oplus T}\right).$$

after a finite time and for all $m > \widehat{m}$. Therefore $\{a_k\}_{k\in\mathbb{Z}}$ is in the basin of attraction of the fixed point for (2.5a). $\widetilde{W \oplus T}$ and $V \oplus \Theta$ are presented in Appendix .1.

To close the proof it remains to argue that the found fixed point for (2.5a) is also a fixed point for (2.2a). There exists $C > 0$ such that the Fourier coefficients $\{a_k\}_{k\in\mathbb{Z}}$ of u_0 satisfy

$$|a_k| \leq \frac{C}{|k|^4}. \tag{2.22}$$

Let $W_0 \subset \overline{H}$ be a trapping region enclosing $\{a_k\}_{k\in\mathbb{Z}}$. $\{a_k(t)\}_{k\in\mathbb{Z}}$ the unique solution of (2.5) at any time $t > 0$ exists within W_0 due to Theorem 2.6.2 (it is verified that the logarithmic norm on W_0 is bounded, see e.g. [ZNS]) and conserves the initial regularity (2.22). $\{a_k(t)\}_{k\in\mathbb{Z}}$ at any time $t > 0$ is a classical solution of (2.2) because, by Lemma 2.2.2, the condition (2.22) suffices to $\sum a_k e^{ikx}$ and every term that appear in (2.2a) converge uniformly. Therefore the solution of (2.5) within $\widetilde{W \oplus T}$ is in fact the classical solution of (2.2), in particular, the fixed point of (2.5a) is the steady state solution of (2.2a). ∎

In the table below we present example results that we have obtained using our algorithm.

ν	$\int_0^{2\pi} \mathbf{u_0(x)\,dx}$	$\mathbf{E_0}$	ε	$\mathbf{m}$	(2.20) $\mathbf{l} <$	**1.**	**2.**	**3.**	**4.**	**5.**
$[10, 10.1]$	$7 \cdot 2\pi$	0.5	0.002	5	-8.24604	9.93	392	✓	✓	✓
$[4, 4.1]$	$2.5 \cdot 2\pi$	0.5	0.002	6	-2.52159	68.13	1963	✓	✓	✓
$[2, 2.1]$	2π	0.82	0.03	3	-0.627527	3.825	765	✓	✓	✓
1	$0.2 \cdot 2\pi$	0.25	0.0002	20	-0.0435519	340.87	419	✓	✓	✓
0.5	$0.05 \cdot 2\pi$	0.08	0.0002	20	-0.000403856	212.67	222	✓	✓	✓
0.15	0	0.22	0	40	-0.278295	32.72	–	✓	✓	
0.12	0	0.3472	0	55	13481.6	114.44	–	✓		

(2.23)

where **1.** total execution time in seconds, **2.** number of integration steps, **3.** if existence of a fixed point was proved, **4.** if the fixed point is attracting locally, **5.** if the fixed point is attracting globally. Order of the Taylor method was 6, time step length was 0.005 in all cases.

For each case we have fixed the radius of the energy absorbing ball E_0 and have chosen by random a forcing $f(x)$ which satisfies $\frac{E_f}{\nu^2} = E_0$. The forcing $f(x)$ has been defined by a finite number of modes $\{f_k\}_{|k|\leq m}$. We have added to each forcing mode f_k the uniform perturbation $[f_\varepsilon] := [-\varepsilon/2, \varepsilon/2] \times [-\varepsilon/2, \varepsilon/2]$ in order to perform a proof for a ball of functions simultaneously.

We stress the fact that provided cases are only examples, our program can attempt to prove any case. The package with the program along with the instruction and all the data from the proofs is available [Package].

2.8 Algorithm for constructing an absorbing set for any Galerkin projection of (2.5a)

The goal of this section is to present an algorithm for constructing a set $V \oplus \Theta \subset \overline{H}$, forming self-consistent bounds for (2.5a) such that $V \oplus \Theta$ is an absorbing set for any Galerkin projection of (2.5a). It is important to require that $V \oplus \Theta$ forms a self-consistent bounds for (2.5a) because in Algorithm 9 $V \oplus \Theta$ is being integrated forward in time to verify that any solution in $V \oplus \Theta$ after a finite time enters a trapping region.

To support our claim that constructed $V \oplus \Theta$ is in fact an absorbing set, in the following description we argue each estimate. We drop the indication of Galerkin projections and times. For the precise meaning reader is refereed to the proof of Lemma 2.4.6.

Input data

- $M > 0$,
- E_0, where $E_0 = \frac{E_f}{\nu^2}$,
- $\{[f_k]\}_{|k|\leq m}$ set of forcing modes perturbed by an uniform and constant perturbation $[f_\varepsilon]$, i.e. $[f_k] = f_k + [f_\varepsilon]$ for $|k| \leq m$ and $[f_k] = 0$ for $k = 0,\ |k| > m$.

Output data $V \oplus \Theta \subset \overline{H}$ forming self-consistent bounds for (2.5a).

begin

init $E := 1.01 \cdot E_0$, $\hat{\varepsilon} := 10^{-25}$.

I

- For $|k| \leq M$ set

$$(V \oplus \Theta)_k := \mathrm{Sq}\left(\frac{\hat{\varepsilon} + \left(\frac{1}{2}E + \max_{|k|\leq J} \frac{|[f_k]|}{|k|}\right)/\nu}{|k|}\right).$$

- For $|k| > M$ set

$$(V \oplus \Theta)_k := \mathrm{B}\left(\frac{\hat{\varepsilon} + \left(\frac{1}{2}E + \max_{|k|\leq J} \frac{|[f_k]|}{|k|}\right)/\nu}{|k|}\right).$$

Originating from the following estimate due to Lemma 2.4.3

$$|a_k| \le \frac{\hat{\varepsilon} + \left(\frac{1}{2}E + \max_{|k|\le J} \frac{|f_k|}{|k|}\right)/\nu}{|k|} =: \frac{C}{|k|}, \quad k \in \mathbb{Z},$$

after a finite time.

II

- For $|k| \le M$ calculate

$$b^-_{k,j} := \frac{\left(-C\sqrt{E}D + \frac{f^-_{k,j}}{|k|^{\frac{1}{2}}}\right)/\nu}{|k|^{\frac{3}{2}}}, \quad b^+_{k,j} := \frac{\left(C\sqrt{E}D + \frac{f^+_{k,j}}{|k|^{\frac{1}{2}}}\right)/\nu}{|k|^{\frac{3}{2}}}, \ j = 1, 2.$$

  Originating from the following estimate due to Lemma 2.3.3

$$|N_k| \le \frac{C\sqrt{E}D}{|k|^{-\frac{1}{2}}},$$

  where C is from Step I.

- For $|k| \le M$ set

$$(V \oplus \Theta)_k := [b^-_{k,1}, b^+_{k,1}] \times [b^-_{k,2}, b^+_{k,2}] + [-\hat{\varepsilon}, \hat{\varepsilon}]^2.$$

  Originating from the following estimate due to Lemma 2.4.2

$$a_k \in [b^-_{k,1}, b^+_{k,1}] \times [b^-_{k,2}, b^+_{k,2}] + [-\hat{\varepsilon}, \hat{\varepsilon}]^2, \quad |k| \le M,$$

  after a finite time, Lemma 2.4.8 is also used.

- For $|k| > M$ set

$$(V \oplus \Theta)_k := \mathrm{B}\left(\frac{\hat{\varepsilon} + \left(C\sqrt{E}D\right)/\nu}{|k|^{\frac{3}{2}}}\right).$$

  Originating from the following estimate due to Lemma 2.4.3

$$|a_k| \le \frac{\hat{\varepsilon} + \left(C\sqrt{E}D\right)/\nu}{|k|^{\frac{3}{2}}}, \quad |k| > M,$$

  after a finite time. Observe that $[f_k] = 0$ for $|k| > M$ and Lemma 2.4.3 is used with $M_1 := M$.

III Refine, until $V \oplus \Theta$ forms a self-consistent bounds for (2.5a), as the stopping criterion use the condition $s(\Theta) > d + p$, where d and p are from (2.15) and $s(\Theta)$ is the order of polynomial decay of the tail $\Theta = \Pi_{|k|>M} \overline{B\left(0, \frac{C(\Theta)}{|k|^{s(\Theta)}}\right)}$. To calculate $[b^-_{k,1}, b^+_{k,1}] \times [b^-_{k,2}, b^+_{k,2}]$ and $C(b)$ use the estimates derived in [SuppMat], in principle giving much more sharp bounds than energy-like estimates.

Every such refinement generates bounds that are reached by the solutions after a finite time. Moreover, to see that the procedure will stop note that at each iteration the order of polynomial decay $s(\Theta)$ is increased by 1. Using the formulas derived in [SuppMat] a bound such that $|N_k| \leq \frac{D}{|k|^{s(N)}}$ is received, where $s(N) = s(\Theta) - 1$ and therefore $s_{new}(\Theta) = s(b) = s(N) + 2 = s(\Theta) + 1$. Finally, as soon as $s_{new}(\Theta) > d + p$, stop.

end

2.9 Rigorous integration forward in time

By *rigorous numerics* we mean algorithms for estimating solutions of differential equations that operate on sets and produce sets that always contain the exact solution. Rigorous numerics for ODEs is a well established and analyzed topic. There exist a few algorithms that offer reliable computations of the solution trajectories for ODEs that are based on interval arithmetic. The approach used in this paper is based on the Lohner algorithm, presented in [Lo] , see also [ZLo]. It has made possible to prove many facts about the dynamics of certain ODEs, let us mention the Rossler equation, the Lorenz equation or the restricted n-body problem (see [ZLo], [MM], [KZ] and references therein). In the context of rigorous integration of ODEs we consider an abstract Cauchy problem

$$\begin{cases} \dot{x}(t) &= f(x(t)), \\ x(0) &= x_0. \end{cases} \tag{2.24}$$

$x \colon [0, \mathcal{T}) \to \mathbb{R}^n$, $f \colon \mathbb{R}^n \to \mathbb{R}^n$, $f \in C^\infty$. The goal of a rigorous ODEs solver is to find a set $\mathbf{x_k} \subset \mathbb{R}^n$ compact and connected such that

$$\varphi(t_k, \mathbf{x_0}) \subset \mathbf{x_k}, \tag{2.25}$$

$t_k \in [0, \mathcal{T})$, $\mathbf{x_0} \subset \mathbb{R}^n$, $\varphi(t_k, \mathbf{x_0})$ is a solution of (2.24) at time t_k.

Notation We denote by $[x]$ an *interval set* $[x] \subset \mathbb{R}^n$, $[x] = \Pi_{k=1}^n [x_k^-, x_k^+]$, $[x_k^-, x_k^+] \subset \mathbb{R}$, $-\infty < x_k^- \leq x_k^+ < \infty$, $\mathrm{mid}\,([x])$ is the middle of an interval set $[x]$ and $\mathrm{r}\,([x])$ is the rest, i.e. $[x] = \mathrm{mid}\,([x]) + \mathrm{r}\,([x])$.

There are some subtle issues regarding intervals and set representation in the Lohner algorithm, which are discussed e.g. in [ZLo]. Let us only mention that it is highly ineffective to use the interval set representation explicitly $\Pi[a_k^-, a_k^+]$ because it leads to the so-called *wrapping effect* [ZLo], large over-estimates appear and prevents us from integrating over a longer time interval. In order to

avoid those problems we do not use interval sets explicitly, but to represent sets in a suitable coordinate system we use the *doubleton* representation of sets [ZLo]

$$[x_k] + B_k \cdot [r_k] + C_k \cdot [r_0], \tag{2.26}$$

where B_k and C_k are matrices representing a coordinate systems, $[x_k]$ is an interval set, likely a single point, $[r_k]$ is an interval set that represents local errors that arise during integration, $[r_0]$ is an interval set that represents error at the beginning (diameter of a set at the beginning).

We stress the fact that we are interested in rigorous numerics for dPDEs, we develop main ideas in the following sections.

2.9.1 Algorithm for integrating rigorously dPDEs

In context of dPDEs we have to solve the following infinite system of ODEs

$$\begin{cases} \frac{dx}{dt} = P_m F(x+y), \\ \frac{dy}{dt} = Q_m F(x+y), \end{cases} \tag{2.27}$$

$x \in X_m$, $y \in Y_m$.

Following [KZ], [Z3] we will get estimates for (2.27) by considering the following differential inclusion

$$\frac{dx}{dt}(t) \in P_m F(x(t)) + \delta, \tag{2.28}$$

where $\delta \subset X_m$ describes influence of y onto $P_m F(x+y)$. We call

$$\frac{dx}{dt} = P_m F(x) \tag{2.29}$$

the m dimensional Galerkin projection of (2.27), where $m > 0$.

We also consider a Cauchy problem, with $a \in X_m$, $x_0 \in X_m$

$$\begin{cases} \frac{dx}{dt}(t) = P_m F(x(t)) + a, \\ x(0) = x_0. \end{cases} \tag{2.30}$$

Let d_{X_m}, d_{Y_m} dimensions associated with X_m and Y_m respectively. From now on we switch to a more concrete setting, which is

$$X_m := \mathbb{R}^{d_{X_m}} \text{ and } Y_m := \mathbb{R}^{d_{Y_m}},\ d_{X_m} < \infty,\ d_{Y_m} = \infty.$$

In this section we assume that the solutions of problems (2.27), (2.29) and (2.30) are defined and unique, later we will prove this.

Notation $T, T(0), T(t_1), T([0,h]) \subset Y_m$ are tails satisfying (2.19), in the context of tails, for notational purposes, the symbol $T(\cdot)$ is not used to denote a function of time, but an enclosure for a tail at the provided time. By

- $\overline{\varphi}^m(t, x_0, a)$ we denote the solution of (2.30) at a time $t > 0$ with $a \in X_m$ and an initial condition $x_0 \in X_m$,
- $\varphi^{X_m}(t, x_0, y_0)$ we denote the solution of (2.27) at a time $t > 0$, projected onto X_m with an initial condition $x_0 \in X_m$ and $y_0 \in Y_m$,
- $\varphi^m([0, h], x_0, T)$ denotes a collection of all possible values of the solution of the inclusion $\frac{dx}{dt} \in P_m F(x + T)$ on the time interval $[0, h]$ with $T \subset Y_m$ and an initial condition $x_0 \in X_m$.

Below we present all steps of the algorithm needed to rigorously integrate (2.27). In [KZ], [Z3] algorithm is given in an abstract setting, here we provide detailed description of an algorithm designed for dPDEs exclusively.

We present in Algorithm 9 steps needed to calculate rigorous bounds for the solutions of (2.27) at $t_1 = h$. The main idea is to get estimates for the solutions of each Galerkin projection of (2.27) simultaneously. For the correctness proof of Algorithm 9 we refer the reader [KZ] or [Z3], note that Algorithm 9 is a subcase of general algorithm, with the set $[W_y] \subset X_m$ chosen to be the Galerkin projection error.

Input

- a time step h,
- $[f_\varepsilon] \subset X_m$, a constant forcing perturbation,
- $[x_0] \subset X_m$, an initial finite part,
- $T(0) \subset Y_m$, an initial tail,
  $[x_0] \oplus T(0) \subset H$ forms self-consistent bounds for (2.16).

Output

- $[x_{t_1}] \subset X_m$ such that $\varphi^{X_m}(t_1, [x_0], T(0)) \subset [x_{t_1}]$, enclosure for the finite part of the solutions at the time t_1.
- $T(t_1) \subset Y_m$, an enclosure for the tail at the time t_1,
  $[x_{t_1}] \oplus T(t_1) \subset H$ forms self-consistent bounds for (2.16).

begin

1. find $T \subset Y_m$ such that $T([0, h]) \subset T$ and $[W_2] \subset X_m$ such that $\varphi^m([0, h], [x_0], T) \subset [W_2]$. Enclosure for the tail on the whole time interval $[0, h]$ and the enclosure for the collection of solutions of the differential inclusion respectively.
   $[W_2] \oplus T$ forms self-consistent bounds for (2.16),
2. calculate the *Galerkin projection error* $X_m \supset [W_y] := \{P_m F(x + T) - P_m F(x) \,|\, x \in [W_2]\}_I$,

3. do the selection $[W_y] \ni y_c := \mathrm{mid}\,([W_y])$,
4. apply the C^0 *Lohner algorithm* to solve the system of autonomous ODEs (2.30) with $a = y_c$, result is a rigorous enclosure for the solution $[\overline{x_{t_1}}] \subset X_m \colon \overline{\varphi}^m(t_1, [x_0], y_c) \subset [\overline{x_{t_1}}]$. As a mid-step the enclosure $[W_1]$ such that $\overline{\varphi}^m([0,h], [x_0], y_c) \subset [W_1]$ is calculated and returned. Refer [ZLo] for the details,
5. calculate *the perturbations vector* $X_m \supset [\delta] := [y_c - [W_y] + [f_\varepsilon]]_I$,
6. initialize the single valued vector $X_m \ni C_i := \sup|[\delta_i]|$,
7. compute the "Jacobian" matrix $\mathbb{R}^{d_{X_m} \times d_{X_m}} \ni J \colon J_{ij} \geq \begin{cases} \sup \frac{\partial F_i}{\partial x_j}([W_2], y_c) & \text{if } i = j \\ \left|\sup \frac{\partial F_i}{\partial x_j}([W_2], y_c)\right| & \text{if } i \neq j \end{cases}$,
8. perform component-wise estimates in order to calculate the set $[\Delta] \subset X_m$, $D := \int_0^h e^{J(t_1 - s)} C\, ds$, $[\Delta_i] := [-D_i, D_i]$ for $i = 1, \ldots, d_{X_m}$,
9. obtain the final rigorous bound $[x_{t_1}] \subset X_m$ for the solution of differential inclusion by combining results of the previous steps $\varphi^{X_m}(t_1, [x_0], T(0)) \subset [x_{t_1}] = [\overline{x_{t_1}}] + [\Delta], \quad [x_0] \subset X_m,\ T(0) \subset Y_m$,
10. perform rearrangements into the doubleton representation,
11. compute $T(t_1) \subset Y_m$ such that $\varphi^{Y_m}(t_1, [x_0], T(0)) \subset T(t_1)$.

end

Algorithm 6: The main algorithm

Our improvement concern steps 1 and 11 of Algorithm 9, which we describe in detail. In the Appendix .2 we included the pseudo-codes. We omit the remaining steps of Algorithm 9 that have already been described in previous works. To realize some of the elements we have used the [CAPD] package.

Step 1 of Algorithm 9. The main loop.

Definition 2.9.1. *Let $W \subset H$, W convex. We call W* the polynomial bound *if there exists numbers $M > 0$, $C > 0$, $s \geq 0$ such that*

$$|W_i| \leq \frac{C}{|i|^s}, \ |i| > M. \tag{2.31}$$

To denote the polynomial bound we use the quadruple (W, M, C, s).

Definition 2.9.2. *Let $W \oplus T \subset H$ forms a self-consistent bounds for* (2.16)*, $m > 0$ be the Galerkin projection dimension, N_k, f_k and λ_k appear on the right-hand side of* (2.16)*, $f_k = 0$ for $|k| > m$. For $k \in \mathbb{Z}$ and $i = 1, \ldots, d_1$ we define*

$$N_{k,i}^{\pm} \colon N_{k,i}^{-} \leq N_{k,i}(W \oplus T) \leq N_{k,i}^{+}, \tag{2.32a}$$

$$b_{k,i}^{\pm} := \frac{N_{k,i}^{\pm} + f_{k,i}}{-\lambda_k}, \tag{2.32b}$$

$$g_{k,i}^{\pm} := \left(T(0)_{k,i}^{\pm} - b_{k,i}^{\pm}\right) e^{\lambda_k h} + b_{k,i}^{\pm}, \tag{2.32c}$$

$$N_k := \Pi_{i=1}^{d_1}[N_{k,i}^{-}, N_{k,i}^{+}], \quad b_k := \Pi_{i=1}^{d_1}[b_{k,i}^{-}, b_{k,i}^{+}], \quad g_k := \Pi_{i=1}^{d_1}[g_{k,i}^{-}, g_{k,i}^{+}].$$

Basically, during the step 1 of Algorithm 9 we have to calculate $T \subset Y_m$ a good enclosure for the tail during the whole time interval $[0, h]$, i.e. T has to satisfy $T([0, h]) \subset T$. In our approach, to estimate the $T([0, h])$ we use a linear approximation of $T(h)$, which combines (2.32c) and the monotonicity of bounds, i.e. $T([0, h])_k \subset T(0)_k \cup g_k$, see [Z3, Lemma 23].

We present procedures dealing with $T_{\mathcal{N}}$ and $T_{\mathcal{F}}$ in Algorithm 7 and Algorithm 8, found in Appendix .2, separately for better clarification. For the exact meaning of the symbols refer Definition 2.5.5. The crucial part of Step 1 of Algorithm 9 is to verify if $T_{\mathcal{F}}([0, h]) \subset T_{\mathcal{F}}$ in a finite number of steps, where $T_{\mathcal{F}}$ is a candidate for the far tail.

Now, let us present how we realize this. Our goal is to enclose the interval sets g_k by a uniform polynomial bound. Once we have a uniform polynomial bound, denoted by g, verification of $T_{\mathcal{F}}([0, h]) \subset T_{\mathcal{F}}$ is straightforward. Firstly, given a polynomial bound

$$(W \oplus T, M_T, C_T, s_T) \tag{2.33}$$

a polynomial bound

$$(N, M_T, C_N, s_N) \text{ such that } \Pi_{k \in \mathbb{Z}} N_k \subset N$$

is found. This task requires some tedious estimates and we do not present them here, we have derived the required estimates for a class of dPDEs including the viscous Burgers equation in [SuppMat]. Generally, a polynomial bound satisfying $s_N = s_T - r$ is found. Then we get immediately a polynomial bound

$$(b, M_T, C_b, s_b) \text{ such that } \Pi_{k \in \mathbb{Z}} b_k \subset b, \tag{2.34}$$

with $C_b = \frac{C_N}{V(M)}$, $V(M) = \inf\{\nu(|k|) \colon k \in \mathcal{I},\ |k| > M\}$ and $s_b = s_N + p$. Finally, a polynomial bound

$$(g, M_T, C_g, s_g) \text{ such that } \Pi_{k \in \mathbb{Z}} g_k \subset g \tag{2.35}$$

is obtained using the formulas as follows

Lemma 2.9.3. *If $|k| > M$ then*

$$|g_k| \leq \frac{C_{T(0)} e^{\lambda_k h} \cdot |k|^{s_b - s_{T(0)}} - C_b(e^{\lambda_k h} - 1)}{|k|^{s_b}} \tag{2.36}$$

and

$$|g_k| \leq \frac{C_{T(0)} \cdot e^{\lambda_{k_{max}} h} (k_{max})^{s_b - s_{T(0)}} + C_b}{|k|^{s_b}} =: \frac{C_g}{|k|^{s_g}} \tag{2.37}$$

where k_{max} is the k for which function $e^{\lambda_k h} \cdot k^r$ attains its maximum.

Proof Maximum of $f(k) = e^{\lambda_k h} k^r$, with $\operatorname{dom} f = \{k \colon |k| > M\}$ is reached at k_{max}, therefore (2.36) is estimated by (2.37) for any $|k| \geq M$. ∎

Note that

$$s_g > s_T \tag{2.38}$$

because $s_N = s_T - r$, $s_g = s_b = s_T - r + p$ and $p > r$.

The main loop

Input $\left([x_0] \oplus T(0), M_{T(0)}, C_{T(0)}, s_{T(0)}\right)$ a polynomial bound, $[x_0] \oplus T(0) \subset H$ forms self-consistent bounds for (2.16).

Output $([W_2] \oplus T, M_T, C_T, s_T)$ a polynomial bound such that $T([0,h]) \subset T$, $\varphi^m([0,h],[x_0],T) \subset [W_2]$ and $[W_2] \oplus T \subset H$ forms self-consistent bounds for (2.16).

begin

1. Initialize $T := T(0)$.
2. Update $T_{\mathcal{F}}$ using `findS` function.
3. `while not` *validated*
   - $[W_2] :=$ `enclosure`$([x_0], T)$, calculate a rough-enclosure $[W_2]$ for the differential inclusion (2.28) using a current candidate for tail enclosure T, after this step $\varphi^m([0,h],[x_0],T) \subset [W_2]$ holds,
   - calculate the polynomial bounds (b, M_T, C_b, s_b) and (g, M_T, C_g, s_g),
   - *validated* := `validateTail`$(T(0),\ T,\ b,\ g,\ [W_2])$ (if T was changed during this step *validated*=`false`).

`end while`

end

Remark 2.9.4. In our algorithm the number M_T in (2.33) is being chosen adaptively in `validateFarTail` and changes from step to step.

Where `enclosure` is *the rough enclosure algorithm based on isolation*, designed for dPDEs presented in [Z3].

Regarding a correctness proof of the `validateNearTail` and `validateFarTail` functions, namely that a T such that the condition $T([0,h])_k \subset T_k$ holds for all $k \in \mathbb{Z}$ is ever returned, refer the comments within the code listings in Appendix .2.

Now, we shall focus on explaining the main idea behind `validateFarTail` and explain why we consider it as an improvement of the existing algorithm. Basically, when a $-\lambda_k$ in (2.16) is small, the nonlinear part N_k dominates the linear term. However, there exists an index $\tilde{k} \in \mathbb{N}$ such that $-\lambda_k$ for $|k| > \tilde{k}$ becomes large enough to make the linear part overtake the nonlinear part. The position of a threshold $\tilde{k}$ depends on the maximum order of the "Laplacian" that appears in the linear part L of (2.16), as well as on the order of the polynomial that appears in the nonlinear part. We remark that the solution of the m-th Galerkin projection of (2.16) with $m < \tilde{k}$ greatly differs from the solution of the whole system (2.16).

The aforementioned effects show that a proper choice of the Galerkin projection dimension m (in our algorithm taken only once at the beginning) and the number M_T of the polynomial bound (2.33) (in our algorithm taken at each timestep) is of critical importance and has to be done carefully. The application of a too small value may result in blow-ups and may prevent completion of the calculations. In the original algorithm from [Z3] the number M_T was fixed in advance. Then heuristic formulas were derived for the KS equation in order to predict if the tail validation function will finish successfully for a given M_T, s_T and to guess the initial values of C_T and s_T in (2.33), see [Z3, Section 8]. We found the original approach insufficient for the purpose of integrating rigorously PDEs that are the subject of our research (for example the Burgers or the Navier-Stokes equations). A similar approach for the mentioned dPDEs is problematic and in the case of lower viscosities especially, heuristic formulas cause performance issues and sometimes offer unfeasible values, mainly due to the lower order of the "Laplacian" in the linear part.

Step 11 of Algorithm 9

Input $([W_2] \oplus T, M_T, C_T, s_T)$, a polynomial bound such that $T([0,h]) \subset T$, $\varphi^m([0,h],[x_0],T) \subset [W_2]$ and $[W_2] \oplus T \subset H$ forms self-consistent bounds for (2.16).

Output $\big(T(h), M_{T(h)}, C_{T(h)}, s_{T(h)}\big)$, a polynomial bound.

begin

1. $M_{T(h)} := M_T$, $T(h)$ inherits M from the enclosure T,
2. calculate the polynomial bound (g, M_T, C_g, s_g),
3. $T(h) := g$, $C_{T(h)} := C_g$, $s_{T(h)} := s_g$.

end

2.10 Algorithm for proving Theorem 2.1.1

Notation By a capital letter we denote *a single valued matrix*, e.g. A, by $[A]$ we denote an *interval matrix*. The inverse matrix of A is denoted by A^{-1}, we use the symbol $[A^{-1}]$ to denote an interval matrix such that $[A^{-1}] \ni A^{-1}$. $[M]_I$ denotes an interval hull of a matrix M, we also use this notation in the context of vectors.

Input

- $m > 0$, an integer, the Galerkin projection (2.6) dimension,
- $[\nu_1, \nu_2] > 0$, an interval of the viscosity constant values,
- $\mathrm{Re}\,(a_0)$, constant value, equal to $\frac{1}{2\pi}\int_0^{2\pi} u_0(x)\,dx$,
- s, the order of polynomial decay of coefficients that is required from the constructed bounds and trapping regions, have to be an integer satisfying $s \geq 4$,
- order and the time step of the Taylor method used by the C^0 Lohner algorithm,
- set of 2π periodic forcing functions $f(x)$ for (2.2), defined by a finite number of modes $\{f_k\}_{|k|\leq m}$ and a uniform and constant perturbation $[f_\varepsilon] = [-\varepsilon/2, \varepsilon/2] \times [-\varepsilon/2, \varepsilon/2]$.

Output

- $\overline{x}$, an approximate fixed point for (2.5a),
- $J \approx dP_m F(\overline{x})$, an approximate Jacobian matrix at $\overline{x}$,
- $[A]$ and $[A^{-1}]$ interval matrices reducing $[dP_m F(\overline{x})]_I$ to an almost diagonal matrix $[D]$ - with dominating blocks on the diagonal,
- $[D] = \left[[A] \cdot [dP_m F(\overline{x})]_I \cdot [A^{-1}]\right]_I$,
- $W \oplus T \subset \overline{H}$ and $\widetilde{W \oplus T} \subset \overline{H}$, trapping regions for (2.5a),
- l, upper bound of the logarithmic norm (2.20) on $\widetilde{W \oplus T}$,

- $V \oplus \Theta \subset \overline{H}$, an absorbing set forming self-consistent bounds for (2.5a),
- total time and integration steps needed to complete the proof.

begin

1. find an approximate fixed point location $\overline{x}$ by non-rigorous integration of $\dot{x} = P_m F(x)$. Refine provided candidate $\overline{x}$ using *the Newton method* iterations,
2. calculate non-rigorously the Jacobian matrix, $J \approx dP_m F(\overline{x})$ (use ν_1 as the viscosity constant in both steps),
3. non-rigorously calculate approximate orthogonal matrix S used for reducing J to an approximate upper triangular matrix T (with 1x1 and 2x2 blocks on the diagonal). Use *the QR algorithm* with multiple shifts to find S. Then find a rigorous inverse $[S^{-1}]$: $S^{-1} \in [S^{-1}]$ using the Krawczyk operator [N],
4. calculate *the eigenvectors* of T to form a block upper triangular matrix E that is used to further reduce T to almost diagonal matrix, then calculate a rigorous inverse matrix $[E^{-1}]$: $E^{-1} \in [E^{-1}]$ using the Krawczyk operator again,
5. calculate $[A] := [S \cdot E]_I$, $[A^{-1}] := [[E^{-1}] \cdot [S^{-1}]]_I$ and $[D] := \left[[A] \cdot [dP_m F(\overline{x})]_I \cdot [A^{-1}]\right]_I$. $[D]$ is in almost diagonal form, having blocks on diagonal and negligible intervals as non-diagonal elements,
6. find $W \oplus T \subset \overline{H}$ a trapping region in a block coordinates that encloses $\overline{x}$, detailed description of an algorithm performing this task is provided by [ZAKS],
7. calculate l an upper bound for the logarithmic norm on the set $[[A^{-1}] \cdot W]_I \oplus T$, for the details refer [ZAKS]. In case if $l < 0$ by Theorem 2.6.3 claim that there exists a locally attracting fixed point and the basin of attraction of the found fixed point is $W \oplus T$. One may be tempted to use the "analytical" trapping region, calculated in Section 2.3.2 for that purpose, but this is unfeasible goal in general because an analytical trapping region may simply be too large to include it into the calculation process,
8. enlarge $W \oplus T$ and return the largest calculated self-consistent bounds $\widetilde{W \oplus T} \subset \overline{H}$ such that $\widetilde{W \oplus T}$ is a trapping region, $l < 0$ and $W \oplus T \subset \widetilde{W \oplus T}$. By Theorem 2.6.3 claim that the basin of attraction of the found fixed point is $\widetilde{W \oplus T}$,
9. using the procedure from Section 2.8 calculate the absorbing set $V \oplus \Theta$,

10. rigorously integrate $V \oplus \Theta$ forward in time until $\varphi\left(t, [[A] \cdot V]_I \oplus \Theta\right) \subset \widetilde{W \oplus T}$. If this step finishes successfully conclude that $\widetilde{W \oplus T}$ after a finite time contains any solution of the problem (2.5) with sufficiently smooth initial data and claim existence of a globally attracting fixed point,

11. translate $[[A^{-1}] \cdot W]_I \oplus T$ into the doubleton representation (2.26) and integrate forward in time in order to estimate the fixed point location with a relatively high accuracy.

end

Remark 2.10.1. All the trapping regions constructed in the main algorithm presented above have been expressed in block coordinates. Where the block decomposition of H is given by $H = \oplus_{(i)} H_i$, where for $(i) > m$ blocks are given by $H_{(i)} =< e_i >$, whereas for $(i) \leq m$ each block $H_{(i)}$ is *an eigenspace* of J. Therefore given a trapping region $W \oplus T \subset H$ the finite part W has the following form

$$W = \prod_{(i)} \begin{cases} \overline{B}\left(0, r_i\right) & \text{, for } (i) \in \mathcal{I}, \\ [a_i^-,\ a_i^+] & \text{, for } (i) \notin \mathcal{I}, \end{cases}$$

where $\mathcal{I} = \left\{(i) \colon H_{(i)} \text{ is two dimensional eigenspace of } J\right\}$.

Remark 2.10.2. In all the proofs presented in Section 2.7 we have got

$$\mathcal{I} = \begin{cases} \emptyset & \text{when } \int_0^{2\pi} u_0(x)\, dx = 0, \\ \{(i) \colon (i) \leq m\} & \text{when } \int_0^{2\pi} u_0(x)\, dx \neq 0. \end{cases}$$

We have not been able of proving this rigorously.

2.11 Conclusion

A method of proving the existence of globally attracting fixed points for a class of dissipative PDEs has been presented. A detailed case study of the viscous Burgers equation with constant in time forcing function has been given. All the computer program sources used are available online [Package]. There are several paths for the future development of the presented method we would like to suggest. An option is, for instance, to apply a technique for splitting of sets in order to see what is the largest domain approachable by this technique. One may also consider working on proving the statement given in Remark 2.10.2. Another, very interesting indeed, possibility is application to higher dimensional PDEs, like the Navier-Stokes equation, and we will address this topic in our forthcoming papers.

.1 Data from the example proof

The parameters were as follows $\nu \in [2, 2.1]$ (the whole interval was inserted), $\widehat{m} = 3$, $a_0 = 1$. To present the following data all the numbers were truncated,

for more precise data we refer the reader to the package with data from proofs available [Package].

The change of coordinates

$$\operatorname{mid}([A]) \approxeq \begin{bmatrix} -0.0174 & -5.27\cdot10^{-3} & -5.89\cdot10^{-4} & -3.76\cdot10^{-4} & -0.078 & -0.997 \\ -5.11\cdot10^{-3} & 0.0172 & -2.35\cdot10^{-4} & 6.96\cdot10^{-4} & -0.997 & 0.078 \\ -0.993 & 0.122 & -7.66\cdot10^{-3} & -3.6\cdot10^{-3} & -1.41\cdot10^{-3} & -5.84\cdot10^{-3} \\ 0.111 & 0.9 & 2.8\cdot10^{-3} & -6.56\cdot10^{-3} & -5.3\cdot10^{-3} & 1.35\cdot10^{-3} \\ -0.0116 & -0.013 & 0.0899 & -0.996 & 5.92\cdot10^{-4} & -1.88\cdot10^{-4} \\ 0.0116 & -9.95\cdot10^{-3} & -0.996 & -0.0899 & -5.23\cdot10^{-5} & -3.43\cdot10^{-4} \end{bmatrix}.$$

$$\operatorname{r}([A]) \approxeq [-1,1]\cdot \begin{bmatrix} 10.4\cdot10^{-18} & 2.6\cdot10^{-18} & 2.17\cdot10^{-19} & 1.63\cdot10^{-19} & 4.16\cdot10^{-17} & 3.33\cdot10^{-16} \\ 3.47\cdot10^{-18} & 6.94\cdot10^{-18} & 8.13\cdot10^{-20} & 2.17\cdot10^{-19} & 2.22\cdot10^{-16} & 2.78\cdot10^{-17} \\ 1.11\cdot10^{-16} & 1.39\cdot10^{-17} & 17.3\cdot10^{-19} & 13\cdot10^{-19} & 2.17\cdot10^{-19} & 8.67\cdot10^{-19} \\ 2.78\cdot10^{-17} & 2.22\cdot10^{-16} & 1.73\cdot10^{-18} & 1.73\cdot10^{-18} & 17.3\cdot10^{-19} & 4.34\cdot10^{-19} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1.73\cdot10^{-18} & 1.73\cdot10^{-18} & 1.11\cdot10^{-16} & 1.39\cdot10^{-17} & 6.78\cdot10^{-21} & 5.42\cdot10^{-20} \end{bmatrix}.$$

$$\operatorname{mid}\left([A^{-1}]\right) \approxeq \begin{bmatrix} 6.01\cdot10^{-3} & 2.2\cdot10^{-4} & -0.992 & 0.135 & 1.95\cdot10^{-3} & 8.19\cdot10^{-3} \\ 2.91\cdot10^{-4} & -6.01\cdot10^{-3} & 0.123 & 1.09 & -7.79\cdot10^{-3} & 1.44\cdot10^{-3} \\ 4.05\cdot10^{-4} & 1.35\cdot10^{-4} & -0.0136 & -7.86\cdot10^{-3} & 0.09 & -0.996 \\ 1.05\cdot10^{-4} & -5.19\cdot10^{-4} & 8.72\cdot10^{-3} & -0.0165 & -0.996 & -0.09 \\ -0.078 & -0.997 & 8.46\cdot10^{-3} & 0.0175 & -8.3\cdot10^{-4} & 1.91\cdot10^{-4} \\ -0.997 & 0.078 & 0.016 & -9.48\cdot10^{-3} & 3.94\cdot10^{-4} & 4.57\cdot10^{-4} \end{bmatrix}.$$

$$\operatorname{r}\left([A^{-1}]\right) \approxeq [-1,1]\cdot \begin{bmatrix} 1.18\cdot10^{-15} & 1.17\cdot10^{-15} & 2.11\cdot10^{-15} & 1.53\cdot10^{-15} & 1.22\cdot10^{-15} & 1.21\cdot10^{-15} \\ 8.4\cdot10^{-16} & 8.46\cdot10^{-16} & 9.85\cdot10^{-16} & 2.22\cdot10^{-15} & 8.69\cdot10^{-16} & 8.73\cdot10^{-16} \\ 6\cdot10^{-16} & 5.99\cdot10^{-16} & 6.25\cdot10^{-16} & 6.63\cdot10^{-16} & 6.8\cdot10^{-16} & 1.55\cdot10^{-15} \\ 3.51\cdot10^{-16} & 3.52\cdot10^{-16} & 3.73\cdot10^{-16} & 4.2\cdot10^{-16} & 8.88\cdot10^{-16} & 4.58\cdot10^{-16} \\ 8.88\cdot10^{-16} & 1.67\cdot10^{-15} & 7.93\cdot10^{-16} & 8.92\cdot10^{-16} & 7.75\cdot10^{-16} & 7.71\cdot10^{-16} \\ 1.89\cdot10^{-15} & 8.74\cdot10^{-16} & 7.88\cdot10^{-16} & 8.31\cdot10^{-16} & 7.56\cdot10^{-16} & 7.54\cdot10^{-16} \end{bmatrix}.$$

The Jacobian matrix in almost diagonal form

$$\operatorname{mid}\left(\left[[A]\cdot[dF(\overline{x})]_I\cdot[A^{-1}]\right]_I\right)$$

$$\approxeq \begin{bmatrix} -18.4 & -3 & 0 & 0 & 0 & 0 \\ 3 & -18.4 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2.05 & -0.995 & 0 & 0 \\ 0 & 0 & 0.995 & -2.05 & 0 & 0 \\ 0 & 0 & 0 & 0 & -8.2 & -2 \\ 0 & 0 & 0 & 0 & 2 & -8.2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0.00664 & -3.24\cdot10^{-3} & 0.832\cdot10^{-4} & 0.952\cdot10^{-4} \\ 0 & 0 & 2.87\cdot10^{-3} & 0.00726 & -2.36\cdot10^{-4} & 0.36\cdot10^{-4} \\ -23.7\cdot10^{-4} & -3.81\cdot10^{-4} & 0 & 0 & -4.33\cdot10^{-4} & -11.9\cdot10^{-4} \\ 0.372\cdot10^{-3} & -21.6\cdot10^{-4} & 0 & 0 & -10.2\cdot10^{-4} & 0.33\cdot10^{-3} \\ -4.64\cdot10^{-5} & 1.63\cdot10^{-4} & 1.48\cdot10^{-3} & -2.37\cdot10^{-3} & 0 & 0 \\ -0.765\cdot10^{-4} & 0.301\cdot10^{-5} & -1.91\cdot10^{-3} & -1.4\cdot10^{-3} & 0 & 0 \end{bmatrix}.$$

$$\operatorname{r}\left(\left[[A]\cdot[dF(\overline{x})]_I\cdot[A^{-1}]\right]_I\right) \approxeq [-1,1]\cdot \begin{bmatrix} 4.5\cdot10^{-1} & 7\cdot10^{-2} & 8.37\cdot10^{-3} & 5.28\cdot10^{-3} & 2.95\cdot10^{-4} & 3.43\cdot10^{-4} \\ 7\cdot10^{-2} & 4.5\cdot10^{-1} & 4.72\cdot10^{-3} & 9.15\cdot10^{-3} & 5.37\cdot10^{-4} & 1.64\cdot10^{-4} \\ 2.97\cdot10^{-3} & 8.87\cdot10^{-4} & 5.01\cdot10^{-2} & 1.35\cdot10^{-2} & 1\cdot10^{-3} & 2.01\cdot10^{-3} \\ 8.37\cdot10^{-4} & 2.7\cdot10^{-3} & 1.11\cdot10^{-2} & 5.01\cdot10^{-2} & 1.72\cdot10^{-3} & 7.87\cdot10^{-4} \\ 1.37\cdot10^{-4} & 3.82\cdot10^{-4} & 2.64\cdot10^{-3} & 4.22\cdot10^{-3} & 2\cdot10^{-1} & 3.58\cdot10^{-2} \\ 2.42\cdot10^{-4} & 7.49\cdot10^{-5} & 3.51\cdot10^{-3} & 2.49\cdot10^{-3} & 3.58\cdot10^{-2} & 2\cdot10^{-1} \end{bmatrix}.$$

Note that the matrix $\left[[A]\cdot[dF(\overline{x})]_I\cdot[A^{-1}]\right]_I$ does not have negligible intervals beyond the diagonal blocks. This is because we have performed the calculations for all the values $\nu \in [2, 2.1]$ simultaneously. If we perform the same calculations for one particular value of ν we would get a thin matrix with intervals of diameter $\sim 10^{-15}$.

The approximate eigenvalues $\operatorname{spect}(J) \approx$

$$\approx (-17.9983 + 3.00022i, -17.9983 - 3.00022i, -2.00094 + 0.994658i, -2.00094 - 0.994658i, -8.0008 + 1.9996i, -8.0008 - 1.9996i)\,.$$

The logarithmic norm upper bound (2.20) $l = -0.627527$.

The trapping region expressed in canonical coordinates $[[A^{-1}] \cdot \widetilde{W]_I \oplus T} \approxeq$

$\mathbf{k}$	$\mathrm{Re}\,(\mathbf{a_k})$	$\mathrm{Im}\,(\mathbf{a_k})$
1	$2.175505333 \cdot 10^{-3} + [-0.1113630596, 0.1113630596]$	$-9.889351992 \cdot 10^{-4} + [-0.1200348588, 0.1200348588]$
2	$0.09414877015 + [-7.163075298, 7.163075298]10^{-2}$	$-0.02352918861 + [-7.20035396, 7.20035396]10^{-2}$
3	$8.980703016 \cdot 10^{-3} + [-3.753720902, 3.753720902]10^{-2}$	$0.054028513 + [-3.74824719, 3.74824719]10^{-2}$
4	$-4.809966103 \cdot 10^{-4} + [-3.643109148, 3.643109148]10^{-3}$	$-7.742142648 \cdot 10^{-4} + [-3.469242401, 3.469242401]10^{-3}$
5	$6.870824445 \cdot 10^{-4} + [-1.276450603, 1.276450603]10^{-3}$	$-3.684198252 \cdot 10^{-4} + [-1.466987393, 1.466987393]10^{-3}$
6	$6.078976175 \cdot 10^{-5} + [-4.51263483, 4.51263483]10^{-4}$	$1.748399165 \cdot 10^{-4} + [-4.080727559, 4.080727559]10^{-4}$
> 7	$\leq 2.543790376/k^4$	

The absorbing set $V \oplus \Theta \approxeq$

$\mathbf{k}$	$\mathrm{Re}\,(\mathbf{a_k})$	$\mathrm{Im}\,(\mathbf{a_k})$
1	$0 + [-0.2229946985, 0.2229946985]$	$0 + [-0.2211861679, 0.2211861679]$
2	$0.09924912703 + [-6.921420805, 6.921420805]10^{-2}$	$0 + [-8.876986926, 8.876986926]10^{-2}$
3	$0 + [-3.597023059, 3.597023059]10^{-2}$	$0.05489946085 + [-2.865567281, 2.865567281]10^{-2}$
4	$4.455707269 \cdot 10^{-6} + [-7.251617173, 7.251617173]10^{-3}$	$-9.142322695 \cdot 10^{-4} + [-6.725541727, 6.725541727]10^{-3}$
5	$7.427338196 \cdot 10^{-4} + [-3.040947363, 3.040947363]10^{-3}$	$-9.324138683 \cdot 10^{-18} + [-3.405297261, 3.405297261]10^{-3}$
6	$-1.785394081 \cdot 10^{-5} + [-1.885458915, 1.885458915]10^{-3}$	$1.596958676 \cdot 10^{-4} + [-1.835094313, 1.835094313]10^{-3}$
> 7	$\leq 35.01037204/k^4$	

The absorbing set is apparently larger than the trapping region, it has been necessary to integrate it rigorously forward in time. The Taylor method used in the C^0 Lohner algorithm was of order 6 with time step 0.005. Total execution time was 6.88 seconds, total number of integration steps needed to verify that $\varphi(V \oplus \Theta) \subset \widetilde{W \oplus T}$ (having in mind that the sets are expressed in different coordinates) was 765, therefore $\widehat{t} = 3.825$.

.2 Validate tail function in pseudo-code

Here we present a pseudo-code of the functions `validateNearTail` and `validateFarTail` used in Section 2.9.1. First, we present the internal representation of sets that was used in actual program, written in C++ programming language and available at [Package].

Data representation

- *double* is a floating point number of double precision in *C++ programming language*,
- *interval* is $[a^-, a^+] \subset \mathbb{R}$ where a^-, a^+ are *double* numbers. All arithmetic operations on such intervals are rigorous and are performed using implementation of the CAPD library [CAPD]. It is verified that the interval arithmetic provides proper in mathematical sense results [N],
- *Vector* represents an interval set, a vector composed of *intervals*,
- *PolyBd* is a structure used for representing a polynomial bound (W, M, C, s). A given *PolyBd* V contains a *Vector* representing the finite part of $W \subset H$, an *integer* representing M denoted by $M(V)$ and two *intervals* representing C and s denoted by $C(V)$ and $s(V)$ respectively.

Below, in Algorithm 7 and Algorithm 8, we present functions `validateNearTail` and `validateFarTail` respectively along with correlated functions. Wherever *previous* keyword appear the value from the previous step is used.

Function: predictM

Input: *PolyBd* T, *PolyBd* g

$L := \left(\frac{C(T)}{C(g)}\right)^{s(T)-s(g)}$;

return L;

Function: correctM

Input: *PolyBd* T, *PolyBd* $T(0)$, *double* L

// function corrects current dimension M of tails in two cases: value of L is increasing

if $L >$*previous* L **then**

  if L *is sufficiently small* **then** $M(T) := M(T(0)) := L_d$;

  else $M(T) := M(T(0)) := M$;

end

/* and test if M can be decreased, by checking if L have established, by comparing approximation of current and previous L up to the order 10^2 */

if `truncate`*(L, 2)* = `truncate`*(previous L, 2)* **then**

  $M(T) := M(T(0)) := L$;

end

Function: findS

Input: *PolyBd* $T(0)$, *PolyBd* T, *Vector* W_2

/* heuristic function, tries to find optimal $s(T)$ at each iteration of the main loop. By optimal $s(T)$ we mean largest possible value such that a predicted M is within desired range. We recall that we start with s(T)=s(T(0)) */

PolyBd $g := g(T(0), T, W_2)$;

currentM := `predictM`(T, g);

potentialM := `predictM`(T, g with decreased s);

while *currentM out of desired range* **and** $s(T) > p + d$ **and** *potentialM* $> 2m$ **do**

  currentM := `predictM`(T, g);

  potentialM := `predictM`(T, g with decreased s);

  $C(T) := C(\text{previous } T) \cdot (M+1)^{s(T)-s(\text{previous}T)}$;

  $s(T) := s(T) - 1$;

end

if $s(T)$ *!= previous* $s(T)$ **then**

`correctM`(*currentM*);

Function: update
Input: *PolyBd* T, *PolyBd* T', *set* $\mathcal{I}$
for i: $i \in \mathcal{I}$ **do**
  if $T_i' \not\subseteq T_i$ **then**
    `calculate new` T_i: $T_i' \subset$ `new` T_i;
    T_i :=`new` T_i;
  end
end

Input: *PolyBd* $T(0)$, *PolyBd* T, *PolyBd* g, *Vector* W_2
Output: *bool*
`/* individually verify condition` $T(0)_i \cup g_i \subset T_i$ `*/`
$vector < bool > inflatesRe$;
$vector < bool > inflatesIm$;
for $k := m+1, \ldots, M$ **do**
  if *!(*$\mathrm{Re}\,(b_k)^+ \leq \mathrm{Re}\,(T(0)_k)^+$*)* **and** *!(*$\mathrm{Re}\,(T_k)^+ > \mathrm{Re}\,(g_k)^+$*)* **then**
    $\mathrm{Re}\,(T_k^+) := \mathrm{Re}\,(g_k^+)$;
    $inflateRe := true$;
  end
  if *!(*$\mathrm{Re}\,(b_k)^- \geq \mathrm{Re}\,(T(0)_k)^-$*)* **and** *!(*$\mathrm{Re}\,(T_k)^- < \mathrm{Re}\,(g_k)^-$*)* **then**
    $\mathrm{Re}\,(T_k^-) := \mathrm{Re}\,(g_k^-)$;
    $inflateRe := true$;
  end
  if *!(*$\mathrm{Im}\,(b_k)^+ \leq \mathrm{Im}\,(T(0)_k)^+$*)* **and** *!(*$\mathrm{Im}\,(T_k)^+ > \mathrm{Im}\,(g_k)^+$*)* **then**
    $\mathrm{Im}\,(T_k^+) := \mathrm{Im}\,(g_k^+)$;
    $inflateIm := true$;
  end
  if *!(*$\mathrm{Im}\,(b_k)^- \geq \mathrm{Im}\,(T(0)_k)^-$*)* **and** *!(*$\mathrm{Im}\,(T_k)^- < \mathrm{Im}\,(g_k)^-$*)* **then**
    $\mathrm{Im}\,(T_k)^- := \mathrm{Im}\,(g_k)^-$;
    $inflateIm := true$;
  end
  if $inflateRe$ **then**
    `inflate`($\mathrm{Re}\,(T_k)$, $1 + c_{inflate}$);
    for $j := -c_{radius}, \ldots, c_{radius}$ **do**
      $inflatesRe[k+j] := inflatesRe[k+j] + 1 + c_{inflate}/|j|$;
    end
  end
  if $inflateIm$ **then**
    `inflate`($\mathrm{Im}\,(T_k)$, $1 + c_{inflate}$);
    for $j := -c_{radius}, \ldots, c_{radius}$ **do**
      $inflatesIm[k+j] := inflatesIm[k+j] + 1 + c_{inflate}/|j|$;
    end
  end
end
for $k := m+1, \ldots, M$ **do**
  if $inflatesRe[k] > 0$ **then**
    `inflate`($\mathrm{Re}\,(T_k)$, $inflatesRe[k]$);
  end
  if $inflatesIm[k] > 0$ **then**
    `inflate`($\mathrm{Im}\,(T_k)$, $inflatesIm[k]$);
  end
end

Input: *PolyBd* $T(0)$, *PolyBd* T, *PolyBd* b, *PolyBd* g, *Vector* W_2
Output: *bool*
$L := \left(\frac{C(T)}{C(g)}\right)^{\frac{1}{s(T)-s(g)}}$; $L_2 := \left\lceil \left(\frac{C(b)}{C(T(0))}\right)^{\frac{1}{s(b)-s(T0)}} \right\rceil$;

Case 1 $s(b) > s(T(0))$
if $T(0)_{M+1} \subset b_{M+1}$ **then** // in particular $T(0)_{M+1} \subset g_{M+1} \subset b_{M+1}$
  if $L_2 < M+1$ **then** throw$(exception)$;
  if $T_{M+1} \subsetneq g_{M+1}$ **then** update$(T, g, \{M+1, M+2, \dots\})$;
  if $L_2 < \infty$ **then**
    if $T_{L_2} \subsetneq T(0)_{L_2}$ **then** update$(T, T(0), \{L_2, L_2+1, \dots\})$;
  end
  /* If $L_2 = \infty$ it is enough to validate T_{M+1} only, because $s_g > s_T$, see (2.38). If $L_2 < \infty$, T_{M+1} is validated to cover the finite number of indices $\{M+1, \dots, L_2\}$ and then validating T_{L_2} covers the infinite rest $\{L_2, L_2+1, \dots\}$ due to $s(T) \le s(T(0))$, see findS function. */
  if *T was updated* **then** correctM$(T, T(0), L)$;
else // $b_{M+1} \subsetneq T(0)_{M+1}$, in particular $b_{M+1} \subsetneq g_{M+1} \subsetneq T(0)_{M+1}$
  if $T_{M+1} \subsetneq T(0)_{M+1}$ **then** update$(T, T(0), \{M+1, M+2, \dots\})$;
  /* It is enough to validate T_{M+1} only, because $s(T) \le s(T(0))$ and $b_i \subsetneq T(0)_i$ for all $i > M$. */
end

Case 2 $s(b) = s(T(0))$
if $b_{M+1} \subseteq T(0)_{M+1}$ **then**
  if $T_{M+1} \subsetneq T(0)_{M+1}$ **then** update$(T, T(0), \{M+1, M+2, \dots\})$;
else // $T(0)_{M+1} \subsetneq b_{M+1}$
  if $T_{M+1} \subsetneq g_{M+1}$ **then**
    update$(T, g, \{M+1, M+2, \dots\})$;
    correctM$(T, T(0), L)$;
  end
end
/* In both cases it is enough to validate T_{M+1} because either $b_i \subset T(0)_i$ for all $i > M$ or $T(0)_i \subset b_i$ for all $i > M$ and $s(T(0)) = s(b) = s(g) \ge s(T)$, see (2.38). */

Case 3 $s(b) < s(T(0))$
if $b_{M+1} \subset T(0)_{M+1}$ **then**
  if $L_2 < M+1$ **then** throw$(exception)$;
  if $T_{M+1} \subsetneq T(0)_{M+1}$ **then** update$(T, T(0), \{M+1, M+2, \dots\})$;
  if $L_2 < \infty$ **then**
    if $T_{L_2} \subsetneq g_{L_2}$ **then** update$(T, g, \{L_2, L_2+1, \dots\})$;
  end
  /* If $L_2 = \infty$ it is enough to validate T_{M+1} only, because $s(T) \le s(T(0))$. If $L_2 < \infty$, T_{M+1} is validated to cover the finite number of indices $\{M+1, \dots, L_2\}$ and validating T_{L_2} covers the infinite rest $\{L_2, L_2+1, \dots\}$ due to $s_g > s_T$, see (2.38). */
  if *T was updated* **then** correctM$(T, T(0), L)$;
else // $T(0)_{M+1} \subsetneq b_{M+1}$
  if $T_{M+1} \subsetneq g_{M+1}$ **then**
    update$(T, g, \{M+1, M+2, \dots\})$;
    correctM$(T, T(0), L)$;
  end
  /* It is enough to validate T_{M+1} only, because $T(0)_i \subsetneq b_i$ for all $i > M$ and $s_g > s_T$, see (2.38). */
end
if *T was updated* **then return** false;
else return true;

Algorithm 8: validateFarTail function

Bibliography

[Software] Software package and the data from tests, http://www.ii.uj.edu.pl/~cyranka/PhD.

[Package] Program performing computer assisted proof package, http://www.ii.uj.edu.pl/~cyranka/Burgers.

[SuppMat] Supplementary material, http://www.ii.uj.edu.pl/~cyranka/Burgers.

[AK] G. Arioli, H. Koch, *Integration of Dissipative Partial Differential Equations: A Case Study*, SIAM J. Appl. Dyn. Syst., vol. 9(2010), 1119-1133.

[BBCG] M. Berz, C. Bischof, G. F. Corliss, and A. Griewank, editors. *Computational Differentiation: Techniques, Applications, and Tools.* Philadelphia, Penn.: SIAM, 1996.

[BS] C. Bendtsen, Ole Stauning, *FADBAD, a Flexible C++ Package for Automatic Differentiation*, Department of Mathematical Modelling, Technical University of Denmark, 1996.

[C] J. Cyranka, *Existence of Globally Attracting Fixed Points of Viscous Burgers Equation with Constant Forcing. A Computer Assisted Proof*, in review.

[CAPD] CAPD - Computer Assisted Proofs in Dynamics, a package for rigorous numeric, http://capd.ii.uj.edu.pl.

[CHQZ] C. Canuto, M.Y. Hussaini, A. Quarteroni, T.A. Zang, *Spectral Methods. Fundamentals in Single Domains.* Springer, 2006.

[ES] W. E and Y. Sinai, *New results in mathematical and statistical hydrodynamics*, Uspekhi Mat. Nauk 55 (2000), vol. 4(334), 25-58.

[CT] J. W. Cooley, J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp. 19 (1965), 297-301.

[FTKS] O. Fogelklou, W. Tucker, G. Kreiss, M. Siklosi *A computer-assisted proof of the existence of solutions to a boundary value problem with an integral boundary condition*, Commun. Nonlinear Sci., vol. 16(2011), 1227-1243.

[G] A. Griewank *Evaluating derivatives. Principles and techniques of algorithmic differentiation*, Frontiers in Applied Mathematics, vol. 19. SIAM, Philadelphia (2000).

[GLM] M. Gameiro, J.P. Lessard, K. Mischaikow *Validated continuation over large parameter ranges for equilibria of PDEs*, Mathematics and Computers in Simulation, vol. 79(2008), 1368-1382.

[HK] Hofschuster, Krämer *C-XSC 2.0: A C++ Library for Extended Scientific Computing* Preprint BUW-WRSWT 2003/5, Universität Wuppertal.

[JZ] À. Jorba, M. Zou, *A Software Package for the Numerical Integration of ODEs by Means of High-Order Taylor Methods*, Experimental Mathematics, vol. 14(2005), 99-117.

[KZ2] T. Kapela, P. Zgliczyński, *The existence of simple choreographies for the N-body problem - a computer assisted proof*, Nonlinearity, vol. 16(2003), 1899-1918.

[KZ] T. Kapela, P. Zgliczyński, *A Lohner-type algorithm for control systems and ordinary differential inclusions*, Discrete Cont. Dyn. Sys. B, vol. 11(2009), 365-385.

[Lo] R.J. Lohner, *Computation of Guaranteed Enclosures for the Solutions of Ordinary Initial and Boundary Value Problems*, Computational Ordinary Differential Equations, J.R. Cash, I. Gladwell Eds., Clarendon Press, Oxford, 1992.

[Lo1] R.J. Lohner, *Einschliessung der Lösung gewonhnlicher Anfangs- and Randwertaufgaben und Anwendungen*, Universität Karlsruhe (TH), these 1988.

[MM] M. Mrozek, K. Mischaikow, *Chaos in Lorenz equations: a computer assisted proof*, Bull. Amer. Math. Soc. (N.S.), vol. 33(1995), 66-72.

[MS] J. Mattingly and Y. Sinai, *An Elementary Proof of the Existence and Uniqueness Theorem for Navier-Stokes Equations*, Comm. in Contemporary Mathematics, vol. 1(1999), 497-516.

[N] A. Neumeier, *Interval methods for system of equations.* Cambridge University Press, 1990.

[PO] S. A. Orszag, G. S. Patterson, *Spectral Calculations of Isotropic Turbulence: Efficient Removal of Aliasing Interactions*, Phys. Fluids 14, 2538 (1971); doi: 10.1063/1.1693365.

[PP] E. Phipps, R. Pawlowski, *Efficient Expression Templates for Operator Overloading-based Automatic Differentiation.* arXiv:1205.3506v1 [cs.MS] 15 May 2012.

[S] C. Simó, *Taylor method for the integration of ODE*, Lectures given at the LTI07 Advanced School on Long Time Integrations, available online http://www.maia.ub.es/dsg/2007/.

[T] C. Temperton, *Self-Sorting Mixed-Radix Fast Fourier Transforms*, J. Comput. Phys., vol. 52(1983), 1-23.

[T2] C. Temperton, *Fast Mixed-Radix Real Fourier Transforms*, J. Comput. Phys., vol. 52(1983), 340-350.

[V] J. Vukadinovic, *Global dissipativity and inertial manifolds for diffusive Burgers equations with low-wavenumber instability*, Discrete Cont. Dyn. Sys., vol. 29(2011), 327-341.

[Wh] G. B. Whitham, *Linear and Nonlinear Waves*. John Wiley & Sons, 1975.

[ZAKS] P. Zgliczyński, *Attracting fixed points for the Kuramoto-Sivashinsky equation - a computer assisted proof*, SIAM Journal on Applied Dynamical Systems, vol. 1(2002), 215-288.

[Z2] P. Zgliczyński, *Rigorous numerics for dissipative Partial Differential Equations II. Periodic orbit for the Kuramoto-Sivashinsky PDE - a computer assisted proof*, Foundations of Computational Mathematics, vol. 4(2004), 157-185.

[Z3] P. Zgliczyński, *Rigorous Numerics for Dissipative PDEs III. An effective algorithm for rigorous integration of dissipative PDEs*, Topological Methods in Nonlinear Analysis, vol. 36(2010), 197-262.

[ZLo] P. Zgliczyński, *C^1-Lohner algorithm*, Foundations of Computational Mathematics, vol. 2(2002), 429-465.

[ZM] P. Zgliczyński and K. Mischaikow, *Rigorous Numerics for Partial Differential Equations: the Kuramoto-Sivashinsky equation*, Foundations of Computational Mathematics, vol. 1(2001), 255-288.

[ZNS] P. Zgliczyński, *Trapping regions and an ODE-type proof of an existence and uniqueness for Navier-Stokes equations with periodic boundary conditions on the plane*, Univ. Iag. Acta Math., vol. 41(2003), 89-113.