CHAPTER I

INTRODUCTION

CHAPTER II

SPATIOTEMPORAL FORMULATION

2.1 Spatiotemporal Fourier Transforms

The spatiotemporal theory developed here focuses solely on the Kuramoto-Sivashinsky equation, however, the first definition provided will be a general one. For any governing equation, the application of this theory is predicated upon the existence of periodic orbits ('orbit' will be used for brevity, as everything is assumed to at least approximate a periodic orbit in this formulation). Therefore, the first step is to define what an 'orbit' is exactly. Let $\mathbf{f}$ represent a partial differential equation (vector or scalar, going with boldface to indicate vector) with all terms grouped on one side such that $\mathbf{f}(\mathbf{v}) = 0$; this will be referred to as the *governing equation*. The variable $\mathbf{v}$represents the 'state-vector', which contains all relevant independent variables; it will be defined shortly. Most of the components of $\mathbf{v}$ correspond to the field u, typically a velocity field (scalar or vector), is defined on a $d+1$-dimensional space-time configuration manifold $\mathcal{M}$. Assuming translational invariance, $\mathcal{M}$ can be defined by the Cartesian product of intervals of real numbers

$$\begin{aligned} \mathcal{M} &= [0,T] \times \prod_{i}^{d} [0, L_i] \\ &= \{(t, \mathbf{x}) = (t, x_1, ..., x_d) \mid x_i \in [0, L_i],\ \forall i = \{1, ..., d\}\}\,. \end{aligned} \tag{1}$$

Periodic orbits are then defined as differentiable fields $\mathbf{u}(t, \mathbf{x})$ which satisfy periodic boundary conditions in time $\mathbf{u}(t, \mathbf{x}) = \mathbf{u}(t+T, \mathbf{x})$. Spatial boundary conditions are not necessarily periodic, however, they are assumed to be compact after quotienting the translational symmetries. Therefore, a periodic orbit, or simply *orbit* is uniquely identified by the tuple $(\mathcal{M}, u(t, \mathbf{x}), f)$. In order these are: the configuration manifold $\mathcal{M}$, the velocity field $\mathbf{u}(t, \mathbf{x})$ defined on $\mathcal{M}$, and the governing equation f for which $\mathbf{f}(\mathbf{u}) = 0$ and $\mathbf{u}(t, \mathbf{x}) = \mathbf{u}(t + T, \mathbf{x})$ are satisfied.

For the numerical implementation all variables need to be stored in a single vector which shall be denoted as the *state vector*. Providing a state vector and specifying any symmetries is sufficient to define an orbit. The vector representation of the orbits (3) in the Fourier basis is

$$\mathbf{v} \equiv [\vec{\mathbf{u}}, \mathbf{p}]^\top\,. \tag{2}$$

Parameters which are constrained or non-applicable can simply be removed from the state vector (2) With this in mind, the following terminology is introduced. The first, the configuration manifold $\mathcal{M}$ shall be referred to as the spatiotemporal tile or simply *tile* of an orbit. The dimensions of a tile, or *tile dimensions* are simply the values $(T, L_1, L_2, ...)$. The field of an orbit refers to $\mathbf{u}(t, \mathbf{x})$. The set of all periodic orbits of an equation f is defined as

$$\mathbb{O}(\mathcal{M}, f, \mathbf{u}) \equiv \{\mathbf{u}(t, \mathbf{x}) \mid \mathbf{f}(\mathbf{v}) = 0 \quad \text{and} \quad \mathbf{u}(t, \mathbf{x}) = \mathbf{u}(t + T, \mathbf{x}) \quad \forall (t, \mathbf{x}) \in \mathcal{M}\} \tag{3}$$

The set of orbits (3) is defined in this manner because it is befitting of the object-oriented programming used for this project, `orbithunter` on Github. In other words, the properties of the abstract set (3) are captured by the 'Orbit' computational class.

As this study only focuses on the Kuramoto-Sivashinsky equation and its orbits, (3) will simply be referenced by $\mathbb{O}$.

Motivated by the translational invariance, periodic boundary conditions, and smoothness, a Fourier-Fourier basis was a natural choice for the numerical representation of $u(t, x)$. In the words of [5]:

> *'The general rule is: Geometry chooses the basis set. The engineer never has to make a choice.' – John P. Boyd*

The first step towards such a description is to discretize (3). Specifically, the invariant 2-torus defined by the tile $\mathcal{M}$ is discretized via the two-dimensional lattice (alternatively, grid or mesh) of space-time points $\mathcal{M}_{nm} = (t_n, x_m)$ where

$$\begin{aligned} t_n &= \frac{nT}{N}, \quad n \in 0, 1, \ldots, N-1 \\ x_m &= \frac{mL}{M}, \quad m \in 0, 1, \ldots, M-1\,, \end{aligned} \tag{4}$$

which has discrete periods N in time and M in space. For brevity, discretized tiles will also be referred to simply as tiles; it should be fairly obvious whether the context is continuous or discrete. In the discrete setting, a 'solution' is now a discretized field $u(t_n, x_m)$ such that at every point on the tile $u(t_n, x_m)$ solves (**??**) locally; or at least this would be the case if we remained in the physical field basis. The Fourier basis, however, is an inherently global description of space-time and (**??**) as the field is expanded in terms global, periodic basis functions. The tile now represents the set of *interpolation points* or *collocation points* where the residual function must vanish; this is getting ahead of the current discussion, however, and so it is left to sect. **??**.

These points are commonly referred to as collocation points, and this type of formulation can be classified as a *collocation method* or *pseudospectral method*. The benefits of pseudospectral methods are numerous; most importantly they increase the accuracy of our calculations while also minimizing the amount of computational memory required [5].

The inherently infinitely dimensional equations are approximated by a Galerkin truncation of these spatiotemporal Fourier modes.

Instead of using an overly verbose and repetitive verbiage for the coefficients of the Fourier basis function, i.e. *spatiotemporal Fourier modes* or *spatiotemporal Fourier coefficients*;they will simply be referred to as *modes*, unless specifically mentioned otherwise. This avoids excessive usage of '*spatiotemporal*' and '*Fourier*'; nearly everything from here on out is spatiotemporal. Likewise, *discrete spatiotemporal real valued Fourier transform* (it can get even worse) will be cut short to either *Fourier transform* or a reference to a computational function such as `rfft` from the SciPy numerical package.

The computational transforms are computed as

$$\begin{aligned} \tilde{u}_k(t_n) &= \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} u(t_n, x_m) e^{-i(2\pi km/M)} \\ u(t_n, x_m) &= \frac{1}{\sqrt{M}} \sum_{k=1}^{M/2-1} \tilde{u}_k(t_n) e^{i(2\pi km/M)} \end{aligned} \tag{5}$$

The convention that they utilize regarding the signs of the exponents are cause for confusion upon reformulation in the real-valued basis.

After the definition of tiles in place one would think that the definition of the Fourier transforms would come very naturally, which is true until some numerical details are considered. These details result in less than perfect expressions for the Fourier transforms, but they are written such that evaluating the expressions on paper will yield the exact same result as that described in sect. **??**. The first detail is that a real-valued (SO(2)) representation of the modes is required (or at very least, greatly beneficial) for this formulation. The first reason behind the choice of is numerical in nature; if all computational degree of freedom are complex numbers, then constraints are needed to ensure the dimensions (T, L) are maintained as real numbers. The second reason is that expanding in sines and cosines turns out to be very helpful for orbits with discrete spatiotemporal symmetries, due to the parity of said functions. For real valued input the negative frequency and positive frequency modes of the Fourier transform are related by conjugation relations. Due to this conjugation relation and the previously described discretization, the Fourier transform takes the form of a truncated Fourier series, whose summation ranges over the frequencies q_k, ω_j

$$\begin{aligned} q_k &= \frac{-2\pi k}{L} \quad \text{where, } k \in 1, \dots \frac{M}{2} - 1 \\ \omega_j &= \frac{-2\pi j}{L} \quad \text{where, } j \in 0, \dots \frac{N}{2} - 1 \end{aligned} \tag{6}$$

It is helpful to derive the spatiotemporal transform via the composition of one-dimensional spatial and temporal transforms. Again, it should be stressed that in the derivations that follow *everything is written to agree with output of the computational codes*. This can lead to some expressions that look unsimplified, but the idea is that calculating the modes by hand would be exactly equivalent. Let $q_k = -q_k$ and $\omega_j = -\omega_j$, the numerically implemented spatial transform (and its inverse) for real valued input take the following form

$$\begin{aligned} e_k(t_n) &= \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} u(t_n, x_m) \cos(q_k x_m) \\ f_k(t_n) &= \frac{1}{\sqrt{M}} \sum_{m=0}^{M-1} u(t_n, x_m) \sin(q_k x_m) \\ u(t_n, x_m) &= \frac{2}{\sqrt{M}} \sum_{k=1}^{\frac{M}{2}-1} e_k(t_n) \cos(q_k x_m) + f_k(t_n) \sin(q_k x_m) \,. \end{aligned} \tag{7}$$

The $k \in \{0, \frac{M}{2}\}$ spatial modes are constrained to 0; hence why they are not included in (**??**). The factor of two in the inverse transform expression is a result of keeping only the positive half of the spectrum. The constraint on the zeroth mode is justified by invariance to Galilean transformations, discussed in sect. **??**. The constraint on the Nyquist mode is a numerical choice; numerically, this mode is returned as the sum of the corresponding negative and positive modes. This, in combination with the conjugation relation, means that only the real component (or twice thereof, technically) is returned. Because of this behavior, and the fact that the magnitude is typically small for sufficiently large discretizations, this mode is excluded from calculations (i.e. discarded after both space and time transforms).

The spatiotemporal transform follows by taking the temporal transform of the time dependent spatial modes $e_k(t_n), f_k(t_n)$ of (**??**). Due to the numerical implementation, time is technically parameterized as $T - t_n$, as the first row corresponds to $t = T$ and the last

row, $t = 0$. The expression (**??**) is still correct, but now that the time ordering matters, the time transform and its inverse take the form

$$\begin{aligned}
a_{jk} &= \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} e_k(T - t_n) \cos(\omega_j t_n) \\
b_{jk} &= \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} e_k(T - t_n) \sin(\omega_j t_n) \\
c_{jk} &= \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f_k(T - t_n) \cos(\omega_j t_n) \\
d_{jk} &= \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f_k(T - t_n) \sin(\omega_j t_n) \\
e_k(T - t_n) &= \frac{2}{\sqrt{N}} \Big(\frac{a_{0k}}{2} + \sum_{j=1}^{\frac{N}{2}-1} a_{jk} \cos(\omega_j t_n) + b_{jk} \sin(\omega_j t_n) \Big) \\
f_k(T - t_n) &= \frac{2}{\sqrt{N}} \Big(\frac{c_{0k}}{2} + \sum_{j=1}^{\frac{N}{2}-1} c_{jk} \cos(\omega_j t_n) + d_{jk} \sin(\omega_j t_n) \Big) \, .
\end{aligned} \tag{8}$$

$$\begin{aligned}
a_{jk} &= \frac{1}{\sqrt{NM}} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} u(T - t_n, x_m) \cos(q_k x_m) \cos(\omega_j t_n) \\
b_{jk} &= \frac{1}{\sqrt{NM}} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} u(T - t_n, x_m) \cos(q_k x_m) \sin(\omega_j t_n) \\
c_{jk} &= \frac{1}{\sqrt{NM}} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} u(T - t_n, x_m) \sin(q_k x_m) \cos(\omega_j t_n) \\
d_{jk} &= \frac{1}{\sqrt{NM}} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} u(T - t_n, x_m) \sin(q_k x_m) \sin(\omega_j t_n) \\
u(T - t_n, x_m) &= \frac{4}{\sqrt{NM}} \sum_{k=1}^{\frac{M}{2}-1} \Bigg[\Big(\frac{a_{0k}}{2} + \sum_{j=1}^{\frac{N}{2}-1} a_{jk} \cos(\omega_j t_n) + b_{jk} \sin(\omega_j t_n) \Big) \cos(q_k x_m) \\
&\qquad + \Big(\frac{c_{0k}}{2} + \sum_{j=1}^{\frac{N}{2}-1} c_{jk} \cos(\omega_j t_n) + d_{jk} \sin(\omega_j t_n) \Big) \sin(q_k x_m) \Bigg] \, .
\end{aligned} \tag{9}$$

This is a technical detail but it is relevant when taking time derivatives; if overlooked then the time derivative would be off by a negative sign. Due to periodicity in time (8) can be rewritten to yield the expressions for the increasing-time parameterization as

$$\begin{aligned}
e_k(t_n) &= \frac{2}{\sqrt{N}} \Big(\frac{a_{0k}}{2} + \sum_{j=1}^{\frac{N}{2}-1} a_{jk} \cos(-\omega_j t_n) + b_{jk} \sin(-\omega_j t_n) \Big) \\
f_k(t_n) &= \frac{2}{\sqrt{N}} \Big(\frac{c_{0k}}{2} + \sum_{j=1}^{\frac{N}{2}-1} c_{jk} \cos(-\omega_j t_n) + d_{jk} \sin(-\omega_j t_n) \Big) \, .
\end{aligned} \tag{10}$$

The convention was chosen to use (8); this should only be viewed as a numerical parameterization of the field $u(t,x)$. This is not trying to imply anything regarding time reversal. The result of this is that an extra negative sign is required for time differentation. Again the entire point of this is that (8) more accurately represents the actual numerical calculations being performed, but (10) it is much more convenient to write the equation for $u(t_n, x_m)$, so it will be expressed in terms of the modes of $u(T-t_n, x_m)$. Finally, upon substitution of (10) into (7) spatiotemporal transform and its inverse are derived.

$$
\begin{aligned}
a_{jk} &= \frac{1}{\sqrt{NM}} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} u(t_n, x_m) \cos(q_k x_m) \cos(\tilde{\omega}_j t_n) \\
b_{jk} &= \frac{1}{\sqrt{NM}} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} u(t_n, x_m) \cos(q_k x_m) \sin(\tilde{\omega}_j t_n) \\
c_{jk} &= \frac{1}{\sqrt{NM}} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} u(t_n, x_m) \sin(q_k x_m) \cos(\tilde{\omega}_j t_n) \\
d_{jk} &= \frac{1}{\sqrt{NM}} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} u(t_n, x_m) \sin(q_k x_m) \sin(\tilde{\omega}_j t_n) \\
u(t_n, x_m) &= \frac{4}{\sqrt{NM}} \sum_{k=1}^{\frac{M}{2}-1} \left[\left(\frac{a_{0k}}{2} + \sum_{j=1}^{\frac{N}{2}-1} a_{jk} \cos(\tilde{\omega}_j t_n) + b_{jk} \sin(\tilde{\omega}_j t_n) \right) \cos(q_k x_m) \right. \\
&\qquad \left. + \left(\frac{c_{0k}}{2} + \sum_{j=1}^{\frac{N}{2}-1} c_{jk} \cos(\tilde{\omega}_j t_n) + d_{jk} \sin(\tilde{\omega}_j t_n) \right) \sin(q_k x_m) \right] \\
&\qquad \text{where, } \omega_j = -\omega_j \,. \qquad (11)
\end{aligned}
$$

Practically speaking the only difference between (11) and (9) that occurs is an extra negative sign upon time differentiation. The expressions (11) and (9) both yield the same Fourier modes; however $u(t_n, x_m)$is used in almost all expressions and so the extra -1 factor must be accounted for moving forward, most importantly for time differentiation. The spatiotemporal `rfft`(11) stands as the equation for orbits of all symmetry types; however, as shown in sect. **??**, discrete symmetries impose constraints which determine the set of nonvanishing modes. These constraints will be referred to as selection rules; their derivation is left to the discussion of spatiotemporal symmetries sect. **??**. The explicit formulae (10), (7), (11) explicitly and exactly state how the discrete Fourier transforms are defined. The numerical implementation is a *pseudospectral* implementation. The defining quality of this implementation is that the computation of the nonlinear term of the Kuramoto-Sivashinsky equation is performed as an elementwise (alternatively, Schur or Hadamard) product of fields as opposed to a convolution of modes. Computationally this provides benefits to accuracy and computation time (especially if the discretization sizes are powers of two, which FFT algorithms can exploit). Due to this, an explicit expression in terms of the modes is not required, hence (11) will only really ever be used implicitly; as a convention will be used to represent the forward and backward transforms as $\mathcal{F}$, $\mathcal{F}^{-1}$. If a transform for a single dimension needs to be referenced, it shall be done via a subscript, i.e. $\mathcal{F}_x$ refers to (7).

Finally, we have arrived at a formula which defines $u(t_n, x_m)$in terms of spatiotemporal Fourier modes. The expression (11) is very cumbersome, however, and so two other representations are explained for usage in future derivations and computations. The first of the

two representations defined is a tensor representation. The tensor arranges the four sets of modes of (11) into blocks based on frequencies

$$\mathbf{u} \equiv \begin{bmatrix} a_{0k} & c_{0k} \\ a_{jk} & c_{jk} \\ b_{jk} & d_{jk} \end{bmatrix} . \tag{12}$$

The zeroth $j = 0$ time modes are indicated separately because there are no analogous terms in b_{jk}, d_{jk}; they were discarded because the zeroth modes must be real valued. This is an important factor to note for time differentiation however for simplicity the awkwardness of the zeroth mode will be implicit in the mode tensor expression such that

$$\mathbf{u} \equiv \begin{bmatrix} a_{jk} & c_{jk} \\ b_{jk} & d_{jk} \end{bmatrix} . \tag{13}$$

In the context of the tensor notation the sets of modes $\{a_{jk}, b_{jk}, c_{jk}, d_{jk}\}$ will be referred to as *mode blocks*. The shape of the mode tensors is fundamentally different when symmetries are considered; therefore, a notation which specifies the shape of the mode tensor based on the j, k indices is developed. Defining

$$\begin{aligned} k &\in \{1, ..., M/2 - 1\} \\ j &\in \{1, ..., N/2 - 1\} \end{aligned} \tag{14}$$

While the indices of (13) are never referenced directly, due to the lack of uniqueness, the blocks' indices (**??**) are by the following notation, which generates the ordered pairs of indices for each block; the first term in the product will be called the *time indices* and the latter, *space indices* the ordered pairs specifying the elements of (13) will be defined via set notation

$$\{(j, k)\} \mathrel{\widehat{=}} \{\{\{0\}, \{j_{ac}\}, \{j_{bd}\}\} \times \{\{k_{ab}\}, \{k_{cd}\}\}\} . \tag{15}$$

where $\{j_{ac}\} = \{j_{bd}\} = \{j\}$ and $\{k_{ab}\}, \{k_{cd}\} = \{k\}$ from (14). This is simply an attempt at formalizing the fact that to uniquely specify an element in (13) the block in addition to the indices j, k must be provided. Additionally, another important usage of (15) is that the dimensions of the mode tensor can be easily deduced given the definitions of the sets (14). For example, the dimensions of the mode tensor (13) are

$$\begin{aligned} |\mathbf{u}| &= |\{\{\{0\}, \{j_{ac}\}, \{j_{bd}\}\}| \times |\{\{k_{ab}\}, \{k_{cd}\}\}\}| \\ &= (|\{0\}| + |\{j\}| + |\{j\}|) \times (|\{k\}| + |\{k\}|) \\ &= (1 + (N/2 - 1) + (N/2 - 1)) \times (M/2 - 1 + M/2 - 1) \\ &= (N - 1) \times (M - 2) \end{aligned} \tag{16}$$

It is very important to get this ordering correct in order to correctly take derivatives.

For practical purposes the non-existent $b_{(0,k)}$ modes can be assumed to equal zero in expressions such as $a_{jk} + b_{jk}$. The reason why this is being discussed at all is so that all equations reflect the numerical computations as accurately as possible, without being overly verbose.

In anticipation of future derivations one final convention concerning the dimensionality of (15) is introduced. Specifically, for the derivation of linear operators there will be Kronecker products with identity matrices of various sizes defined by the number of modes;

something which varies depending on symmetry. The following definitions allow for a general description without having to constantly specify symmetries

$$\begin{aligned} \mathbb{I}_{\frac{N}{2}-1} &\equiv \mathbb{I}_j \\ \mathbb{I}_{\frac{M}{2}-1} &\equiv \mathbb{I}_k \\ \mathbb{I}_x &\equiv \mathbb{I}_{|\{(j,k)\}|_t} \\ \mathbb{I}_t &\equiv \mathbb{I}_{|\{(j,k)\}|_x}\,. \end{aligned} \tag{17}$$

The last two identity matrices diagonals' have dimension equal to the cardinality of the set of space indices and time indices. For (16) this would imply $\mathbb{I}_t = N-1$ and $\mathbb{I}_x = M-2$

The second representation of the modes will be a vector representation. This requires two decisions to be made; how to order real and imaginary components with respect to one another and how to order space and time indices relative to each other. The conventions employed here are to separate the real and imaginary components such that they have the general form (using dummy variables) $\vec{\mathbf{u}} = [\mathrm{Re}(z_i), \mathrm{Im}(z_i)]$. The alternative is to alternate between real and imaginary components $\vec{\mathbf{u}} = [\mathrm{Re}(z_0), \mathrm{Im}(z_0), \ldots, \mathrm{Re}(z_k), \mathrm{Im}(z_k)]$. Next is how to order the vector elements with respect to space and time, i.e. the indices j, k. The convention here is to have the spatial index k as the 'inner' index. That is, cycling through the vectors' elements will cycle through all values of the inner index k for each increment of the outer index j. For those familiar with programming parlance, this would be equivalent to nested 'for-loops'. To represent this visually this can informally be written as

$$\vec{\mathbf{u}} \equiv \overbrace{\Big[\underbrace{a_{(0,k)}}_{\mathrm{Re}(x)}\ \underbrace{c_{(0,k)}}_{\mathrm{Im}(x)}\ \ldots}^{\mathrm{Re}(t)}\ \overbrace{\underbrace{b_{(1,k)}}_{\mathrm{Re}(x)}\ \underbrace{d_{(1,k)}}_{\mathrm{Im}(x)}\ \ldots\Big]}^{\mathrm{Im}(t)}\,. \tag{18}$$

The labels $\mathrm{Re}(x), \mathrm{Im}(x), \mathrm{Re}(t), \mathrm{Im}(t)$ are simply a means of demonstrating the pattern with which indices are cycled through; no formal Mathematical statement is being made in (18) other than the mode ordering.

This vector representation is used almost exclusively in the implementation of Newton's method sect. **??**, which requires explicit construction of the Jacobian. Naturally, in order to explicitly construct the Jacobian matrix, a matrix representation is required for each relevant linear operator. Therefore as the final task of this section, the matrix representations of the `rfft` and `irfft` operators $\mathcal{F}$, $\mathcal{F}^{-1}$are derived. In practice this is constructed by first constructing the matrix for a single instant in time or space, and then using a Kronecker product to extend it to space-time. In either case, the matrix is constructed by taking the `rfft` (`irfft`) of each column of appropriately sized identity matrix; this yields a Vandemonde matrix wherein the elements are powers of the roots of unity. The matrix representation of the linear operator $\mathcal{F}_x$ is

$$\begin{bmatrix} 1 & 1 & 1 & 1 & \ldots & 1 \\ 1 & r & r^2 & r^3 & \ldots & r^{(M-1)} \\ 1 & r^2 & r^4 & r^6 & \ldots & r^{2(M-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & r^{(\frac{M}{2})} & r^{2(\frac{M}{2})} & r^{3(\frac{M}{2})} & \ldots & r^{(\frac{M}{2})(M-1)} \end{bmatrix} \tag{19}$$

with $r = \cos(\frac{2\pi}{M}) + i\sin(\frac{2\pi}{M})$. The remaining steps are: to account for the constraints on $k \in \{0, \frac{M}{2}\}$, to make bring (21) to a real-valued form and to extend the definition to create an operator which acts on the entirety of space-time.

The first of these steps, accounting for the constraints, is handled by discarding the first and last rows; these correspond to the constrained modes.

$$\begin{bmatrix} 1 & r & r^2 & r^3 & \dots & r^{(M-1)} \\ 1 & r^2 & r^4 & r^6 & \dots & r^{2(M-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & r^{(\frac{M}{2}-1)} & r^{2(\frac{M}{2}-1)} & r^{3(\frac{M}{2}-1)} & \dots & r^{(\frac{M}{2}-1)(M-1)} \end{bmatrix} \tag{20}$$

Substituting the identity $r^p = (\cos(\frac{2\pi}{M}) + i\sin(\frac{2\pi}{M}))^p = (\cos(\frac{2\pi p}{M}) + i\sin(\frac{2\pi p}{M}))$ into this previous expression allows us to easily separate the real and imaginary components; the spatial transform operator is created by concatenating (alternatively: conjoining, appending) the imaginary components of this matrix to the bottom of the real component and multiplying by a normalization factor

$$[\mathcal{F}_x] \equiv \frac{1}{\sqrt{M}} \begin{bmatrix} 1 & \cos(\frac{2\pi}{M}) & \cos(\frac{4\pi}{M}) & \dots & \cos(\frac{2(M-1)\pi}{M}) \\ 1 & \cos(\frac{4\pi}{M}) & \cos(\frac{8\pi}{M}) & \dots & \cos(\frac{4(M-1)\pi}{M}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \cos(\frac{2(\frac{M}{2}-1)\pi}{M}) & \cos(\frac{4(\frac{M}{2}-1)\pi}{M}) & \dots & \cos(\frac{2(\frac{M}{2}-1)(M-1)\pi}{M}) \\ 1 & \sin(\frac{2\pi}{M}) & \sin(\frac{4\pi}{M}) & \dots & \sin(\frac{2(M-1)\pi}{M}) \\ 1 & \sin(\frac{4\pi}{M}) & \sin(\frac{8\pi}{M}) & \dots & \sin(\frac{4(M-1)\pi}{M}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \sin(\frac{2(\frac{M}{2}-1)\pi}{M}) & \sin(\frac{4(\frac{M}{2}-1)\pi}{M}) & \dots & \sin(\frac{2(\frac{M}{2}-1)(M-1)\pi}{M}) \end{bmatrix} \tag{21}$$

The discrete Fourier transform in the real-valued basis is an orthogonal transformation and so the inverse of (21) can be derived by taking the matrix transpose of (21) and accounting for the normalization factor in definition of the inverse transform (7).

$$[\mathcal{F}_x^{-1}] = 2[\mathcal{F}_x]^\top \tag{22}$$

Recall that (21) was constructed such that it returns the spatial modes defined at a single instant in time $u(t', x_m)$. The generalization to space time is therefore simply derived by taking the Kronecker outer product of (21) with the identity matrix whose diagonal is the same dimension as the time discretization.

$$\mathbf{M}[\mathcal{F}_\mathbf{x}] = \mathbb{I}_N \otimes [\mathcal{F}_x] \tag{23}$$

and likewise for the matrix inverse.

The matrix representation for the temporal transforms follows from the previous exposition (21), the only difference being the inclusion of the zeroth mode. The temporal

transform matrix is then

$$[\mathcal{F}_t] \equiv \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \cos(\frac{2\pi}{N}) & \cos(\frac{4\pi}{N}) & \cdots & \cos(\frac{2(N-1)\pi}{N}) \\ 1 & \cos(\frac{4\pi}{N}) & \cos(\frac{8\pi}{N}) & \cdots & \cos(\frac{4(N-1)\pi}{N}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \cos(\frac{2(\frac{N}{2}-1)\pi}{N}) & \cos(\frac{4(\frac{N}{2}-1)\pi}{N}) & \cdots & \cos(\frac{2(\frac{N}{2}-1)(N-1)\pi}{N}) \\ 1 & \sin(\frac{2\pi}{N}) & \sin(\frac{4\pi}{N}) & \cdots & \sin(\frac{2(N-1)\pi}{N}) \\ 1 & \sin(\frac{4\pi}{N}) & \sin(\frac{8\pi}{N}) & \cdots & \sin(\frac{4(N-1)\pi}{N}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \sin(\frac{2(\frac{N}{2}-1)\pi}{N}) & \sin(\frac{4(\frac{N}{2}-1)\pi}{N}) & \cdots & \sin(\frac{2(\frac{N}{2}-1)(N-1)\pi}{N}) \end{bmatrix} \tag{24}$$

and the inverse

$$[\mathcal{F}_t^{-1}] = 2[\mathcal{F}_t]^\top . \tag{25}$$

The Kronecker product appears except in reversed order; also, the identity matrix reflects the spatial dimension

$$\mathbf{M}[\mathcal{F}_\mathbf{t}] = [\mathcal{F}_t] \otimes \mathbb{I}_x \tag{26}$$

This concludes the definition of the spatial and temporal Fourier transform matrices. In order to get the spatiotemporal version, simply take the product $M[\mathcal{F}_t]M[\mathcal{F}_x]$. However, these matrices are kept separate for reasons pertaining to the formulation of $f(\mathbf{v})$.

2.2 Spatiotemporal Symmetries

The symmetries of the Kuramoto-Sivashinsky equation in the spatiotemporal formulation include Galilean invariance and the symmetry group $G \equiv \mathrm{SO}(2) \times On2$. This spatiotemporal symmetry group provides a new perspective with which to view orbits and their symmetries. This in turn provides a new tool which greatly benefits the computations to follow. These benefits rely on an important distinction to be made; namely, a distinction must be made between *equivariance* and *invariance* of orbits with respect to group operations.

For a subgroup $H \subset G$, an orbit is H-equivariant, if every element of the group orbit produced by H

$$\mathcal{M} = h \cdot \mathbf{z} | h \in H \tag{27}$$

are themselves orbits, that is, they are all solutions to (**??**). Invariance is used to describe the isotropy subgroup (alternatively *stabilizer group* or *little group*) with respect to an orbit; that is if $K \subset H \subset G$ is the isotropy subgroup of $\mathbf{u}$ if

$$k \cdot \mathbf{u} = \mathbf{u}, \quad \forall k \in K .; s \tag{28}$$

Note that in this context, this is a very precise statement that the velocity field $u(t_n, x_m)$is *exactly* the same as it was prior to the symmetry operation.

For clarity, the statements: 'an orbit has a discrete symmetry' and 'an orbit with discrete symmetry' is always a statement of invariance, not equivariance. First the group orbits and isotropy groups considered here are defined. In order to

The group orbit for orbits without discrete symmetries are G-equivariant, their isotropy subgroups are the trivial subgroup. In the discrete setting, the group orbit $u(t_n, x_m)$ is not G. Because the field is defined only at the collocation points, arbitrary rotations are essentially interpolations of the field because they return the field at points in between the collocation points, which do not satisfy (**??**) numerically. Therefore, the group orbit of the discretized field $u(t_n, x_m)$is defined by spatial reflection and discrete spatiotemporal rotations, i.e.

$$H_{\mathbf{z}} = \mathrm{C}_N \times \mathrm{D}_M \tag{29}$$

C_N is the cyclic group of order N that accounts for discrete time rotations. D_M is the dihedral group generated by discrete spatial rotations and reflection about $\frac{L}{2}$. The isotropy group in this case is the trivial subgroup of G.

For brevity of the discussion that follows, let σ represent spatial reflection about $x = \frac{L}{2}$ such that

$$\sigma u(t_n, x_m) = -u(t_n, L - x_m) \tag{30}$$

and let τ represent half-cell $(T/2)$ translations in time

$$\tau u(t_n, x_m) = u(t_n + \frac{T}{2}, x_m)\,. \tag{31}$$

Finally, the spatiotemporal shift-reflection is defined by the composition of these two operations $s = \sigma\tau$ such that

$$\sigma\tau u(t_n, x_m) = -u(t_n + \frac{T}{2}, L - x_m)\,. \tag{32}$$

The effect of these transformations in the mode basis is determined by the trigonometric basis functions, written in tensor notation

$$\sigma\mathbf{u} = \begin{bmatrix} -a_{jk} & c_{jk} \\ -b_{jk} & d_{jk} \end{bmatrix} \quad \tau\mathbf{u} = \begin{bmatrix} (-1)^j a_{jk} & (-1)^j c_{jk} \\ (-1)^j b_{jk} & (-1)^j d_{jk} \end{bmatrix} \quad \sigma\tau\mathbf{u} = \begin{bmatrix} (-1)^{j+1} a_{jk} & (-1)^j c_{jk} \\ (-1)^{j+1} b_{jk} & (-1)^j d_{jk} \end{bmatrix} \tag{33}$$

Hereafter, orbits invariant under spatial reflection and spatiotemporal shift-reflection will be referred to as antisymmetric orbits and shift-reflection orbits, respectively.

The group orbit for both antisymmetric and shift-reflection symmetric orbits is generated by half-cell shifts and reflections in space, D_2, and discrete time rotations, C_N.

$$H_{\mathbf{z}} = \mathrm{C}_N \times \mathrm{D}_2,\,. \tag{34}$$

For antisymmetric orbits, the isotropy subgroup consists of only spatial reflections,

$$H_a = \{e, \sigma\} \tag{35}$$

and similarly for shift-reflection, represented as $\sigma\tau$,

$$H_{\sigma\tau} = \{e, \sigma\tau\}\,. \tag{36}$$

There are a number of important similarities between antisymmetric and shift-reflection orbits. The group orbits for both are generated by (34), the isotropy groups are both equivalent to Z_2 in their structure, and each symmetry constrains half of the orbits' modes to be equal to zero (different subsets of modes). A connection can be made by realizing that

the isotropy groups are both subgroups of the Klein four-group, K_4, comprised of the elements $\{e, \sigma, \tau, \sigma\tau\}$ which represent the identity, spatial reflection, half-cell time translation, and spatiotemporal shift-reflection. Shift-reflection is also sometimes referred to as *glide reflection*. Shift-reflection orbits are the spatiotemporal equivalent of pre-periodic orbits described in the dynamical systems formulation, as can be seen by making the connection to plane Couette flow; another system where shift-reflection presents itself. If constrained to two dimensions, the correspondence is seen immediately between the shift-reflection defined here and defined in.

$$\begin{aligned} s_1 \circ w(z,x) &= -w(L_z - z, x + \frac{L_x}{2}) \\ \sigma\tau \circ u(x,t) &= = -u(L - x, t + \frac{T}{2}) \end{aligned} \tag{37}$$

Clearly, $(w(z,x), z, x)$ of plane Couette share a correspondence with $(u(x,t), x, t)$ of the Kuramoto-Sivashinsky equation, respectively. The notion of 'fundamental domains' and pre-periodicity is replaced with a spatiotemporal symmetry invariant subspace of the modes, defined by constraints which require modes to vanish. These constraints shall hereafter be referred to as *selection rules*.

There are three different methods that these invariant subspaces and their selection rules can be derived. These three methods are: group theoretic derivation using projection operators and irreducible subspaces, multiplication of the matrix representations of projection operators with the mode vector, and lastly the application of the invariance condition (28). The advantages and disadvantages for each of these three methods are as follows. The 'group theoretic' derivation is useful when the symmetry group is simple; i.e. of low order, when the character table is small, or the number of irreducible representations is small (are all equivalent conditions). When this is the case, the decomposition into components and their substitution into any nonlinearities is manageable. When the order of the symmetry group is large it would likely not be obvious which subspace nonlinear combinations of components belong to, especially in the presence of non-commuting differential operators. The advantage of this calculation is a precise description of the invariant subspaces. The second method, construction of the matrix representations of the projection operators allows other linear operators to be projected into the invariant subspaces; something which will be done to the temporal Fourier transform shortly hereafter. The third method, via the invariance condition (28) allows for quick computation of the selection rules for usage in computations.

The symmetry groups in are also representations of K_4, the difference is that now the group elements involve operations in both space and time. The proceeding analysis is nearly identical in its derivation but the interpretation is necessarily different due to the lack of dynamics.

Starting with the group theoretic calculations; starting with the character table table 1 is (38). The character table, table 1, leads to the construction of four linear projection operators

$$\begin{aligned} P^{(++)} &= \frac{1}{4}(1 + \sigma + \tau + \sigma\tau) \\ P^{(+-)} &= \frac{1}{4}(1 + \sigma - \tau - \sigma\tau) \\ P^{(-+)} &= \frac{1}{4}(1 - \sigma + \tau - \sigma\tau) \\ P^{(--)} &= \frac{1}{4}(1 - \sigma - \tau + \sigma\tau)\,, \end{aligned} \tag{38}$$

Table 1: Because the direct product group is abelian we only have one dimensional representations and as such the character table follows directly.

	e	σ	τ	$\sigma\tau$
E	1	1	1	1
Γ_1	1	1	-1	-1
Γ_2	1	-1	1	-1
Γ_3	1	-1	-1	1

The solution space can be decomposed into the irreducible subspaces $\mathbb{U} = \mathbb{U}^{(++)} \oplus \mathbb{U}^{(+-)} \oplus \mathbb{U}^{(-+)} \oplus \mathbb{U}^{(--)}$. Applying the projection operators (38) to the modes in tensor notation and using the relations (**??**) to evaluate (38) yields the four irreducible subspaces of modes

$$P^{(--)}\mathbf{u} = \mathbf{u}^{(--)} = \begin{bmatrix} 0 & 0 \\ a_{1,k} & 0 \\ 0 & 0 \\ \vdots & \vdots \\ b_{1,k} & 0 \\ 0 & 0 \\ \vdots & \vdots \end{bmatrix} \qquad P^{(-+)}\mathbf{u} = \mathbf{u}^{(-+)} = \begin{bmatrix} a_{0,k} & 0 \\ 0 & 0 \\ a_{2,k} & 0 \\ \vdots & \vdots \\ 0 & 0 \\ b_{2,k} & 0 \\ \vdots & \vdots \end{bmatrix}$$

$$P^{(+-)}\mathbf{u} = \mathbf{u}^{(+-)} = \begin{bmatrix} 0 & 0 \\ 0 & c_{1,k} \\ 0 & 0 \\ \vdots & \vdots \\ 0 & d_{1,k} \\ 0 & 0 \\ \vdots & \vdots \end{bmatrix} \qquad P^{(++)}\mathbf{u} = \mathbf{u}^{(++)} = \begin{bmatrix} 0 & c_{0,k} \\ 0 & 0 \\ 0 & c_{2,k} \\ \vdots & \vdots \\ 0 & 0 \\ 0 & d_{2,k} \\ \vdots & \vdots \end{bmatrix} \tag{39}$$

These four subspaces are apparently based on being symmetric or antisymmetric with respect to reflections about $L/2$ and half-cell time translations of $T/2$. Before applying these subspaces the effect of exchanging the projection operators (38) and spatial differentiation operator is derived; as this is directly relevant for the calculation of the nonlinear term in

(??)

$$
\begin{aligned}
D_x P^{(++)} &= \frac{1}{4} D_x (1 + \sigma + \tau + \sigma\tau) \\
&= \frac{1}{4} (1 - \sigma + \tau - \sigma\tau) D_x \\
&= P^{(-+)} D_x \\
D_x P^{(+-)} &= \frac{1}{4} D_x (1 + \sigma - \tau - \sigma\tau) \\
&= \frac{1}{4} (1 - \sigma - \tau + \sigma\tau) D_x \\
&= P^{(--)} D_x \\
D_x P^{(-+)} &= \frac{1}{4} D_x (1 - \sigma + \tau - \sigma\tau) \\
&= \frac{1}{4} (1 + \sigma + \tau + \sigma\tau) D_x \\
&= P^{(++)} D_x \\
D_x P^{(--)} &= \frac{1}{4} D_x (1 - \sigma - \tau + \sigma\tau) \\
&= \frac{1}{4} (1 + \sigma - \tau - \sigma\tau) D_x \\
&= P^{(+-)} D_x \,. \qquad (40)
\end{aligned}
$$

The effect of exchanging the operators can be summarized by flipping the sign of the first $\pm$, pertaining to the coefficient of the spatial reflection terms in (38). The components of the nonlinear term in each subspace are found to be

$$
\begin{aligned}
P^{(++)}(u\partial_x u) &= u^{(\pm\pm)} \partial_x (u^{(\pm\pm)}) \\
P^{(+-)}(u\partial_x u) &= u^{(\pm\pm)} \partial_x (u^{(\pm\mp)}) \\
P^{(-+)}(u\partial_x u) &= u^{(\pm\pm)} \partial_x (u^{(\mp\pm)}) \\
P^{(--)}(u\partial_x u) &= u^{(\pm\pm)} \partial_x (u^{(\mp\mp)}) \,. \qquad (41)
\end{aligned}
$$

Proper usage of $\pm$ and $\mp$ in (41) generates only four unique terms per projection; i.e. the values of plus-minuses outside the differential are paired to those within. Using these relations (41) we can produce the projections of the Kuramoto-Sivashinsky equation onto the different irreducible subspaces, noting that the projection operator commutes with the

linear terms such that

$$\begin{aligned}
P^{(++)}f(\mathbf{v}) &= u_t^{(++)} + u_{xx}^{(++)} + u_{xxxx}^{(++)} \\
&\quad + \big[u^{(++)}\partial_x(u^{(++)}) + u^{(+-)}\partial_x(u^{(+-)}) \\
&\quad + u^{(-+)}\partial_x(u^{(-+)}) + u^{(--)}\partial_x(u^{(--)})\big] \\
P^{(+-)}f(\mathbf{v}) &= u_t^{(+-)} + u_{xx}^{(+-)} + u_{xxxx}^{(+-)} \\
&\quad + \big[u^{(++)}\partial_x(u^{(+-)}) + u^{(+-)}\partial_x(u^{(++)}) \\
&\quad + u^{(-+)}\partial_x(u^{(--)}) + u^{(--)}\partial_x(u^{(-+)})\big] \\
P^{(-+)}f(\mathbf{v}) &= u_t^{(-+)} + u_{xx}^{(-+)} + u_{xxxx}^{(-+)} \\
&\quad + \big[u^{(++)}\partial_x(u^{(-+)}) + u^{(+-)}\partial_x(u^{(--)}) \\
&\quad + u^{(-+)}\partial_x(u^{(++)}) + u^{(--)}\partial_x(u^{(+-)})\big] \\
P^{(--)}f(\mathbf{v}) &= u_t^{(--)} + u_{xx}^{(--)} + u_{xxxx}^{(--)} \\
&\quad + \big[u^{(++)}\partial_x(u^{(--)}) + u^{(+-)}\partial_x(u^{(-+)}) \\
&\quad + u^{(-+)}\partial_x(u^{(+-)}) + u^{(--)}\partial_x(u^{(++)})\big]\,. \qquad (42)
\end{aligned}$$

With this we can determine the *symmetry invariant subspaces* or simply *invariant subspaces* by solving $Pf(\mathbf{v}) = \mathbf{F}(P\mathbf{v})$ where $P = \sum_k P^{(k)}$ noting that the projection operator acts only on the $\mathbf{u}$ component of $\mathbf{v}$. This is similar to the notion of *flow invariant subspaces* except that there are no longer any dynamics, and so the use of 'flow' does not apply here. One method of deriving the invariant subspaces is to simply assume a decomposition of u, then show that the sum of the relevant equations (42) commute with the projection operator. For example, assume $u = u^{(++)}$; substitution into (42) yields

$$\begin{aligned}
P^{(++)}F(u^{(++)}) &= u_t^{(++)} + u_{xx}^{(++)} + u_{xxxx}^{(++)} + u^{(++)}\partial_x(u^{(++)}) \\
F(P^{(++)}u) &= u_t^{(++)} + u_{xx}^{(++)} + u_{xxxx}^{(++)} + \frac{1}{2}\partial_x((u^{(++)})^2)\,. \qquad (43)
\end{aligned}$$

which are equivalent after applying the derivative in the second equation. Therefore $\mathbb{U}^{(++)}$ constitutes an invariant subspace. This subspace corresponds to antisymmetric equilibria. The rest of the symmetry invariant subspaces follow from a similar substitutions. To expedite the derivation process, note that the equation for $P^{(++)}F$ contains all of the symmetric terms $u^{\pm\pm}\partial_x(u^{\pm\pm})$ such that invariant subspaces must have non-empty intersection with $\mathbb{U}^{(++)}$. Following a process of elimination we can show that the possible (nontrivial) symmetry invariant subspaces are $\mathbb{U}^{(++)}$, $\mathbb{U}^{(++)} \oplus \mathbb{U}^{(--)}$, $\mathbb{U}^{(++)} \oplus \mathbb{U}^{(+-)}$ and $\mathbb{U}^{(++)} \oplus \mathbb{U}^{(-+)}$. We can interpret these subspaces by addition of the corresponding projection operators (38)

$$\begin{aligned}
P^0 \equiv P^{(++)} &= \frac{1}{4}(1 + \sigma + \tau + \sigma\tau) \\
P^\sigma \equiv P^{(++)} + P^{(+-)} &= \frac{1}{2}(1 + \sigma) \\
P^\tau \equiv P^{(++)} + P^{(-+)} &= \frac{1}{2}(1 + \tau) \\
P^{\sigma\tau} \equiv P^{(++)} + P^{(--)} &= \frac{1}{2}(1 + \sigma\tau)\,. \qquad (44)
\end{aligned}$$

The invariant subspaces following trivially (with a slight abuse of the irreducible subspace

notation)

$$\begin{aligned} \mathbb{U}_0 &\equiv \{\mathbf{v} \mid P_0\mathbf{v_u} = \mathbf{v_u}\} \\ \mathbb{U}_\sigma &\equiv \{\mathbf{v} \mid P_\sigma\mathbf{v_u} = \mathbf{v_u}\} \\ \mathbb{U}_\tau &\equiv \{\mathbf{v} \mid P_\tau\mathbf{v_u} = \mathbf{v_u}\} \\ \mathbb{U}_{\sigma\tau} &\equiv \{\mathbf{v} \mid P_{\sigma\tau}\mathbf{v_u} = \mathbf{v_u}\} \end{aligned} \tag{45}$$

where $\mathbf{v_u}$ are the modes of (2). $\mathbb{U}^0$ represents the subspace of antisymmetric equilibria, $\mathbb{U}^\sigma$ the spatial reflection invariant subspace, $\mathbb{U}^{\sigma\tau}$ the shift-reflection invariant subspace, and lastly $\mathbb{U}^\tau$ with solutions 'twice-repeating' solutions, i.e. those symmetric with respect to $T/2$. The continuous spatial translations are eliminated from every subspace *except* $\mathbb{U}^{(++)} \oplus \mathbb{U}^{(-+)}$. This can be made explicit by acting on the projected modes (**??**) with the generator of translations (to be defined in sect. **??**. Because of its unique properties, the $\mathbb{U}^{(++)} \oplus \mathbb{U}^{(-+)}$ subspace will be discussed at the end of the section.

Naturally the definition of $f(\mathbf{v})$ in the shift-reflection and reflection invariant subspaces comes from the substitution of $u^{(++)} + u^{(--)}$ and $u^{(++)} + u^{(+-)}$ into $f(\mathbf{v})$ however, this is not a very useful representation of the equations; instead the equations are derived in terms of the components of the mode tensor (13) in sect. **??**.

The invariance condition (28) is used to derive constraints on which modes are non-vanishing, referred to as *selection rules*. These selection rules correspond directly to the invariant subspaces produced by (44). Therefore, using the invariance condition (28) is equivalent to applying the corresponding projection operator (**??**) and finding which modes survive, $P\mathbf{u} = \mathbf{u}$. The invariance condition will be used to derive the selection rules in a functional form, but the matrix representations of the projection operators (44) are useful for applying the selection rules to other linear operators. The focus is only on reflection and shift-reflection invariance, such that the derivations are limited to the matrix representations of P_σ and $P_{\sigma\tau}$. All derivations will utilize the conventions on the notation for identity matrices from (17). For the real and imaginary components of a single spatial mode, the spatial reflection operator (with reflection axis $x = L/2$) is

$$\sigma'' = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \tag{46}$$

This changes the sign of the real component of the mode. Generalizing to a set of $\frac{M}{2} - 1$ spatial modes (using the arrangement convention described in sect. **??** this becomes

$$\sigma' = \sigma'' \otimes \mathbb{I}_k = \begin{bmatrix} -\mathbb{I}_k & 0 \\ 0 & \mathbb{I}_k \end{bmatrix} \tag{47}$$

Where 0 represents the appropriately sized *matrix* of zeros whose dimension in this case is $(\frac{M}{2} - 1)^2$. The spatiotemporal version of (47) is a diagonal matrix consisting of $N - 1$

repeats of (47)

$$\begin{aligned}\sigma &= \mathbb{I}_{(N-1)} \otimes \sigma^{'} \\ &= \begin{bmatrix} -\mathbb{I}_k & & & & \\ & \mathbb{I}_k & & & \\ & & -\mathbb{I}_k & & \\ & & & \mathbb{I}_k & \\ & & & & \ddots \end{bmatrix}. \end{aligned} \tag{48}$$

with (48) the projection operator P_σ can be defined

$$\begin{aligned}P_\sigma &= \frac{1}{2}(\mathbb{I} + \sigma) \\ &= \begin{bmatrix} 0 & & & & \\ & \mathbb{I}_k & & & \\ & & 0 & & \\ & & & \mathbb{I}_k & \\ & & & & \ddots \end{bmatrix}. \end{aligned} \tag{49}$$

Which can be seen clearly keeps the spatially antisymmetric components of the modes with respect to the ordering (18).

The shift-reflection projection operator requires both the temporal half-cell shift τ and the reflection operator (48). Half-cell shift is equivalent to rotation by π such that

$$\tau^{'} = \begin{bmatrix} \cos(j\pi) & \sin(j\pi) \\ -\sin(j\pi) & \cos(j\pi) \end{bmatrix} = \begin{bmatrix} 1 & & \\ & (-1)^j\,\mathbb{I}_j & \\ & & (-1)^j\,\mathbb{I}_j \end{bmatrix} \tag{50}$$

This specific form of (50) is used to represent the fact that j follows the tensor index convention defined in (15). The spatiotemporal version of (**??**) is achieved via another Kronecker product.

$$\tau = \tau^{'} \otimes \mathbb{I}_{(M-2)} = \begin{bmatrix} 1 & & & \\ & (-1)^j\,\mathbb{I}_j & & \\ & & (-1)^j\,\mathbb{I}_j & \\ & & & \ddots \end{bmatrix} \tag{51}$$

Using the Kronecker product identity $(A \otimes B)(C \otimes D) = (AC \otimes BD)$ (if the sizes are

consistent) we have

$$\begin{aligned} P_{\sigma\tau} &= \frac{1}{2}(\mathbb{I} + \sigma\tau) \\ &= \frac{1}{2}(\mathbb{I} + (\mathbb{I}_{(N-1)} \otimes \sigma^{'})(\tau^{'} \otimes \mathbb{I}_{(M-2)})) \\ &= \frac{1}{2}(\mathbb{I} + (\tau^{'} \otimes \sigma^{'}))) \end{aligned} \tag{52}$$

where

$$(\tau^{'} \otimes \sigma^{'}) \equiv \begin{bmatrix} -\mathbb{I}_k & & & & & & & & \\ & \mathbb{I}_k & & & & & & & \\ & & (-\mathbb{I}_k)^{j+1} & & & & & & \\ & & & (-\mathbb{I}_k)^{j} & & & & & \\ & & & & \ddots & & & & \\ & & & & & (-\mathbb{I}_k)^{j+1} & & & \\ & & & & & & (-\mathbb{I}_k)^{j} & & \\ & & & & & & & \ddots & \end{bmatrix} \tag{53}$$

Substitution into the expression for $P_{\sigma\tau}$ yields a diagonal matrix where the terms with $j+1$ survive with j is odd, and the terms with j survive when j is even. This makes it so the diagonal alternates between two identity matrix blocks and two zero blocks until, that is, j 'restarts'.

$$P_{\sigma\tau} \equiv \begin{bmatrix} 0 & & & & & & & \\ & \mathbb{I}_k & & & & & & \\ & & \mathbb{I}_k & & & & & \\ & & & 0 & & & & \\ & & & & 0 & & & \\ & & & & & \ddots & & \\ & & & & & & \mathbb{I}_k & \\ & & & & & & & \ddots \end{bmatrix}. \tag{54}$$

This concludes the derivation of the two projection operators found to be useful in this study. These projections can be formulated in a more efficient manner where the matrices are never explicitly constructed.

The explicit construction of the matrices (49) and (54) is inefficient and unnecessary as the projections can be written in a functional form as a set of constraints or *selection rules* as they are denoted here.

The selection rules pertaining to a particular symmetry are derived by substituting (11) into (28). The modes that can only satisfy the equation if they are identically zero will be referred to as *vanishing modes*.

For spatial reflection symmetry the selection rules come immediately without any calculations; only modes attached to $\sin(q_k x_m)$ are allowed. In other words the vanishing modes are coefficients to $\cos(q_k x_m)$ in (11)

$$a_{jk} = b_{jk} \ = 0 \quad \forall (j,k)\,. \tag{55}$$

The selection rules for equilibria follow trivially from this, simply get rid of any dependence on time index j.

$$\begin{aligned} a_{jk} = b_{jk} = d_{jk} = \ & 0 \quad \forall (j,k) \\ c_{jk} = \ & 0 \quad \forall j \neq 0 \\ (j,k) \quad & \widehat{=} \quad \{\{0\}\} \times \{\{k\}\} \\ |(j,k)| \quad & = \quad \frac{M}{2} - 1 \quad \text{non-vanishing modes.}\,. \end{aligned} \tag{56}$$

Unfortunately the shift-reflection selection rules are not as simply derived. The exact long form expression of the Fourier transform (11) is replaced with a simplified version to make the derivation concise and clear. Utilizing trigonometric identities and the parity of sin and cos we have

$$\begin{aligned} \sigma\tau\, u(t_n, x_m) = & -\sum_{k,j} \ \Big(a_{jk}\cos(\tilde{\omega}_j(t_n + T/2)) + b_{jk}\sin(\tilde{\omega}_j(t_n + T/2))\Big)\cos(q_k(L - x_m)) \\ & \qquad \Big(-c_{jk}\cos(\tilde{\omega}_j(t_n + T/2)) - d_{jk}\sin(\tilde{\omega}_j(t_n + T/2))\Big)\sin(q_k(L - x_m)) \\ = & \sum_{k,j} -\cos(\pi j)\Big((a_{jk}\cos(\tilde{\omega}_j t_n) + b_{jk}\sin(\tilde{\omega}_j t_n))\Big)\cos(q_k x_m) \\ & \quad +\cos(\pi j)\Big(-c_{jk}\cos(\tilde{\omega}_j t_n) - d_{jk}\sin(\tilde{\omega}_j t_n)\Big)\sin(q_k x_m) \\ = & \sum_{k,j} \ (-1)^{j+1}\Big((a_{jk}\cos(\tilde{\omega}_j t_n) + b_{jk}\sin(\tilde{\omega}_j t_n))\Big)\cos(q_k x_m) \\ & \quad +(-1)^j\Big(-c_{jk}\cos(\tilde{\omega}_j t_n) - d_{jk}\sin(\tilde{\omega}_j t_n)\Big)\sin(q_k x_m)\,, \end{aligned} \tag{57}$$

substitution into the invariance condition (28) yields the following selection rules for spatiotemporal shift-reflection

$$\begin{aligned} a_{jk} = b_{jk} = 0 \ \text{ for } \ j \ \text{ even} \\ c_{jk} = d_{jk} = 0 \ \text{ for } \ j \ \text{ odd } \ . \end{aligned} \tag{58}$$

To benefit from these selection rules numerically, simply constraining the modes to be zero is not sufficient. The main benefit is derived from the discarding of the vanishing modes which reduces the number of computational degree of freedom. The non-vanishing modes must be represented in a contiguous fashion and so a rearrangement of the array is in order. Much like the derivation of the selection rules themselves, this is straightforward for antisymmetric orbits; this can also be described via the index notation of (15)

$$\begin{aligned} \mathbf{u}_\sigma \quad & \equiv \quad \begin{bmatrix} c_{jk} \\ d_{jk} \end{bmatrix}\,. \\ (j,k) \quad & \widehat{=} \quad \{\{0\},\{j\},\{j\}\} \times \{k\} \\ |(j,k)| \quad & = \quad (N-1) \times (\frac{M}{2} - 1) \quad \text{non-vanishing modes}\,. \end{aligned} \tag{59}$$

Shift-reflection seems harder to handle at first but luckily there is a trick that can be exploited to yield a coherent form. This is most easily seen by explicitly applying the selection rules to (13)

$$\mathbf{u}_{\sigma\tau} \equiv \begin{bmatrix} 0 & c_{0,k} \\ a_{1,k} & 0 \\ 0 & c_{2,k} \\ \vdots & \vdots \\ b_{1,k} & 0 \\ 0 & d_{2,k} \\ \vdots & \vdots \end{bmatrix} . \tag{60}$$

This staggered structure can be exploited by discarding the zeros and 'shuffling' the modes together

$$\begin{aligned} \mathbf{u}_{\sigma\tau} &\equiv \begin{bmatrix} c_{0,k} \\ a_{1,k} \\ c_{2,k} \\ \vdots \\ b_{1,k} \\ d_{2,k} \\ \vdots \end{bmatrix} \\ &\widehat{=} \quad \{\{0\},\{j\},\{j\}\} \times \{k\} \\ |(j,k)| &= (N-1) \times (\frac{M}{2} - 1) \quad \text{non-vanishing modes}\,. \end{aligned} \tag{61}$$

By formatting the modes in this way note that the temporal frequencies are arranged in a manner identical to (13) and (59). In other words, no extra work is required to define time differentiation. Also note that the shift-reflection subspace has the same dimension as the antisymmetric subspace which makes sense in the scope of the symmetry group. The down side is that the modes need to be 'unshuffled' before the inverse transform `irfft` can be applied, that is, the zeros present in (60) need to be reinserted.

The selection rules are incorporated into the definition of the temporal transforms to yield a 'symmetry invariant' Fourier transform. By doing so, there is no possibility of forgetting to apply the selection rules. It is also necessary to apply these selection rules to the matrix representations of linear operators. Discarding the vanishing modes requires a modification of the projection operators (49), (54). Note that the projection operators are diagonal matrices and that the vanishing modes correspond to where the diagonal equals zero. Therefore, the vanishing modes are properly discarded by removing these rows of the projection matrices. Applying this to other linear operators is simply derived by matrix multiplication and so the explicit results are not derived.

Before proceeding to continuous symmetries, here is a summary on what has been covered for discrete symmetries. First, the distinction between equivariance and invariance was made. By doing so, very specific definitions of group orbits and isotropy groups were defined. It was realized that the isotropy subgroups of interest were both subgroups of a larger group which is equivalent to the Klein four group. Owing to the structure of this group, irreducible subspaces and their linear projection operators were defined. Four symmetry invariant subspaces were then derived by acting on the Kuramoto-Sivashinsky

equation with combinations of these projection operators. Each of these subspaces has a very useful set of constraints denoted as selection rules (equivalent to projection onto the respective subspace). The selection rules were defined in both a functional form and as matrices. Finally, the rules were incorporated into the temporal transforms, referring to these new transforms as symmetry invariant Fourier transforms. For orbits within each invariant subspace the redefined Fourier transforms remain orthogonal transformations.

There are two continuous symmetries that are accounted for, Galilean invariance and spatial translation symmetry which results in relative periodic orbits. The first of these two symmetries, invariance under Galilean transformations, results in the following equivariance: if $u(t,x)$is a solution, then $u(t,x-ct)-c$ is also a solution (c being an arbitrary constant speed or 'mean flow'). This is handled with a simple constraint which constrains the mean flow with respect to space

$$\langle u\rangle(t) \;=\; \int_0^L dx\, u(x,t) = 0\,. \tag{62}$$

The practical method to enforce this is to simply discard the zeroth spatial modes, i.e. those with $k=0$. This is a conventional choice but not necessarily the best choice; there is to be a longer discussion on this later, but the general idea is that reducing the group orbit without purpose is not desirable. Finding orbits is the goal, not finding specific members of group orbits. It is believed that reducing the dimension of the group orbit may reduce the likelihood of finding that group orbit numerically. Regardless, the convention defined by (**??**) is enforced.

The last spatiotemporal symmetry to discuss is spatial translation symmetry which creates relative periodic orbits. To make a distinction between uniform rotations, this symmetry will be referred to as *spatial shift* symmetry. A relative periodic orbit is defined here as an orbit whose field $u(t_n,x_m)$is only periodic after accounting for a 'spatial drift' or 'shift'. In other words, relative periodic orbits abide by the following relation

$$u(t,x) = g\circ u(t+T,x)\,. \tag{63}$$

Where g represents a spatial shift defined by the $f(\mathbf{v})$. If this shift is not accounted for, the $u(t,x)$ will be discontinuous in time. To make the most out of the Fourier basis some intervention is required to generate a truly periodic representation of the field.

Two ideas come to mind, either utilize a comoving reference frame or slice (quotient) the continuous symmetry. The slicing method has a number of disadvantages and so only a comoving frame is used. These disadvantages and their derivation are relegated to the appendix. The comoving frame and its spatial shift needs to be formulated in dimensionless spatial units as opposed to an angular quantity. This is important because T and L and s are changing in the numerical optimization process and they are coupled in a non-trivial manner. First, spatial rotations are defined in a manner practically identical to (**??**) The matrix representation of SO(2) group elements for a spatial shift by an amount s, for a single time value $u(t', x_m - s)$ is

$$g(s) \equiv \begin{bmatrix} \cos(q_k s) & \sin(q_k s) \\ -\sin(q_k s) & \cos(q_k s) \end{bmatrix}\,. \tag{64}$$

The index k indicate that each element (64) represents a diagonal matrix where k takes values (14). This matrix applied to a set of spatial modes defined at a specific instant in time, shifts them by s in the positive x direction. The spatiotemporal version of this operator, that is, uniform spatial rotations of the entirety of $u(t_n,x_m)$, is a block diagonal matrix

with N blocks; one for each value of t_n. Both uniform spatial translations and comoving rotations are applied in the spatial mode basis, as it makes more sense considering the parameterization in time to follow. The uniform shift (64) generalizes to space-time via Kronecker product as seen before in (51)

$$\mathbf{g}(s) = \mathbb{I}_N \otimes g(s) \tag{65}$$

which, explicitly, equals

$$\mathbf{g}(s) \equiv \begin{bmatrix} g(s) & 0 & \cdots & 0 \\ 0 & g(s) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & g(s) \end{bmatrix} . \tag{66}$$

The general idea originates in López *et al.* [40] where the field is kept in a co-moving reference frame. The co-moving reference frame transformation is a generalization of (66) where the previously uniform shift s is replaced by a spatial shift linearly parameterized by time. That is, at every discrete time t_n, the field is shifted by an amount $\frac{\mathrm{S}t_n}{T}$ via the symmetry operation $g(\frac{\mathrm{S}t_n}{T})$. For brevity let $s_n \equiv s(t_n) \equiv \frac{\mathrm{S}t_n}{T}$. Using (**??**) the matrix representation of the co-moving frame transformation is as follows

$$\mathbf{g}(\phi_n) \equiv \begin{bmatrix} g(\phi_{N-1}) & 0 & \cdots & 0 \\ 0 & g(\phi_{N-2}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & g(\phi_0) \end{bmatrix} . \tag{67}$$

Note that the time is decreasing along the diagonal due to the conventions of the numerical implementation. Using (67) The ansatz for relative periodic solutions can be written as the modification of the spatiotemporal transform (11). As a convention, the shift s is defined as the shift from the comoving frame to the physical frame such that. The most succinct form of the comoving transformation is writing the expression in terms of operators for the translations and Fourier transforms.

$$u(t_n, x_m) = \mathcal{F}_x^{-1}(\mathbf{g}(\frac{\mathrm{S}t_n}{T})\mathcal{F}_t^{-1}(\vec{\mathbf{u}})) . \tag{68}$$

Where the action of $\mathbf{g}(\frac{\mathrm{S}t_n}{T})$ maps $x_m \to x_m - \frac{\mathrm{S}t_n}{T}$, a translation in the positive spatial direction. Equivalently, this can be written as the inverse Fourier transform of rotated spatial modes

$$\begin{aligned} \mathbf{g}\, u(t_n, x_m) = \frac{2}{\sqrt{M}} \sum_{k=1}^{\frac{M}{2}-1} & \left[e_k(t_n)\cos(q_k\frac{\mathrm{S}t_n}{T}) + f_k(t_n)\sin(q_k\frac{\mathrm{S}t_n}{T})\right]\cos(q_k x_m) \\ & + \left[-e_k(t_n)\sin(q_k\frac{\mathrm{S}t_n}{T}) + f_k(t_n)\cos(q_k\frac{\mathrm{S}t_n}{T})\right]\sin(q_k x_m) \\ = \frac{2}{\sqrt{M}} \sum_{k=1}^{\frac{M}{2}-1} & e_k(t_n)\cos(q_k(x_m - \frac{\mathrm{S}t_n}{T})) + f_k(t_n)\sin(q_k(x_m - \frac{\mathrm{S}t_n}{T})) . \end{aligned} \tag{69}$$

An important detail is that (69) reproduces the $u(t_n, x_m)$ which solves (**??**), *however*, numerically the field is kept always kept in the comoving frame; such that symmetry operation

is not built into the temporal Fourier transform as it is for other symmetries. The reasoning for this choice presents itself when deriving the $f(\mathbf{v})$ in terms of the modes.

In conclusion, a spatiotemporal formulation requires a spatiotemporal description of symmetries. The same concepts from apply here and so the derivations look quite similar. The interpretation however is not similar due to the absence of dynamics. Previously, the symmetries flow-invariant subspaces which are unstable to perturbations. Now, the spatiotemporal symmetries manifest as selection rules; constraints which require subsets of modes to vanish, i.e. equal zero. One application of this is the shift-reflection invariant subspace, which cannot be derived as a flow invariant subspace, as the symmetry involves time. It *is* possible to derive a shooting-method type constraint $u(t) = \sigma u(t + \frac{T}{2})$

but this type of equation is clearly highly dependent on time evolution. Therefore, the spatiotemporal formulation Before moving on, one last comment is to be made about the subspace corresponding to P_τ of (44) which was not mentioned so far. This subspace contains the 'twice-repeating' solutions of the Kuramoto-Sivashinsky equation. Now, this might seem trivial at first, because every orbit is doubly-periodic by definition. That is, any orbit has a representation which exists in this invariant subspace; created by simply taking a tiling of size $[0, 2T] \times [0, L]$. Note that one must use this representation of the orbit otherwise it would not be *invariant* under the transformation. This subspace remains unutilized currently; one hypothesis is that spatiotemporal symmetry groups of higher order (i.e. higher order cyclic subgroups) would permit more interesting behavior than simply 'twice-repeating'. One usage of the 'twice-repeating' subspace is that it can be used in conjunction with other symmetries to impose even stricter selection rules than (55) and (58). Why? For example, two repeats of a shift reflection orbit is invariant under more combinations of symmetry actions: for example, compositions of quarter-cell and three-quarter cell shifts with reflection leave the orbit invariant. Therefore, in order to be invariant under more symmetries, more modes are constrained to zero. A preliminary investigation shows that for twice-repeating shift-reflection orbits, the number of non-vanishing modes reduces by another factor of two. This can be interpreted via spatiotemporal symbolic dynamics, however the discussion is not yet equipped to handle this, however, the followed hypothesis is offered up as a potential explanation. Part of the hypothesis is that the frequency with which invariant 2-tori are shadowed is somehow inversely proportional to their size. Therefore, a region of space-time shadowing two-repeats of an orbit is a stricter and more specific condition than shadowing a single period. This manifests as a stricter constraint on the modes which in turn yields more selection rules.

In summary, the symmetries of the Kuramoto-Sivashinsky equation have been cast into a spatiotemporal form. From the discrete and continuous symmetries considered the following categories are permissable: antisymmetric, shift-reflection, relative periodic, equilibrium and relative equilibrium. Discrete symmetries can be utilized to produce selection rules, constraints on the modes, such that the computational degree of freedom can be reduced substantially. On the other hand, orbits with continuous symmetry actually require an additional computational degree of freedom, namely, the spatial shift accumulated after one temporal period.

2.3 Spatiotemporal Mapping

On these comments, the main references I have been studying are refs. [7, 11, 31, 40].

Now that the spatiotemporal Fourier transforms and spatiotemporal symmetries have been properly derived, the discussion can finally move on to the Kuramoto-Sivashinsky

equation itself. The Kuramoto-Sivashinsky equation (**??**) can now be transformed to yield the system of differential algebraic equations which are the foundation for everything that follows. The equations will be derived by substitution of (11) into (**??**); after a closed form expression is achieved the *efficient* manner of computing the nonlinear term will be derived.

To most efficiently explain how to compute the nonlinear term and derivatives 'element-wise multiplication' must be defined. The result of element-wise multiplication will be referred to as the *element-wise product*, sometimes referred to as either the Schur product or Hadamard product. The latter two names are often used in the context of matrices, but the generalization of this operation to tensors is straightforward. Given a pair of rank two tensors, the product will be represented by the operator $\cdot$ and is defined as

$$\begin{bmatrix} m_{11} & \dots & m_{1k} \\ \vdots & \ddots & \vdots \\ m_{j1} & \dots & m_{jk} \end{bmatrix} \cdot \begin{bmatrix} p_{11} & \dots & p_{1k} \\ \vdots & \ddots & \vdots \\ p_{j1} & \dots & p_{jk} \end{bmatrix} = \begin{bmatrix} m_{11}p_{11} & \dots & m_{1k}p_{1k} \\ \vdots & \ddots & \vdots \\ m_{j1}p_{j1} & \dots & m_{jk}p_{jk} \end{bmatrix} . \tag{70}$$

To make the distinction, matrix multiplication will be represented simply by adjacency, i.e. **AB**.

The product (70) is only well defined for tensors with the same dimensions; in other words, element-wise multiplication is only defined for orbits whose tiles' discretizations have the same number of grid points. This is not the same as requiring the tile dimensions (T, L) to be identical; although for practical purposes this is nearly always the case. With (70), Differentiation in the spatiotemporal mode basis can be described easily. This method, sometimes referred to as *spectral differentiation*, is performed efficiently by multiplying the modes (13) by the appropriate spatial or temporal frequencies noting that the SO(2) representation requires tracking of an additional factor of -1. It follows that derivatives of higher order require powers of the frequencies. Explicitly, these details can be represented with the differentiation matrices, i.e. the generators of SO(2) translations for space

$$\mathbf{M}[\partial_x] \equiv \mathbb{I}_{N-1} \otimes \frac{2\pi}{L} \begin{bmatrix} 0 & -k \\ k & 0 \end{bmatrix} \equiv \mathbb{I}_{N-1} \otimes \begin{bmatrix} 0 & q_k \\ -q_k & 0 \end{bmatrix} . \tag{71}$$

and for time

$$\mathbf{M}[\partial_t] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -\omega_j \\ 0 & \omega_j & 0 \end{bmatrix} \otimes \mathbb{I}_x . \tag{72}$$

There is a sign change in (73) resulting from the time ordering of $u(t, x)$ as discussed in the definition of the spatiotemporal Fourier transform. The time differentiation is close to the same form as (**??**) except for the inclusion of the zeroth time mode. At this point the reader is reminded to recall the conventions of the signs of the frequencies (6). The derivatives of higher order which are relevant are defined

$$\mathbf{M}[\partial_x] = \mathbb{I}_{N-1} \otimes \begin{bmatrix} 0 & q_k \\ -q_k & 0 \end{bmatrix} \qquad \mathbf{M}[\partial_{xx}] = \mathbb{I}_{N-1} \otimes \begin{bmatrix} -q_k^2 & 0 \\ 0 & -q_k^2 \end{bmatrix}$$

$$\mathbf{M}[\partial_{xxx}] = \mathbb{I}_{N-1} \otimes \begin{bmatrix} 0 & -q_k^3 \\ q_k^3 & 0 \end{bmatrix} \qquad \mathbf{M}[\partial_{xxxx}] = \mathbb{I}_{N-1} \otimes \begin{bmatrix} q_k^4 & 0 \\ 0 & q_k^4 \end{bmatrix}$$

Spectral differentiation can be calculated in two ways: the first method is to explicitly construct the matrices (**??**) and (**??**) and then multiply by the vector of modes $\vec{\mathbf{u}}$. The second

method is element-wise product of modes and frequencies. In the tensor representation (13) the element-wise products are the following

$$\partial_x \mathbf{u} = \begin{bmatrix} -q_k c_{jk} & q_k a_{jk} \\ -q_k d_{jk} & q_k b_{jk} \end{bmatrix} \qquad \partial_{xx} \mathbf{u} = \begin{bmatrix} -q_k^2 a_{jk} & -q_k^2 c_{jk} \\ -q_k^2 b_{jk} & -q_k^2 d_{jk} \end{bmatrix}$$

$$\partial_{xxx} \mathbf{u} = \begin{bmatrix} q_k^3 c_{jk} & -q_k^3 a_{jk} \\ q_k^3 d_{jk} & -q_k^3 b_{jk} \end{bmatrix} \qquad \partial_{xxxx} \mathbf{u} = \begin{bmatrix} q_k^4 a_{jk} & q_k^4 c_{jk} \\ q_k^4 b_{jk} & q_k^4 d_{jk} \end{bmatrix}$$

$$\partial_t \mathbf{u} = \begin{bmatrix} 0 & 0 \\ -\omega_j b_{jk} & -\omega_j d_{jk} \\ \omega_j a_{jk} & \omega_j c_{jk} \end{bmatrix} \tag{73}$$

The formulation of the Kuramoto-Sivashinsky equation (**??**) in terms of modes will be a *pseudospectral* formulation. The nonlinearity will be computed as an element-wise product in physical space as opposed to a convolution in spectral space. In operator notation the Kuramoto-Sivashinsky equation (**??**) is written

$$f(\mathbf{v}) \equiv (\partial_t + \partial_{xx} + \partial_{xxxx})\mathbf{u} + \frac{1}{2}\partial_x \mathcal{F}((\mathcal{F}^{-1}(\mathbf{u}) \cdot \mathcal{F}^{-1}(\mathbf{u}))\,. \tag{74}$$

Because they are never actually calculated the convolution sums will not be derived; it is much easier to express this in terms of components of the products in physical space. Using the derivatives in tensor notation (73), (**??**) the linear component of the Kuramoto-Sivashinsky equation as a system of differential algebraic equations is written (in the tensor format) as

$$\begin{bmatrix} (-q_k^2 + q_k^4)a_{jk} - \omega_j b_{jk} & (-q_k^2 + q_k^4)c_{jk} - \omega_j d_{jk} \\ (-q_k^2 + q_k^4)b_{jk} + \omega_j a_{jk} & (-q_k^2 + q_k^4)d_{jk} + \omega_j c_{jk} \end{bmatrix} \tag{75}$$

Computation of the nonlinear component of $f(\mathbf{v})$ is performed in physical space as an element-wise product, as opposed to a (2-d) convolution in Fourier space. This can easily be computed using a *pseudospectral* method; which is represented in operator notation as

$$\tilde{N} \equiv \mathcal{F}(u \cdot u) \equiv \mathcal{F}(\mathcal{F}^{-1}(\mathbf{u}) \cdot \mathcal{F}^{-1}(\mathbf{u}))\,. \tag{76}$$

The expression (76) clearly does not specify the blocks, however; this is derived for posterity before moving on.

The product in physical space u^2 corresponds to convolutions of the products of sums in `rfft`e-rfft in spectral space. Because convolution, Fourier transforms are linear transformations, the product u^2 is equal to the sum of the products of each of its components. Each block of modes corresponds to a spatiotemporal parity of its corresponding field. For example, the field $\mathcal{F}^{-1}(a_{jk})$ would be even with respect to $T/2$ and $L/2$ because the basis functions are cosines. The parity of the product can easily be determined by the parity of products of sines and cosines. For example, the parity of the product $\mathcal{F}^{-1}(a_{jk}) * \mathcal{F}^{-1}(c_{jk})$ results from the parity of the products of the general form which can be crudely represented by $\cos(\omega_j t_n)\cos(q_k x_m) * \cos(\omega_j t_n)\sin(q_k x_m)$. The product is therefore odd with respect to

space and even with respect to time; the correspondence is then $\cos(\omega_j)\sin(q_k)$ hence it lies in the $\tilde{N}_{c_{jk}}$ mode block. Applying this logic to all other products yields an expression of the components of (76); let $\cdot$ represent the element-wise product.

$$\begin{bmatrix} \tilde{N}_{a_{jk}} & \tilde{N}_{c_{jk}} \\ \tilde{N}_{b_{jk}} & \tilde{N}_{d_{jk}} \end{bmatrix} = \begin{bmatrix} -q_k\mathcal{F}(a\cdot c + b\cdot d)) & \frac{q_k}{2}\,\mathcal{F}((a\cdot a)+(b\cdot b)+(c\cdot c)+(d\cdot d))) \\ -q_k\,\mathcal{F}(a\cdot b + c\cdot d)) & q_k\,\mathcal{F}(a\cdot d + b\cdot c)) \end{bmatrix}$$
$$\text{where}$$
$$a = \mathcal{F}^{-1}(a_{jk}),\ b = \mathcal{F}^{-1}(b_{jk}),\ c = \mathcal{F}^{-1}(c_{jk}),\ d = \mathcal{F}^{-1}(d_{jk})\,. \tag{77}$$

The dependence of (74) on the parameters T, L is implicit by definition of q_k, ω_j. The spatial derivative within the nonlinear term is inserted between the temporal and spatial transform for details relevant to the As a shorthand notation is can be convenient to condense (74) into a more compact notation

$$f(\mathbf{v}) \equiv \begin{bmatrix} (-q_k^2+q_k^4)a_{jk} - \omega_j b_{jk} + \tilde{N}_{a_{jk}} & (-q_k^2+q_k^4)c_{jk} - \omega_j d_{jk} + \tilde{N}_{c_{jk}} \\ (-q_k^2+q_k^4)b_{jk} + \omega_j a_{jk} + \tilde{N}_{b_{jk}} & (-q_k^2+q_k^4)d_{jk} + \omega_j c_{jk} + \tilde{N}_{d_{jk}} \end{bmatrix} \tag{78}$$

This is the form of (**??**) in terms of the spatiotemporal mode blocks of (13).

This represents the explicit expression in the spatiotemporal mode basis, however, symmetries have yet to be incorporated. Before deriving the variants of (78) a modification to the pseudospectral product must be made. Prior to any modifications, the nonlinear term (74) for modes constrained to either $\mathbb{U}^0$, $\mathbb{U}^\sigma$, or $\mathbb{U}^{\sigma\tau}$ (45) yields $P^i\mathcal{F}(\mathcal{F}^{-1}(\mathbf{u}^2)) = \mathbf{0}$. The projection operator P^i is made explicit here but is typically baked into the time transform. In other words, the modes $\mathcal{F}(\mathcal{F}^{-1}(\mathbf{u}^2))$ lie in the complement to whichever invariant subspace is being considered. This can be corrected however by exchanging the order of the spatial derivative and projection operators using the relations (**??**). The accommodation employed is to do just this; take the spatial derivative in the spatial mode basis, preempting the time transform. With this modification we are now fully equipped to handle the symmetry variants of (78).

In the $\mathbb{U}_0$ subspace of antisymmetric equilibria the only non-zero modes are given by (56); discarding all vanishing modes from (78) yields

$$\mathbf{f}_{\mathbb{U}^0}(\mathbf{v}) \equiv \left[(-q_k^2+q_k^4)c_{0k} + \tfrac{1}{2}\mathcal{F}_t(q_k\mathcal{F}_x(\mathcal{F}^{-1}(c_{0k})\cdot\mathcal{F}^{-1}(c_{0k})))\right] \tag{79}$$

Continuing on with the antisymmetric subspace $\mathbb{U}^\sigma$ using the selection rules (**??**)

$$\mathbf{f}_{\mathbb{U}^\sigma}(\mathbf{v}) \equiv \begin{bmatrix} (-q_k^2+q_k^4)c_{jk} + \omega_j d_{jk} + \frac{1}{2}\mathcal{F}_t(q_k\mathcal{F}_x((c\cdot c)+(d\cdot d)) \\ (-q_k^2+q_k^4)d_{jk} - \omega_j c_{jk} + \mathcal{F}_t(q_k\mathcal{F}_x(c\cdot d)) \end{bmatrix}. \tag{80}$$

Unfortunately there is no useful simplification for the shift reflection subspace as each block has non-vanishing modes; however it should be noted that the components of (**??**) also obey the selection rules (58).

For relative periodic orbits and relative equilibria, substitution of the comoving frame ansatz (69) into (**??**) generates an additional linear term via the time derivative of terms

with $x_m - \frac{St_n}{T}$. The result can explicitly be represented by action of the operator $\frac{S}{T}\partial_x$

$$
\begin{aligned}
f_S(\mathbf{v}) \equiv & \begin{bmatrix} (-q_k^2+q_k^4)a_{jk} - \omega_j b_{jk} + \tilde{N}_{a_{jk}} & (-q_k^2+q_k^4)c_{jk} - \omega_j d_{jk} + \tilde{N}_{c_{jk}} \\ (-q_k^2+q_k^4)b_{jk} + \omega_j a_{jk} + \tilde{N}_{b_{jk}} & (-q_k^2+q_k^4)d_{jk} + \omega_j c_{jk} + \tilde{N}_{d_{jk}} \end{bmatrix} \\
& + \begin{bmatrix} -q_k \frac{S}{T} c_{jk} & q_k \frac{S}{T} a_{jk} \\ -q_k \frac{S}{T} d_{jk} & q_k \frac{S}{T} b_{jk} \end{bmatrix}
\end{aligned} \tag{81}
$$

For a $\mathbf{v}$ the equation (74) is comprised of the same number of components as non-vanishing modes; there are no components for the parameters T, L, S. For a state $\mathbf{v}$ with $|\mathbf{u}|$ non-vanishing modes and p parameters

These equations can also be written in terms of matrices, this is useful for the derivation of the Jacobian matrix later on.

I was hung up on how to actually compute these "matrix-vector approximations" in my context, after some exploring ref. [4, 16, 56]. I think I've figured them out but I plan on talking to J. F. Gibson to confirm. The other things I am trying to reconcile between the different notations is how López *et al.* [40] handles symmetries of solutions as it's slightly different than when one has a forward time mapping, I believe. These things and working in the class objects I've defined in python turned out to be somewhat trickier than I had first intended but I hope to get things up and running by the weekend. Today's dealings mainly involved working through the "direct matrix method" in ref. [12]. In order to explicitly construct the Jacobian of $f(\mathbf{v})$ and its adjoint it will be helpful to construct (74) using matrices where possible. The three general cases are considered; orbits without symmetry, discrete symmetry, and continuous symmetry. Element-wise multiplication can be represented by multiplication of a vector with a diagonal matrix; however, it will be written in pseudospectral form for reasons relevant to the gradient of $f(\mathbf{v})$. Much like (74) the nonlinear term is written in a term to accomodate orbits with discrete symmetries, $\tilde{\mathbf{M}}[\partial_x] = \mathbf{M}[\mathcal{F}_x^{-1}]\mathbf{M}[\partial_x]\mathbf{M}[\mathcal{F}_x]$

$$
\begin{aligned}
f(\mathbf{v}) &= (\mathbf{M}[\partial_t] + \mathbf{M}[\partial_{xx}] + \mathbf{M}[\partial_{xxxx}])\vec{\mathbf{u}} + \frac{1}{2}\mathbf{M}[\mathcal{F}_t]\,\tilde{\mathbf{M}}[\partial_x]\,\mathbf{M}[\mathcal{F}_x]\,(\mathcal{F}^{-1}(\vec{\mathbf{u}}) \cdot \mathcal{F}^{-1}(\vec{\mathbf{u}})) \\
f_S(\mathbf{v}) &= f(\mathbf{v}) + \frac{S}{T}\mathbf{M}[\partial_x]\vec{\mathbf{u}}
\end{aligned} \tag{82}
$$

Solving for the roots of the expression (74) and its symmetry subspace variants are the core of all spatiotemporal computations. To do so it will be very important to derive the gradient of $f(\mathbf{v})$with respect to all variables represented in $\mathbf{v}$. The gradient of $f(\mathbf{v})$ is defined as The resulting matrix is non-square as there as there are no components of $f(\mathbf{v})$for the parameters.

$$
\begin{aligned}
\nabla_{\mathbf{v}} f(\mathbf{v}) &= \begin{bmatrix} \nabla_{\mathbf{u}} f(\mathbf{v}) & \nabla_{\mathrm{T}} f(\mathbf{v}) & \nabla_{\mathrm{L}} f(\mathbf{v}) \end{bmatrix} \\
\nabla_{\mathbf{v}} \mathbf{f}_0(\mathbf{v}) &= \begin{bmatrix} \nabla_{\mathbf{u}} \mathbf{f}_0(\mathbf{v}) & \nabla_{\mathrm{L}} \mathbf{f}_0(\mathbf{v}) \end{bmatrix} \\
\nabla_{\mathbf{v}} f_S(\mathbf{v}) &= \begin{bmatrix} \nabla_{\mathbf{u}} f_S(\mathbf{v}) & \nabla_{\mathrm{T}} f_S(\mathbf{v}) & \nabla_{\mathrm{L}} f_S(\mathbf{v}) & \nabla_{\mathrm{S}} f_S(\mathbf{v}) \end{bmatrix}
\end{aligned} \tag{83}
$$

These Jacobian matrices are defined by $|\mathbf{u}|$ equations (number of modes) and $|\mathbf{u}| + p$ unknowns (modes and parameters).

For an arbitrary matrix $\mathbf{A}$ (independent of $\mathbf{u}$), functions $\mathbf{F}(\mathbf{u}), \mathbf{G}(\mathbf{u})$, the relevant chain and product rules are

$$\begin{aligned} \frac{\partial}{\partial \mathbf{u}}(\mathbf{A}\mathbf{u}) &= \mathbf{A} \\ \frac{\partial}{\partial \mathbf{u}}(\mathbf{F}(\mathbf{u}) \cdot \mathbf{G}(\mathbf{u})) &= \text{Diag}[G(\mathbf{u})]\frac{\partial \mathbf{F}(\mathbf{u})}{\partial \mathbf{u}} + \text{Diag}[\mathbf{F}(\mathbf{u})]\frac{\partial \mathbf{G}(\mathbf{u})}{\partial \mathbf{u}} \\ \frac{\partial}{\partial \mathbf{u}}(\mathbf{A}\mathbf{F}(\mathbf{u})) &= \mathbf{A}\frac{\partial F(\mathbf{u})}{\partial \mathbf{u}} \end{aligned} \tag{84}$$

The gradient $\mathbf{J} \equiv \nabla_{\mathbf{u}} f(\mathbf{v})$ is written in terms of matrices as opposed to individual matrix elements (74) will be used to write. $\mathbf{J}$ will be used

$$\begin{aligned} \nabla_{\mathbf{u}} f(\mathbf{v}) &= \mathbf{M}[\partial_t] + \mathbf{M}[\partial_{xx}] + \mathbf{M}[\partial_{xxxx}] + \mathbf{M}[\mathcal{F}_t]\tilde{\mathbf{M}}[\partial_x]\, \mathbf{M}[\mathcal{F}_x]\, \mathbf{M}[\mathcal{F}^{-1}(\vec{\mathbf{u}})]\, \mathbf{M}[\mathcal{F}^{-1}] \\ \nabla_{\mathbf{u}} f_S(\mathbf{v}) &= \nabla_{\mathbf{u}} f(\mathbf{v}) + \frac{S}{T}\mathbf{M}[\partial_x] \end{aligned} \tag{85}$$

where $\text{Diag}[\mathcal{F}^{-1}(\vec{\mathbf{u}})]$ is a diagonal matrix with u as its elements. This result is derived from what [12] denotes as 'direct-matrix notation'.

The derivatives with respect to the parameters T, L, S apply to the frequencies within (74). All of the terms with dependence on T and L have inverse proportionality therefore the derivatives can always be written, for example, as $\frac{-p}{L} q_k^p$. This is the convention used here; computationally it allows for scalar multiplication of previously defined functions. If the symmetry variants of the equations are no different from the general expression or are not relevant (i.e. time is not relevant for equilibria) then they are not defined.

$$\begin{aligned} \nabla_{\mathrm{T}} f(\mathbf{v}) &= -\frac{1}{T}\mathbf{M}[\partial_t]\vec{\mathbf{u}} \\ \nabla_{\mathrm{T}} f_S(\mathbf{v}) &= \nabla_T f(\mathbf{v}) - \frac{S}{T^2}\mathbf{M}[\partial_x]\vec{\mathbf{u}} \end{aligned} \tag{86}$$

$$\begin{aligned} \nabla_{\mathrm{L}} f(\mathbf{v}) &= (-\frac{2}{L}\mathbf{M}[\partial_{xx}] - \frac{4}{L}\mathbf{M}[\partial_{xxxx}])\vec{\mathbf{u}} - \frac{1}{2L}\mathbf{M}[\mathcal{F}_t]\tilde{\mathbf{M}}[\partial_x]\, \mathbf{M}[\mathcal{F}_x]\, (\mathcal{F}^{-1}(\vec{\mathbf{u}}) \cdot \mathcal{F}^{-1}(\vec{\mathbf{u}})) \\ \nabla_{\mathrm{L}} f_S(\mathbf{v}) &= \nabla_L f(\mathbf{v}) - \frac{1}{L}\frac{S}{T}\mathbf{M}[\partial_x]\vec{\mathbf{u}} \end{aligned} \tag{87}$$

$$\nabla_{\mathrm{S}} f_S(\mathbf{v}) = \frac{1}{T}\mathbf{M}[\partial_x] \tag{88}$$

Another component required for the numerical techniques is to derive the adjoint of (85). For now, this will be accomplished by taking the transposition of (85) this was most easily done in the context of *formal Lagrangians*.

For now, it is suitable to reverse engineer the adjoint equation by simply taking the transpose of (85). As the matrices are all either skew-symmetric, diagonal, or orthogonal with respect to the respective subspace these can be derived as

$$\begin{aligned} \nabla_{\mathbf{u}}^{\top} f(\mathbf{v}) &= -\mathbf{M}[\partial_t] + \mathbf{M}[\partial_{xx}] + \mathbf{M}[\partial_{xxxx}] - \mathbf{M}[\mathcal{F}^{-1}]\, \mathbf{M}[\mathcal{F}^{-1}(\vec{\mathbf{u}})]\, \mathbf{M}[\mathcal{F}_x^{-1}]\tilde{\mathbf{M}}[\partial_x]\, \mathbf{M}[\mathcal{F}_t^{-1}] \\ \nabla_{\mathbf{u}}^{\top} f(\mathbf{v})_S &= \nabla_{\mathbf{u}}^{\top} f(\mathbf{v}) - \frac{S}{T}\mathbf{M}[\partial_x] \end{aligned} \tag{89}$$

Unlike the Jacobian (85) the adjoint is rarely ever explicitly constructed. Instead, it is the product of matrix and vector which is important. In conventional numerical studies, matrix-vector products most commonly arise as finite-difference approximations of tangent space evolution. The Jacobian in the dynamical systems context maps a linear neighborhood forward in time,

On these comments, the main references I have been studying are refs. [7, 11, 31, 40].

in my context, after some exploring ref. [4, 16, 56]. I think I've figured them out but I plan on talking to J. F. Gibson to confirm. The other things I am trying to reconcile between the different notations is how López *et al.* [40] handles symmetries of solutions as it's slightly different than when one has a forward time mapping, I believe. These things and working in the class objects I've defined in python turned out to be somewhat trickier than I had first intended but I hope to get things up and running by the weekend.

You can think of the way it was traditionally done in continuing solutions. First we fixed the spatial domain to a constant L, varied T to $T+\delta T$ in order to find a spatiotemporally periodic solution $u^{(1)}(x,t)$. Then we increased the spatial domain size L to $L+\delta L$, and used $u^{(1)}(x,t)$ (the same N Fourier modes) as a starting guess, varied T to $T+\delta T$ and determines the spatiotemporally periodic solution $u^{(2)}(x,t)$, parametrized by $(L+\delta L, T+\delta T)$.

As a reminder, the intention of this was to allow us to rewrite (74) in a much less confusing manner. The specific form we utilize is what is known as a *pseudospectral* form [10]. The general motivation is that explicit computation of the nonlinearity via summation is not only inefficient but also susceptible to cancellation error. Pseudospectral methods compute the quadratic nonlinearity as a pointwise product in physical space which is equivalent to the spectral convolution. Note that this extra work is merely to increase the transparency of our numerical work; in practice the linear operators are not constructed once the memory cost becomes too great, rather, we deploy functions which have an equivalent effect. To begin, let us define the linear operators mentioned shortly before. Every such operator can be described by the following prescription: construct the operator that would act on a single point in either space or time, then extend this operator to act on the entire spatiotemporal domain at once. Because we have ordered the Fourier coefficients in a contiguous manner this extension always takes the form of a Kronecker outer product with an appropriately sized identity matrix. This outer product is defined for two matrices A and B of sizes $N_a \times M_a$ and $N_b \times M_b$ as

There is a storm in the distance however, as this general procedure is ruined for the spatial problem. As we know from the chronotopic literature refs. [18, 37, 38, 55], that iteration in space typically does not converge to the same attractor as iteration in time, and generally corresponds to a strange repeller. Therefore I cannot hope to form an initial guess loop from using a Poincaré section in the spatial direction, as typically all of my Fourier coefficients go off to infinity before a recurrence is found.

I thought that I would have to somehow permute the elements (145) of the "Loop Vector" (vector that encodes the parameterization of initial condition for periodic orbit search). The reasoning behind this was in order to use differentiate with respect to a parameterization variable s, I would need the elements to be in sequential order with respect in parameterization variable s, in order to multiply by vector $i\vec{m}$, where m is the conjugate variable (in a Fourier transform sense) to s. This is ***not*** the case, as I can merely exploit the Kronecker outer product to produce a diagonal matrix such that along the diagonal there are M duplicates of each element of $\vec{m}$

Began parsing through the literature on (variational) newton descent, specifically refs. [13, 35] and for invariant tori ref. [36].

A useful class of numerical methods often used in optimization [6, 15] are known as *descent methods*. In our context, optimization denotes the numerical process of finding minimizers of a scalar valued cost function which vector valued inputs, the spatiotemporal Fourier modes and spatiotemporal parameters. Broadly speaking, descent methods are numerical methods which iteratively solve unconstrained optimization. These methods accomplish this by one way or another providing a direction to step in which monotonically decreases the value of the cost functional, hence "descent" (assuming a non-negative scalar cost function). The method with which to compute the descent direction is the distinguishing property between descent methods. In the limit of an infinitesimal step size, the iterative descent can be characterized as a fictitious flow with respect to a fictitious time [36]. The advantages of descent methods are that they do not require the construction nor the inversion of any matrices. The computational and memory requirements are relatively cheap in comparison to direct methods but the trade off is the rate of convergence.

2.4 Variational Methods

We still refer to the numerical method as the adjoint descent method, because the calculation does require the multiplicative action of the adjoint Jacobian. For the convergence analysis of the method we refer to proofs for the steepest descent method. Define the Lagrangian density or *formal Lagrangian* as

$$\mathcal{L}(t, x, p_i, u, \lambda, u_t, u_x, u_{xx}, u_{xxxx}) = \frac{1}{2}\lambda[u_t + u_{xx} + u_{xxxx} + uu_x], . \tag{90}$$

This density is a function defined on the tile $\mathcal{M}$of parameters p_i, dependent parameter u and its relevant higher order derivatives, and finally the *adjoint variable* λ. In this context λ is analogous to a Lagrange multiplier; it is taken to only be dependent on t, x. The problem of finding solutions to $f(\mathbf{v}) = 0$ is reformulated as a variational problem whose goal is to find the stationary points of the functional equation

$$S(\mathbf{v}) = \int_{\mathcal{M}} \mathcal{L}(t, x, p_i, u, \lambda, u_t, u_x, u_{xx}, u_{xxxx}) \, . \tag{91}$$

The specific form of ϕ tells us that subject to the constraint $f(\mathbf{v}) = 0$, the extrema of ϕ will all be minima, as ϕ is non-negative and $f(\mathbf{v}) = 0$ implies $\phi = 0$. The stationary points of (91) are solutions to the Euler-Lagrange equations, which are derived via the functional derivative (this goes by many names, material derivative, total derivative, advective derivative, variational derivative) of (90). The general definition of the functional derivative for a Lagrangian with f_i dependent variables in q_i dimensions with derivatives up to order j is

$$\frac{d}{df_i} = \frac{\partial}{\partial q_i} + \sum_{j=1}^{n} \sum_{\mu_1 \leq \cdots \leq \mu_j} (-1)^j \frac{\partial^j}{\partial q_{\mu_1} \ldots \partial q_{\mu_j}} \left(\frac{\partial}{\partial f_{i,\mu_1 \ldots \mu_j}} \right) \tag{92}$$

where d is used to indicate the functional derivative and ∂ indicates the partial derivatives. The bounds on the inner sum simply take care of repeating terms arising from the permutations of derivatives. For clarity (92) is simply a generalization of the Euler-Lagrange equations to more variables and higher order derivatives. This general expression can be confusing at first however it simplifies dramatically upon application to (90). Namely, the Kuramoto-Sivashinsky equation and its adjoint equation as well as the derivatives with

respect to parameters are recovered

$$\begin{aligned}
\frac{d\mathcal{L}}{du} &= \left(\frac{\partial}{\partial u} - \frac{\partial}{\partial x}\frac{\partial}{\partial u_x} - \frac{\partial}{\partial t}\frac{\partial}{\partial u_t} + \frac{\partial^2}{\partial x^2}\frac{\partial}{\partial u_{xx}} + \frac{\partial^4}{\partial x^4}\frac{\partial}{\partial u_{xxxx}}\right)\mathcal{L} \\
&= -\lambda_t + \lambda_{xx} + \lambda_{xxxx} + u\lambda_x \\
\frac{d\mathcal{L}}{d\lambda} &= f(\mathbf{v}) \\
\frac{d\mathcal{L}}{dp_i} &= \lambda\frac{\partial f(\mathbf{v})}{\partial p_i}
\end{aligned} \tag{93}$$

The system of equations equations are satisfied when $\lambda^\top = f(\mathbf{v})^\top$. This specific choice of λ for the Lagrangian defines

$$(\mathbf{??})S(\mathbf{v}) \equiv \int_{\mathcal{M}} \frac{1}{2}(f(\mathbf{v}))^2 dt d^d x \tag{94}$$

A distinction is made between (**??**) and its discrete representation in terms of modes

$$\phi(\mathbf{v}) = \frac{1}{2}\sum_{j,k}(f(\mathbf{v})_{jk})^2\,. \tag{95}$$

Hereafter, the function (95) will be referred to as the *cost function*; the value returned by evaluated the cost function will be referred to as the *residual*.

An 'orbit guess' is defined as the state vector whose residual is non-zero. The 'orbit guess' $\mathbf{v}^{'}$ is a state vector whose residual is non-zero. As a blanket term *the optimization* of a guess state $\mathbf{v}^{'}$ denotes the process of using numerical algorithms to create the sequence (**??**) to $\mathbf{v}^{'}$ in efforts to bring its residual to 0. The optimization of $\mathbf{v}^{'}$ will be said to have *converged* if the residual is brought to a value smaller than the *tolerance*, ϵ.

The goal then becomes clear; starting from using numerical algorithms create a sequence of

CHAPTER III

NUMERICAL OPTIMIZATION

3.1 Gradient descent

While this theoretically guarantees convergence it does not guarantee the *rate* of convergence. The fictitious time derivative's dependence on $G(u)$ necessarily indicates that as magnitude of the cost functional approaches zero so does the gradient. Therefore, it is like that the adjoint descent method becomes less effective as it approaches a solution.

The intent is that these details will not serve as a guideline for the reader but also demonstrate that the implementation is relatively easy and straightforward. To begin our computational discussion, we note that even though the descent direction includes the adjoint of the Jacobian, $\frac{\partial F}{\partial u}^\top$, this matrix is never explicitly formed. This is important because the computational cost, in terms of time and resources, becomes prohibitive as the dimensionality of the system increases. Instead, we only ever compute the matrix-vector *product* that appears in (98). This is done in a matrix-free manner without the use of finite-difference approximations found in [8, 31].

Computation of (**??**) in a matrix-free manner is much more efficient than explicitly constructing the adjoint matrix. All that is left is to numerically integrate (**??**) until some error tolerance is met. Because the intermediate states represent approximate solutions the accuracy of such states has no meaning, we only require that the cost functional (**??**) is decreasing. While the adjoint descent does guarantee this the integration scheme we employ, explicit Euler method, does not. The reason for using such a simple method is merely due to its speed; one could apply a higher order method but once again, the accuracy of intermediate states is not important. We have found that decreasing the step size is sufficient to ensure that the cost functional always decreases. This covers the basic implementation of the adjoint descent method. There are other modifications which can be made that improve the rate of convergence. The only change that seems to have a drastic effect is known as preconditioning . The discussion regarding preconditioning, its formulation and effects, is located in sect. **??**. Differentiation of (95) with respect to fictitious time τ induces the fictitious flow which defines the numerical descent method.

$$\partial_\tau \phi(\mathbf{v}) = [(\nabla f(\mathbf{v}))^\top f(\mathbf{v})]^\top \partial_\tau \mathbf{v} \tag{96}$$

The term $\partial_\tau \mathbf{v}$ is free to be defined by the numerical method. The specific selection

$$\partial_\tau \mathbf{v} = -[(\nabla f(\mathbf{v}))^\top] f(\mathbf{v}) \tag{97}$$

defines the gradient descent method as (**??**) equals $-\nabla\phi$. This choice ensures that the cost function is non-increasing as a function of τ

$$\partial_\tau \phi(\mathbf{v}) = -([\nabla^\top f(\mathbf{v})] f(\mathbf{v}))^2 < 0\,. \tag{98}$$

The iteration (**??**) is the most basic explicit method that could be applied to this problem. There are a number of motivations behind this choice. Numerically (**??**) is fast and, practically speaking, the residual can is preferred the intermediate states do not matter so long as the residual is decreasing, as this stipulations

This is implemented numerically using Euler's method. Let $\mathbf{v}^{'}$ represent an initial guess for which $\phi(\mathbf{v}^{'}) \neq 0$. The sequence of corrections

$$(??)\mathbf{v}^{'}_{n+1} = \mathbf{v}^{'}_{n} - \nabla\phi(\mathbf{v})\Delta\tau \tag{99}$$

approaches a minima of ϕ; if this happens to be a local minimum then the optimization of $\mathbf{v}^{'}$ will be considered a failure. The distinction is made between this numerical integration and time evolution; in this case the steps are in a fictitious time direction and represent successive variational corrections. The only requirement that is stipulated is that the residual is non-increasing. The sequence of intermediate states parameterized by discrete τ are not of consequence, as any state that does not satisfy (74) is not a solution. Therefore the choice (**??**) is made as to favor computational speed over computational accuracy. Define the residual of a state $\mathbf{v}^{'}$ near an orbit $\mathbf{v}$ as the following.

' To ensure that the residual is non-increasing, τ must be chosen to be sufficiently small, and is chosen by the following prescription. Let $\tau_0 = 1$, while $r(\mathbf{v}^{'}_{n+1}) = r(\mathbf{v}^{'}_{n} - \nabla\phi(\mathbf{v})\Delta\tau) \geq r(\mathbf{v}^{'}_{n})$, set $\tau_{j+1} = tau_j/2$. Computationally the smallest accepted τ_j is chosen to be quite small, $\min\tau = 10^{-8} \approx 2^{-26}$ however in practical terms the termination of (**??**) is controlled by the condition

$$r_{n+1} \tag{100}$$

such that the optimization process is subject to the following condition

$$(??)\frac{r_n - rn + 1}{\max[1, r_n]} \geq \epsilon_f \tag{101}$$

if for whatever reason (**??**)

These conditions are similar but less precise than the Wolfe conditions or Armijo-Goldstein conditions; the optimization considered here does not concern itself with finding an optimal step length as defined by a line-searching procedure.

3.2 Least-squares Newton

Technically this equation is solved iteratively, each time producing its own least-squares solution which guides the field to invariant 2-torus. The equations are augmented to include variations in L, T and as such the linear system is actually rectangular. We chose to solve the equations in a least-squares manner as we are not focused on finding a unique solution; any member of a invariant 2-tori group orbit will do. The price of this indefiniteness is that we might collect invariant 2-tori which belong to the same group orbit. To improve the convergence rate of the algorithm we also include backtracking: the length of the Newton step is reduced until either a minimum length is reached (failure) or the cost function decreases.

The second method is application of a least-squares solver to the root finding problem $F = 0$. The Newton system is derived here for context.

$$\mathbf{f}(\mathbf{v} + \delta\mathbf{v}) \approx f(\mathbf{v}) \ + \nabla f(\mathbf{v}) \ \delta\mathbf{v} \ + \mathcal{O}(\delta\mathbf{v}^2)\,. \tag{102}$$

substitution of zero for the LHS (the root) yields

$$\nabla f(\mathbf{v})\delta\mathbf{v} = -f(\mathbf{v})\,. \tag{103}$$

where $\nabla f(\mathbf{v})$ is given by (85) and its symmetry variants. The equation (103) will be referred to as the *Newton equations* of (74). The newton equations can be solved by either iterative or direct methods.

solve linear systems $\mathbf{Ax} = \mathbf{b}$ by creating the inverse (or pseudo-inverse) of $\mathbf{A}$. The underdetermined nature of the equations Because the period and spatial domain size are maintained as variables the linear system of equations is represented by a rectangular matrix, which can be solved by either applying a least-squares solver or forming the *normal equations* which creates a square linear system by multiplying by the transpose of the rectangular matrix.

As A in our case is already ill-conditioned because of the stiff linear components; that is, the higher order spatial derivatives dominate the spectrum of eigenvalues by manifesting as diagonal matrices with large magnitudes when compared to the typical matrix element magnitude. Because of this, the product $A^T A$ would be affected in turn. The linear system in question is the Newton system derived from the linear expansion of the root finding problem $F = 0$.

$$\text{minimize}||Ax - b|| \tag{104}$$

In practice (104) is solved iteratively by constructing the Moore-Penrose pseudo-inverse, which provides the least-squares solution to (104) of minimum norm.

$$\min_{\Delta \mathbf{z}_i} ||DG(\mathbf{z}_{i-1})\Delta \mathbf{z}_i + \mathbf{G}(\mathbf{z}_{i-1})||_2 \tag{105}$$

for $i = 1, 2, 3, ...$ This produces a sequence of approximate solutions $\{\mathbf{z}_i = \mathbf{z}_{i-1} + \Delta \mathbf{z}_{i-1}\}$ which we require to monotonically decrease the value of the cost functional (**??**). If this monotonic behavior is violated then we deem the trial a failure. Some common ways of reducing the likelihood of failure include introducing a line-search, trust-region, and backtracking techniques [15]. We elect to introduce a scalar "damping" parameter ρ which modifies the magnitude of each step produced by any of our iterative methods. The recursive equation for the sequence of approximate solutions to (105)

$$\mathbf{z}_i = \mathbf{z}_{i-1} + \rho_{i-1}^n \Delta \mathbf{z}_{i-1} \tag{106}$$

where

$$\rho_i^n = \frac{1}{2^n} \tag{107}$$

for $n \in \mathbb{Z}^+$. In practice, if $F(\mathbf{z}_i) > F(\mathbf{z}_{i-1})$ then n is increased until either the cost functional's value decreases $F(\mathbf{z}_i) < F(\mathbf{z}_{i-i}$, or $\rho_i^n = 1/2^8$.

This proceeds until either the cost functional is within double floating point precision of zero or the magnitude $|z_i|$ becomes too small. When computationally viable we elect to use a direct method to solve (**??**). This entails using a subroutine which computes the pseudo-inverse of the matrix DG. This process has been optimized by the creators of LAPACK, a Fortran package. Specifically we use a SciPy's implementation of the linear least squares solver (LSTSQ) which uses the GELSD driver. This driver solves the linear least squares problem via a divide and conquer singular value decomposition (SVD) process to compute the pseudoinverse.

This method of solving the least-squares problem established by construction of the Newton system [6, 58, 59]

3.3 Combination of gradient descent and least-squares Newton

CHAPTER IV

SPATIOTEMPORAL TECHNIQUES

CHAPTER V

RESULTS

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

APPENDIX A

NUMERICAL DETAILS

Extracted information from ref. [10] on what I thought was one of the gaps in my code which was the fact that it seemed to perform much better as a fully aliased pseudospectral calculation rather than a de-aliased calculation. I feared that this would bring criticism so as I tried to armor myself by actually *reading*. My argument is that while aliasing can be devastating for temporal evolution due to the contamination of the higher Fourier mode components by their aliases, ($k' = k + N$) where N is the number of collocation points, the spatiotemporal problem should be fine as long as the calculation is well resolved enough. In fact, this is likely why I have solutions that "converge" on larger spatial domains than the discretizations can seem to resolve.

Aliasing commments:

> For evolution problems one must address the issue of the temporal numerical stability of the calculation. Collocation approximations must be formulated with more care than Galerkin approximations. The reason is that for evolution problems with quadratic conservation properties, the Galerkin formulation will automatically yield semi-discrete quadratic conservation laws.
>
> Numerous comparisons have been performed for aliased and de-aliased calculations of the periodic, multidimensional Navier-stokes equations. Useful discussions may be found in [17, 30, 47, 51]. All of these authors conclude that with sufficient resolution, aliased calculations are quite acceptable.
>
> Moser, Moin and Leonard [48] caution against aliased calculations. They present a single, poorly resolved, aliased calculation and compare it with three de-aliased calculations, one poorly resolved, one moderately resolved, and one well resolved. Their single aliased result is certainly much worse than their well resolved, de-aliased case, but their poorly resolved, de-aliased case is no better than the aliased one. Hence, their conclusion is not supported by their evidence.

In light of these comments and discussions I believe that de-aliasing is more important in the context of accurate temporal evolution, but not required in the spatiotemporal fixed point problem as long as the patterns are sufficiently resolved. One way of thinking about this is that in the discretization of the spatiotemporal Kuramoto-Sivashinsky equation we could add a term that represents the aliasing,

as it is implicit in the fully aliased representation of the equations.

I believe the spectrum of the Kuramoto-Sivashinsky equation plays a role, and it might be wise to at least dealias the temporal convolution sum as there is less of a precedent for ignoring it; In the spatial case we can at least claim the hyperdiffusion term diminishes the amount of corruption in the spatial wave number, but this is harder to motivate for the temporal terms. More motivation from Canuto, Hussaini, Quateroni and Zhang [10] are their fig. 7.1 and fig. 7.4, where the fully aliased but more resolved terms seem to beat out even the dealiased computations in energy conservation of the KdV equation for fig. 7.1.

Their fig. 7.4 is a reproduction of the effects of aliasing in the transition to turbulence in channel flow by Krist and Zang [33]. Only the high resolution (aliased) seems to be physically representative of the actual solution, and even the dealiased computation on a coarser discretization (while better than the equivalent aliased discretization) still does not prevent artificial oscillations. This is also a temporal evolution problem which we are not dealing with.

Implement a number of various means for rediscretization, reordered a bunch of processes. List of rediscretization processes, "Residual guided rediscretization": Lower N and M while $|F|^2$ is decreasing "Converged Solution guided rediscretization": Lower N to minimum value dependent on period, incrementally increase until solution converges, then do the same with M.

The first of these is the cheap, expedient procedure that should be used to produce initial conditions that have a smaller discretization that one started with; The second tries is a much more expensive procedure because it tries to find the minimal discretization for *converged* solutions; therefore, it requires a bit of computational time. From testing it seems much better to increase the discretization at all points in the calculation, and then as the very last preprocessing step, decrease the discretization. The motivation for this increased cost is because the second procedure has repeated gluings in mind; not only do we want a converged solution, but also the converged solution with minimal discretization. Because the convergence seems to be much more finicky with respect to changes in the spatial discretization when working at a fixed length $L = 22$.

A correction was made that more fairly weights solutions depending on their initial parameters; I was mistakenly creating initial conditions where solutions with drastically different periods were receiving equal number of points in the final discretization, as opposed to being weighted by how much they contribute to the final periods. In other words I wasn't incorporating the correct scales in the initial conditions.

In terms of specific testing, the main parameters that I tested were,

- Rediscretization before and after adjoint descent
- The size of the discretization of the buffers
- The method of creating the buffers
- The use of the "residual-guided" rediscretization

which lead to the following conclusions. It's always better to rediscretize before before numerical methods. A moderate buffer size is 5, a moderate number of points and likely should be proportional to the total number of points; still testing. Use of the residual guided rediscretization routine should always be used on the dimension perpendicular to the gluing dimension first. Some (seemingly) improved lower bounds on the number of points in the discretization is something like $M = 2^{(}\log_2 L + 1)$ and $N = 2^{(}\log_2 L - 1)$. Main point: less sensitive to changing (reducing) the maximum frequency mode (less points in time).

One aspect that must be taken into consideration whenever a linear system is being solved numerically is whether or not the results are accurate. One quantitative measure of this is the condition number of the corresponding matrix. The condition number is calculated by taking the ratio of the largest and smallest singular values computed by singular value decomposition. Broadly speaking, the condition number indicates how sensitive the system is to error or perturbation. If the condition number large the system is said to be "ill-conditioned" which in turn greatly affects the accuracy of solutions. For chaotic initial

value problems the magnitude of the condition number depends on the maximal Lyapunov exponent [60]. For our spatiotemporal problem, we do not have a quantitative measure of this but the numerical problem is almost always ill-conditioned. Therefore, accounting for this is very important if we want to trust our computations. This can be accomplished by applying what is known as *preconditioning*. Deciding on a specific *preconditioner* to use is a dark art, but some general guidelines are that the preconditioner should not be expensive to compute and it should approximately equal the inverse of the ill-conditioned linear operator.

This motivates preconditioning in the context of iterative methods and solving linear systems, but how does it apply to descent methods? The dimensionality of our problem is large, but that does not imply that all dimensions are equally important. To gain insight on this we will apply what we known about the Kuramoto-Sivashinsky equation as a dynamical system and then extend these ideas to our spatiotemporal formulation. The linear spatial derivative terms in the Kuramoto-Sivashinsky equation have direct effect on the spectrum of Fourier coefficients. Namely, the dissipation due to the fourth order term prevents shocks (discontinuities) and small wavelength oscillations. This ensures that the solutions are relatively smooth, the implication being that the spectra of Fourier coefficients converges as the wavenumbers increase. In other words, the magnitudes of the spatiotemporal Fourier modes vary greatly.

We demonstrate this with the following comparison. Let us consider two spatiotemporal Fourier modes, whose spatiotemporal wavenumbers k_1, k_2 and j_1, j_2 differ greatly, $k_1 << k_2$ and $j_1 << j_2$. Due to the higher order spatial derivatives the linear term of (**??**) is dominant when the wavenumbers become large. As a result, the spatiotemporal Fourier coefficients with large wavenumbers are approximately magnified by a factor of $|q_m^2 - q_m^4|\ m^4$. This magnification is large enough (technically its even larger due to the matrix-vector product) such that the components of the descent direction (**??**) are all of the same order of magnitude. This completely betrays the convergent behavior of the spectrum. Therefore in order to match the adjoint descent direction to our expectations we apply preconditioning. In this context, the preconditioning can be viewed geometrically; it rescales the components of the adjoint descent direction to better reflect the spectra of Fourier coefficients. As mentioned before, preconditioners should be cheap and approximate the inverse of whatever transformation is being applied. As previously argued the ill-conditioned nature of the spatiotemporal Kuramoto-Sivashinsky equation is the linear term. By applying a transformation which approximately equals the inverse of the linear term we can correctly scale the related components of both the adjoint descent direction and the solutions that result by iterative methods. Specifically let P denote the linear transformation defined by

$$P = \mathrm{diag}\Big(\frac{1}{|\omega_j| + q_k^2 + q_k^4}\Big)\,. \tag{108}$$

In addition we have found from experience that rescaling the magnitude of the changes in torus parameters (L, T) is helpful. Depending on the circumstances the following preconditioning may or may not be applied

$$\begin{aligned} P \cdot \delta T &= \frac{\delta T}{T} \\ P \cdot \delta L &= \frac{\delta L}{L^4}\,. \end{aligned} \tag{109}$$

The preconditioning given by (109) is mainly an attempt to imitate the effect of (108) for the parameters (L, T). The main motivation behind this rescaling is experiential in nature. A

common behavior for initial conditions which are very much non-solutions (cost functional has very large magnitude) and initial conditions on small spatiotemporal domains the partial derivatives with respect to (L, T) tend to be large in magnitude. Without rescaling this often leads to dramatic changes in spatial domain size or convergence to equilibrium solutions. One way of interpreting this is that the trivial manner that can reduce the magnitude of the cost functional is to increase the values of (L, T) when the linear term and nonlinear term are out of proportion.

By disabling changes to the period and length of the system this is lowered by an order of magnitude, however, I believe this still implies that the system is highly ill-conditioned even when trying to find fixed points of the spatiotemporal mapping on a fixed spatiotemporal domain. Xiong suggested that perhaps I can use the singular values to produce a preconditioner that might help, I'm looking into this as well as some other papers [46]

Our intent of this discussion is to very thoroughly describe what we are doing, in part because the notion of approximating the Jacobian via finite-differences is seemingly hardwired in certain disciplines. In order to do keep the derivations as concise and understandable as possible we elect to use a "direct matrix" notation [12]. That is to say, we rewrite (**??**) in terms of linear operators and pointwise multiplications

$$(\mathcal{D}_t + \mathcal{D}_{xx} + \mathcal{D}_{xxxx} + \frac{\sigma}{T}\mathcal{D}_x)\mathbf{u} + \frac{1}{2}\mathcal{D}_x\mathcal{F}(\mathcal{F}^{-1}\mathbf{u} * \mathcal{F}^{-1}\mathbf{u}) \tag{110}$$

where subscripts indicate differentiation. This notation will simplify the expressions derived for the sensitivity matrix and its adjoint. Namely, the columns of the sensitivity matrix

can be written as follows in the direct matrix notation

$$\begin{aligned}
\frac{\partial F}{\partial \mathbf{u}} &= (\mathcal{D}_t + \mathcal{D}_{xx} + \mathcal{D}_{xxxx} + \frac{\sigma}{T}\mathcal{D}_x) + \mathcal{D}_x\mathcal{F} \cdot \mathrm{Diag}(u) \cdot \mathcal{F}^{-1} \\
\frac{\partial F}{\partial L} &= (\frac{-2}{L}\mathcal{D}_{xx} - \frac{4}{L}\mathcal{D}_{xxxx} - \frac{\sigma}{TL}\mathcal{D}_x)\mathbf{u} - \frac{1}{2L}\mathcal{D}_x\mathcal{F}(\mathcal{F}^{-1}\mathbf{u}) * \mathcal{F}^{-1}\mathbf{u}) \\
\frac{\partial F}{\partial T} &= (-\frac{1}{T}\mathcal{D}_t - \frac{\sigma}{T^2}\mathcal{D}_x)\mathbf{u} \\
\frac{\partial F}{\partial \sigma} &= \frac{1}{T}\mathcal{D}_x\mathbf{u}\,.
\end{aligned} \tag{111}$$

The adjoint of the first equation of (111) is

$$\frac{\partial F}{\partial \mathbf{u}}^{\dagger} = -\mathcal{D}_t + \mathcal{D}_{xx} + \mathcal{D}_{xxxx} - \frac{\sigma}{T}\mathcal{D}_x - \mathcal{F} \cdot \mathrm{Diag}(u) \cdot \mathcal{F}^{-1}\mathcal{D}_x \tag{112}$$

The remaining adjoints (row vectors) are not shown because the matrix free computation is the same as (111) merely with an additional transposition. In summary, to compute all relevant matrix vector products we need matrix free versions of Fourier transforms, differentiation, elementwise multiplication. The only difficulty that arises are when *symmetry specific* Discrete Fourier Transform are desired. As implied by the name, these are Discrete Fourier Transform which remove redundant spatiotemporal Fourier coefficients which are constrained by symmetry. For a discussion of these constraints see sect. **??**.

The main goal is to be able to write the matrix-vector product in terms of function calls so that no matrices are explicitly formed. This would dramatically increase the speed at which my adjoint descent code runs and due to the global convergence of the method, and the speed when used in a hybrid method, this might be exactly the robustness that we need to find large doubly-periodic spacetime solutions at least.

More work into relative periodic orbit matrix-vector product formulation of the spatiotemp code. Still trying to figure out what I can use as extra constraints. For pre-periodic orbits I can still constrain to be transverse to time translations to give me corrections but because of the reflection symmetry I cannot impose tranversality to the spatial translation direction, which is what is used to complete the underdetermined linear system. Maybe I'm thinking about it the wrong way as usually these are just implemented as constraints to prevent the Newton correction from pointing in symmetry directions, and so I might be able to use the

I sort of went down a rabbit hole as to test what extra constraints are possible I tested a constraint the prevents changes in area. This was easy enough to implement, and it worked, but its the exact opposite of what I would like to accomplish. The tangent space(s) are what should be telling me the correct scales for L and T. Because of this I tried to impose a constraint such that the area is minimized but I can't currently think of a way that doesn't do this *too well*. What I mean by this is that if you say that you want to minimize $T * L$ then it will just send one of those parameters to 0, naturally.

One interesting thing that came about from the area constraint is that if you tell my code the wrong area. If for instance I changed L_0 to $L_0 - 2 = 20$ then it had to change the characteristic wavelength in time to fit in my box. Including figures for scientific curiousity.

The main idea I'm trying to flush out here is that if I cannot use tranversality constraints to complete the linear system then perhaps I need to complete the linear system by creating constraints on the parameters themselves.

Investigated this method in hopes that maybe I could prove Burak wrong by including "pseudo arc-length continuation" in the actual spatiotemporal code at the same, but sadly it doesn't look like one can have cake and eat it too.

Wrote the "reformulated" (rescaled) version of the mean velocity frame code for relative periodic orbit invariant 2-tori. As expected, it works a lot better with iterative methods such as GMRES.

I'm only able to get away with this reformulation because the diagonal terms corresponding to laplacian and laplacian squared operators dominate. For the mean velocity frame representation of the condition number is reduced to around $\kappa \approx 2000$ which is about an order of magnitude better than pre-periodic orbit invariant 2-tori.

It seems that these reformulations will be necessary as the matrix free code relies on approximations to iterative methods. When the iterative methods fail for the exact problem (explicitly formed matrices) I know I'm in danger. Luckily the test cases I've run through where, for instance, GMRES fails for the original equation but succeeds with the reformulated equations. Like I have mentioned before the reformulation is sort of a cheap trick I am using that could essentially be accomplished by preconditioning but I came up with some kooky ideas I want to try out to see if the reformulation of invariant 2-tori actually induces an iterative map or if I just think it does. Maybe I'm in fantasy land but because I am essentially rewriting (**??**), which can be drastically simplified as $F(u, T, L, \sigma) = 0$, as another equation $\tilde{F} - \mathbf{u} = 0$. I think the physical interpretation of the second equation is much clearer than the first, but it has its own issues as it demands for inversion of an operator that can technically become singular. The reformulated equations for the mean velocity frame

This was not the equation I put in practice as the inversion of the Laplacian and Laplacian squared term turned out to not be useful.

$$G(\mathbf{u}) = (D_{xx} - D_{xxxx})^{-1}((D_t + S)\mathbf{u} + D_x F((F^{-1}\mathbf{u})^2)) - \mathbf{u} = 0, \tag{113}$$

where S is the derivative of the Lie algebra element, $\dot{\phi}\cdot\mathbb{T} = 2\pi n\sigma/TL$, such that $2\pi\sigma/L = \theta$ is the parameter needed to perform SO(2) group action.

$$G(\mathbf{u}) = (D_t + S + D_{xx} - D_{xxxx})\mathbf{u} + D_x F((F^{-1}\mathbf{u})^2)) = 0, \tag{114}$$

where S is the derivative of the Lie algebra element, $\dot{\phi}\cdot\mathbb{T} = 2\pi n\sigma/TL$, such that $2\pi\sigma/L = \theta$ is the parameter needed to perform SO(2) group action.

Now that I am comfortable with saying the direct-matrix (forming matrices explicitly) methods will work for both Z_2 and SO(2) isotropy subgroup (relative periodic orbit in its mean velocity frame) solutions I have been working on the matrix-free methods. This was done in a poor order however because I should have made the changes to the automated invariant 2-torus finder to run on light while figuring this out.

Realized that perhaps adjoint descent would be better applied to full state space as opposed to spatiotemporal symmetry subspaces because its an integration(descent) type method. I wrote new codes that apply adjoint descent to the complex representation of the spatiotemporal Kuramoto-Sivashinsky equation, i.e. (**??**). The main benefit of doing so is that in the real-valued representation that I use for everything it isn't straight forward how to calculate the transpose of the Fourier transform and inverse transform operations, but when I use a complex representation the transformation is unitary; meaning that the conjugate transposes that arise are easily replaced using the unitarity property of the Fourier transform (with proper normalization).

Currently running tests on spatiotemporal domains initiated from random noise, large scale domains, and seeing what comes out. Random noise on a $T, L = 500, 500$ domain ran long enough starts to pick out the length scales of the Kuramoto-Sivashinsky equation, and its the matrix-free computation that allows it to run fast enough to do anything.

Figure **??** is an example of the limits of the method so far. Taking (pseudo) white noise on a large space-time domain and then running the adjoint descent reproduces structures that look similar to those seen within simulations of the Kuramoto-Sivashinsky equation.

Another topic I need to breach is whether I can constrain the adjoint descent to preserve the symmetry subgroup of any solutions found. If I can get the matrix-free method to work in the real representations this will automatically hold but not so in the complex representation as I'm abusing the extra variables and unitarity of the Fourier transform so that everything is unconstrained.

The finite difference calculation that is used to approximate the matrix vector product is just too inaccurate to be of use for the spatiotemporal problem. Comparing the norms of the two vectors, the exact product, $|J\delta\mathbf{x}|$, and its approximation, $|\frac{F(x+\delta\mathbf{x})-F(x)}{|\delta x|}|$ they're off by almost exactly $|\delta x|$, which makes me feel like I had made some stupid mistake but changing this in the GMRES code makes it completely worthless.

While these equations are the same for both the IVP and BVP, there are implicit differences in what u and v represent. For the IVP, $u(x,t)$ and $v(x,t)$ represent a single instant in time, while for the BVP they represent an entire spatiotemporal area.

Therefore, for the IVP the adjoint equation is another dynamical equation which is arguably more complicated than the original Kuramoto-Sivashinsky equation because it relies on two field variables instead of one. In the context of the BVP, however, it represents an additional differential algebraic equation defined on the same domain as $u(x,t)$. This intrinsic difference is the reason for the drastic difference in what the "adjoint operator" entails. Specifically, The adjoint operator for the IVP is the stability matrix [57] of this adjoint equation, that is, the matrix that results from applying the gradient operator ∇_v.

It represents the instantaneous rate of change in a linear neighborhood of the co-state space point v. Similar to the stability matrix of the Kuramoto-Sivashinsky equation, it is a time-dependent quantity that depends only on the current point in state space, $u(x,t)$ as it is linear in v. This time dependent nature is captured by the time dependent Jacobian operator J^t and its adjoint $(J^\top)^(t)$. These operators map the corresponding linear neighborhoods forward in time and evolve according to the differential equation

$$\dot{J} = AJ \tag{115}$$

where A is the stability matrix and J is the Jacobian, $J^(0) \equiv \mathbb{I}$. To simplify the notation, we have dropped the implicit time dependence as well as the dependence on $u(x,t)$. To solve this equation, that is, to solve for a finite time Jacobian matrix, one must numerically simultaneously integrate the augmented system of equations comprised of the original dynamical equation and (115). By taking a recurrence from ergodic trajectories using a crude Poincaré section formulation, and then using convolution with a Gaussian mollifier yields bad spatiotemporal results. Due to the discrepancy between the few steps of convergence it takes for known orbits and the complete lack of convergence for the few new orbits tried it seems to indicate that either the initial Newton residual for convergence has to be smaller than what I am starting with. 10^{-3} seems to be the maximum for the initial residual value.

Began parsing through the literature on (variational) newton descent, specifically refs. [13, 35] and for invariant tori ref. [36].

$$\frac{\partial x}{\partial \tau}(s+\omega,\tau) + \frac{\partial x}{\partial s}(s+\omega,\tau)\frac{\partial \omega}{\partial \tau}(\tau) - J(x(s,\tau))\frac{\partial x}{\partial \tau}(s,\tau) = f(x(s,\tau)) - x(s+\omega,\tau), \tag{116}$$

Still trying to find a systematic way of producing initial conditions for the use of variational newton descent. The main equation governing the fictitious time evolution is again (**??**) or the non-square matrix variant, (**??**). The only differing components from that of the Rössler system, or spatial Kuramoto-Sivashinsky is the definition of the velocity field v and therefore the definition of the stability matrices A.

Began parsing through the literature on (variational) newton descent, specifically refs. [13, 35] and for invariant tori ref. [36].

$$\frac{\partial x}{\partial \tau}(s+\omega,\tau) + \frac{\partial x}{\partial s}(s+\omega,\tau)\frac{\partial \omega}{\partial \tau}(\tau) - J(x(s,\tau))\frac{\partial x}{\partial \tau}(s,\tau) = f(x(s,\tau)) - x(s+\omega,\tau), \tag{117}$$

Still trying to find a systematic way of producing initial conditions for the use of variational newton descent. The main equation governing the fictitious time evolution is again (**??**) or the non-square matrix variant, (**??**). The only differing components from that of the Rössler system, or spatial Kuramoto-Sivashinsky is the definition of the velocity field v and therefore the definition of the stability matrices A.

There is a storm in the distance however, as this general procedure is ruined for the spatial problem. As we know from the chronotopic literature refs. [18, 37, 38, 55], that iteration in space typically does not converge to the same attractor as iteration in time, and generally corresponds to a strange repeller. Therefore I cannot hope to form an initial guess loop from using a Poincaré section in the spatial direction, as typically all of my Fourier coefficients go off to infinity before a recurrence is found.

My idea to remedy this is to actually use *time* integration to form a initial guess loop for applying newton descent in *space*. If I integrate a spatially periodic initial condition in time, by virtue of the spatial periodicity there is a close recurrence in the spatial direction (close

and not exact only due to discretization I believe). If I've thought about this the right way. It's the smartest way I can think of to generate an initial condition for the spatial newton descent (**??**) given that my spatial integration code is ill-behaved. If my *spatial* code was working and there is no lapse in my rationale then it might actually have been a way to produce smooth initial guess loops for the *time* direction newton descent code.

For any changes that I make that translate generically to any system I try to test them with Rössler first before applying them to antisymmetric subspace $\mathbb{U}^+$ of Kuramoto-Sivashinsky. The other changes that are unique to antisymmetric Kuramoto-Sivashinsky must of course be tested in that realm. The changes are looking promising, as I can find longer periodic orbits in the Rössler system now, however the period seems to be slightly off (Integrating the solution after application of newton descent yields a solution that is periodic but overlaps). This is my number one suspect for why the newton descent code continues to stall at the moment; however, for the Rössler system although it is much slower for longer periodic orbits, it still converges very quickly once $F^2 < 1$.

I realized that I should not have to worry about implementing a slicing condition in the spatial version of variational newton descent; All that was required in the time case was to reduce the symmetry associated with the marginal direction parallel to velocity, i.e. a Poincaré section. I didn't worry about the spatial translational invariance and it was able to converge to a solution just fine in the time case. There also might be a smarter way of choosing a constraint that enables better convergence, as opposed to the "first coordinate" hyperplane (i.e. the first Fourier mode in most systems). I'm currently playing around with using a hyperplane condition on the "more dynamical" variables which is a hasty and crude name not to be taken seriously. What I mean by this is that in (**??**) the spatial derivatives of the Fourier coefficients of $u^{(3)}$, which represents the third derivative, are much more complex than the other derivatives, so perhaps using a hyperplane condition on one of these coordinates would be better; this hasn't seemed to be the case yet.

I thought that I would have to somehow permute the elements (145) of the "Loop Vector" (vector that encodes the parameterization of initial condition for periodic orbit search). The reasoning behind this was in order to use differentiate with respect to a parameterization variable s, I would need the elements to be in sequential order with respect in parameterization variable s, in order to multiply by vector $i\vec{m}$, where m is the conjugate variable (in a Fourier transform sense) to s. This is ***not*** the case, as I can merely exploit the Kronecker outer product to produce a diagonal matrix such that along the diagonal there are M duplicates of each element of $\vec{m}$

We are essentially diagonalizing a sparse matrix for $\mathcal{O}(M\,(nlog(n)))$ flops from taking M Fourier transforms of length n = power of 2. This is all well and good, but I think that there might be complications from the stability matrices; I need to go through the calculation, but the naive way to write the stability matrices in their new representation is: $\tilde{\mathbf{A}} = \mathbf{FAF}^*$, where F is a unitary matrix representing the discrete Fourier transform.

Next is the representation of the fictitious time evolution as a system of linear equations, similar to (**??**), which is restated here for comparison to the new system of equations.

The old linear system is given by,

$$\begin{bmatrix} M & -v \end{bmatrix} \begin{bmatrix} \delta\tilde{x} \\ \delta\lambda \end{bmatrix} = \delta\tau \begin{bmatrix} \lambda v - \tilde{v} \end{bmatrix}, \tag{118}$$

where $M = D - \lambda Diag(A_n)$ with D being the finite difference matrix, and A_n a block diagonal matrix containing stability matrices.

Now, the equations the same form, with new variables described by over-bars

$$\begin{bmatrix} \bar{M} & -\bar{v} \end{bmatrix} \begin{bmatrix} \delta\tilde{\bar{x}} \\ \delta\bar{\lambda} \end{bmatrix} = \delta\tau \begin{bmatrix} \lambda\hat{\bar{v}} - \tilde{\bar{v}} \end{bmatrix}, \tag{119}$$

where $\bar{M} = \mathbf{F}Diag(i\vec{m})\mathbf{F}^* \otimes \mathbf{I}_d - \lambda Diag(A_n)$ and $\bar{v} = \mathcal{F}(v)$, $\tilde{\bar{v}} = (Diag(i\vec{m}) * \tilde{\bar{x}}$. [1]

After talking to Ashley, who told me to start the multishooting effort with only a few number of points rather than the large discretization used as if it was a newton descent, I looked back into the variational multishooting technique that he described back in Santa Barbara. I took four point on the original orbit, while my code is minimizing the cost functional (**??**) I am yet again getting the "equilibrium descent" for an antisymmetric initial condition $\in \mathbb{U}^+$ that converges with my variational newton descent code. This resulting equilibrium "solution" is a typical result when something is ill-defined. I would speculate that the manner in which I am handling the adjoint equations is the culprit, as I tried to modify the ETDRK4 of ref. [27] to be the numerical integration routine to integrate the equations.

Fixed memory issue by making it such that the "Newton descent matrix" i.e. the matrix in is not evaluated before each least squares evaluation; rather, we keep this matrix constant as an approximation and then when the cost functional can no longer decrease, i.e. we have left the local neighborhood of the stability matrices that define the matrix, we redefine the matrix and then restart the search; this is similar to what is implemented in other variational Newton descent code; forgot this fact when I rewrote the spatial Newton descent code to use LSQR to solve the least squares problem as opposed to using matrix inversion. Application of the spatial Newton descent code to ergodic trajecto- ries that have been deformed to be periodic in time were resulting in

Debugging and changing spatial newton descent code. I think I've pinned down the main problem to be an error in trying to use Fourier transforms rather than explicit sums in my definition for the stability matrix elements, or initial condition generation. I'm currently using the shortest periodic orbit in time as an initial guess; Therefore, I would expect the value of the cost functional to be small relative to something that isn't already doubly-periodic in time and space. While the value of the cost functional is relatively small, it still may be too large for an initial guess. That being said I can at least list what I believe to be the most likely cause of errors.

The reason for this is such: Spectral differentiation with respect to a parameterization variable, s, can be rewritten as multiplication by a diagonal matrix with elements is, but this requires the vector that is representing the entire loop to be ordered in a very specific way. If the loop was ordered in this specific way it would reduce to multiplication of a large diagonal matrix which would be repeating the (small) diagonal matrices with elements is. If one wants to do it this way I believe the easiest way, in order to avoid reordering the stability matrix elements, would be to formulate it this way mathematically: The matrix $D_{\mathcal{F}}$, which produces the approximate tangent space after multiplication with the "loop vector" $\mathbf{x}$, could be represented in such a way,

$$D_{\mathcal{F}} = \mathbf{P}^{-1}\mathbf{F}^{-1}\mathbf{Diag}(is)\mathbf{FP} \quad \text{where,}$$

$\mathbf{F}$ = Block diagonal matrix composed of Fourier transform matrices (i.e. to Fourier transform each of the Fourier coefficient series with respect to parameterization variable),

[1] Matt: I changed this such that the only difference between my current code and this formulation is the calculation of approximate tangent space via Fourier methods.

$\mathbf{P}$ = Permutation matrix to reorder in specific way to enable easy Fourier transforms.

Another main challenge is how to implement a slice condition to deal with translational invariance. Typically this is dealt with when the spatial Fourier series is being used, and therefore it is easier to represent a hypersurface that eliminates this marginal direction; in the spatial newton descent code (this is what I call using (**??**) with variational newton descent) I am trying to eliminate the translational freedom but it's not as straightforward as the first Fourier mode slice; as the first Fourier mode slice in this case would eliminate time translations. I've been looking towards some of the papers about invariant tori and their "phase conditions" as a possible means of escape.

I realized that I should not have to worry about implementing a slicing condition in the spatial version of variational newton descent; All that was required in the time case was to reduce the symmetry associated with the marginal direction parallel to velocity, i.e. a Poincaré section. I didn't worry about the spatial translational invariance and it was able to converge to a solution just fine in the time case.

I thought that I would have to somehow permute the elements (145) of the "Loop Vector" (vector that encodes the parameterization of initial condition for periodic orbit search). The reasoning behind this was in order to use differentiate with respect to a parameterization variable s, I would need the elements to be in sequential order with respect in parameterization variable s, in order to multiply by vector $i\vec{m}$, where m is the conjugate variable (in a Fourier transform sense) to s. This is ***not*** the case, as I can merely exploit the Kronecker outer product to produce a diagonal matrix such that along the diagonal there are M duplicates of each element of $\vec{m}$

We are essentially diagonalizing a sparse matrix for $\mathcal{O}(M\,(nlog(n)))$ flops from taking M Fourier transforms of length n = power of 2. This is all well and good, but I think that there might be complications from the stability matrices; I need to go through the calculation, but the naive way to write the stability matrices in their new representation is: $\tilde{\mathbf{A}} = \mathbf{FAF}^*$, where F is a unitary matrix representing the discrete Fourier transform.

Next is the representation of the fictitious time evolution as a system of linear equations, similar to (**??**), which is restated here for comparison to the new system of equations.

The old linear system is given by,

$$\begin{bmatrix} M & -v \end{bmatrix} \begin{bmatrix} \delta\tilde{x} \\ \delta\lambda \end{bmatrix} = \delta\tau \begin{bmatrix} \lambda v - \tilde{v} \end{bmatrix}, \tag{120}$$

where $M = D - \lambda Diag(A_n)$ with D being the finite difference matrix, and A_n a block diagonal matrix containing stability matrices.

Now, the equations the same form, with new variables described by over-bars

$$\begin{bmatrix} \bar{M} & -\bar{v} \end{bmatrix} \begin{bmatrix} \delta\tilde{\tilde{x}} \\ \delta\bar{\lambda} \end{bmatrix} = \delta\tau \begin{bmatrix} \lambda\hat{\tilde{v}} - \tilde{\tilde{v}} \end{bmatrix}, \tag{121}$$

where $\bar{M} = \mathbf{F}Diag(i\vec{m})\mathbf{F}^* \otimes \mathbf{I}_d - \lambda Diag(A_n)$ and $\bar{v} = \mathcal{F}(v)$, $\tilde{\tilde{v}} = (Diag(i\vec{m}) * \tilde{\tilde{x}}$. [2]

The best results, (i.e. better than square matrix problem, but still not good enough) was with SciPy's LSQR algorithm, which, in the paper that it is based on ref. [52], describes it as a "conjugate-gradient-like" algorithm, with better stability. I haven't gotten into the nitty gritty as of yet.

[2] Matt: I changed this such that the only difference between my current code and this formulation is the calculation of approximate tangent space via Fourier methods.

While waiting for Arnoldi iteration to finish so that I could begin testing the spatial variational newton descent without fear of memory problems I was trying to think about the best way to use (**??**), which I will restate here:

$$\left[i\omega_\ell - (q_k^2 - q_k^4)\right] \mathbf{u}_{k,\ell} + i\frac{q_k}{2}\sum_{k'=0}^{N-1}\sum_{m'=0}^{M-1}\mathbf{u}_{k',m'}\mathbf{u}_{k-k',m-m'} = 0\,. \tag{122}$$

First, $\mathbf{u}_{k,\ell}$ represents matrix elements, so it makes sense to rewrite the equation as a matrix equation. Define matrices $\mathbf{Q_1} \equiv Diag(-q_k^2 + q_k^4)$, $\mathbf{W} \equiv Diag(i\omega_\ell)$, $\mathbf{Q_2} \equiv Diag(\frac{iq_k}{2})$, and let the two dimensional FFT be represented by matrix multiplication $\mathbf{U} = \mathbf{F}_M\mathbf{u}\mathbf{F}_N$, where the matrix elements $U_{k,\ell} = \mathbf{u}_{k,\ell}$.

With these definitions the equation can be rewritten as:

$$\mathbf{Q_1 U W} + \mathbf{Q_2 F_M}(\mathbf{u}\otimes\mathbf{u})\mathbf{F_N} = 0 \tag{123}$$

where the nonlinear term is calculated in configuration space as to avoid the two dimensional convolution.

Because $\mathbf{Q_1}$ and $\mathbf{W}$ are diagonal, their inverses are easily found, and the equation (123) can be rewritten

$$\mathbf{Q_1}^{-1}\mathbf{Q_2 F_M}(\mathbf{u}\otimes\mathbf{u})\mathbf{F_N}\mathbf{W}^{-1} + \mathbf{U} = 0 \tag{124}$$

Now we can redefine $\mathbf{U} \to -\mathbf{U}$, and remember to convert back after finding the fixed point.

Define

$$f_{k,\ell}(\mathbf{u}_{k,\ell}) \equiv \mathbf{Q_1}^{-1}\mathbf{Q_2 F_M}(\mathbf{u}\otimes\mathbf{u})\mathbf{F_N}\mathbf{W}^{-1} \tag{125}$$

and therefore we have an equation of the form: $f_{k,\ell}(\mathbf{u}_{k,\ell}) - \mathbf{u}_{k,\ell} = 0$, where the Jacobian matrix is given by the fourth rank tensor that arises from taking partial derivatives with respect to $\mathbf{u}_{k,\ell}$ More to be derived in the future, hoping to make headway into finding tori; I can't tell if this equation is going to be useful or if I should really be working towards deriving and learning variational newton descent equations for finding tori similar to Lan, Chandre and Cvitanović [36].

Application of the spatial newton descent code to ergodic trajectories that have been deformed to be periodic in time were resulting in the "falling into equilibrium" problem, this was due to a bug where the wrong temporal system sizes were being used.

Application of spatial newton descent on pre-periodic orbit10.2 results in a reduced cost functional but seems rather obstinate in regards to convergence. Luckily, the approximate loop seems to fluctuate around spatial extent $L = 22$. I think this is a good indication as it means the spatial newton descent is capturing the spatial geometry of pre-periodic orbit10.2. That is to say, even while reducing the cost functional the solution doesn't want to betray itself, as it originates from the spatial system size $L = 22$.

I tried whether it be the error tolerances, step sizes (variable or constant), initial condition discretizations, least squares solvers, pseudoinverse or regular inverse methods, hypersurface constraints, matrix preconditioners, etc, did not help the converge properties. By examining the corrections being applied to deform the loop, specifically the maximum correction applied in each step it seems most of the steps are modifying the "period" i.e. the spatial extent of the initial condition the most. There might be some way to discourage this with an additional condition on the rescaling factor λ that matches the magnitudes of the approximate tangent space to the actual tangent space. The majority of modifications

being put into changing this rescaling factor seem to be the cause of the critical slow down of the algorithms, which might be indication of the presence of a continuous symmetry that needs to be dealt with that isn't currently being dealt with.

Still haven't been able to get this to work, after some thought over the weekend I have been trying to implement a major change to the code. The general idea is this, the first three equations of (**??**) will by definition match the approximate loop tangents as they are generated via spectral differentiation, which is now how I am computing the approximate loop tangents. I have been trying to work out how this can be exploited as to greatly reduce the dimensionality of the system. I.e. instead of keeping track of the real and imaginary components of the Fourier coefficients of u, u_x, u_{xx}, u_{xxx}, I should be able to only keep track of u, and then match the last equation of (**??**) to the fourth derivative of u computed by spectral differentiation. The main problem with this formulation is that I haven't been able to rewrite figure out the best way to rewrite the stability matrix elements, other than they should only depend on the real and imaginary components of the temporal Fourier modes of u.

The key idea is that because the initial conditions for the spatial system, u, u_x, u_{xx}, u_{xxx} were all being generated through spectral differentiation, it made matching the tangent spaces redundant in three variables, as the approximate tangent spaces were being generated with spectral differentiation as well. Therefore, the idea is to turn only the last equation in (**??**) into a "direct-matrix" [12] equation similar to the spatiotemporal mapping, and match it to the velocity field's fourth derivative. By proceeding in this manner, the equation for the fourth derivative, (which I will refer to from here on as *the* tangent space) takes the following "direct-matrix" [12] form. Note, that I am attempting to devise an equation that is only dependent on the Fourier coefficient of the velocity field and not any of the spatial derivatives. This is acceptable because the spatial derivatives are derived from the original velocity field anyway. What appears now in the equations are linear operators that produce the derivatives accordingly. Therefore, $\mathbf{u}$ will refer to $\mathbf{u}^0$ in accordance with notation previously used. Finally, the velocity equation (fourth spatial derivative) now appears as follows,

The key idea is that because the initial conditions for the spatial system, u, u_x, u_{xx}, u_{xxx} were all being generated through spectral differentiation, it made matching the tangent spaces redundant in three variables, as the approximate tangent spaces were being generated with spectral differentiation as well. Therefore, the idea is to turn only the last equation in (**??**) into a "direct-matrix" [12] equation similar to the spatiotemporal mapping, and match it to the velocity field's fourth derivative. By proceeding in this manner, the equation for the fourth derivative, (which I will refer to from here on as *the* tangent space) takes the following "direct-matrix" [12] form. Note, that I am attempting to devise an equation that is only dependent on the Fourier coefficient of the velocity field and not any of the spatial derivatives. This is acceptable because the spatial derivatives are derived from the original velocity field anyway. What appears now in the equations are linear operators that produce the derivatives accordingly. Therefore, $\mathbf{u}$ will refer to $\mathbf{u}^0$ in accordance with notation previously used. Finally, the velocity equation (fourth spatial derivative) now appears as follows,

$$v = -W\dot{\mathbf{u}} - Q_2\dot{\mathbf{u}} - F\dot{(}(F^{-1}\dot{\mathbf{u}}) \star ((F^{-1}\dot{Q}1\mathbf{u})), \tag{126}$$

Using the direct-matrix differentiation rules noted above (84), the stability matrix takes on the following form,

$$A = -W - Q_2 - F\dot{(}diag(F^{-1}\dot{Q}1\dot{\mathbf{u}})\dot{F}^{-1} + diag(F^{-1}\dot{\mathbf{u}})\dot{F}^{-1}\dot{Q}1) \tag{127}$$

For a quick description of the operators, W is the operator that produces the time derivative of a given field $\mathbf{u}$, Q_2 produces the second spatial derivative, F performs a forward FFT of a time-series, Q_1 produces the first spatial derivative. In this notation, the approximate tangent space would be the fourth spatial derivative as produced by spectral differentiation, i.e. $\tilde{v} = Q_4\dot{\mathbf{u}}$. Everything else from the variational newton descent is left untouched.

I'm hoping that this will enable convergence of the spatial system of equations to find periodic orbits in space, the main motivation for performing these changes were firstly, the code wasn't working probably due to some inaccuracies or errors, secondly, by keeping everything defined in terms of only the original velocity field $u(x,t)$ I dramatically reduce the memory requirements and degrees of freedom of the system.

Spent the day debugging the spatial newton descent code, found a negative sign error in the expression for matrices governing Fourier transforms, converted from taking real(cosine and sine) fft's in one variable (time) to complex fft's as I was having some trouble reproducing the time derivative term otherwise. Took a while to figure out what was wrong until I looked at matrix-product results piece by piece and compared to expected results in MATLAB. Rewrote how matrices (derivative operators) are formulated in terms of the Fourier transform operators. Also found a peculiarity when it came to the accuracy of matrix multiplcation depending on the order in which the matrices were multiplied...haven't figured that one out yet but nonetheless I corrected it.

I believe I got it working finally, however, the results so far aren't as interesting as I had dreamed. I finished this at the end of the day so I didn't get to test it too much, but so far there are two resulting possibilities. First, with a time periodic initial condition, i.e. one of the periodic orbits in time of Kuramoto-Sivashinsky, when allowing for spatial domain changes and changes to the temporal Fourier coefficients, the solution (only tested one so far) converged to one of the temporal equilibria of the Kuramoto-Sivashinsky system. This I believe is an indication that my code is indeed working, even though this was usually a sign of numerical issues when searching for periodic orbits in time using the variational methods, (i.e. the only way to reduce the cost functional $\mathcal{F}^2 = \lambda v - \tilde{v}$ is to send v and $\tilde{v}$ to 0. The reason I believe this is still valid is because the spatial derivatives of the equilibrium state found are nonzero; i.e. one of the "spatial periodic orbits" I have found is indeed the temporal equilibrium of the system.

When using a coarse discretization of 16-by-16 space-by-time points the spatial domain size that the solution settled to was $L = 19.9324743429$, with the value of cost functional being within machine precision of 0. When the spatial discretization was doubled, i.e. a 32-by-16 space-by-time grid, the resulting domain size was 23.7360639824. Likewise, when using a 16-by-32 space-by-time discretization, the resulting domain size was $L = 19.9398768032$, which indicates that the domain size of the converged solution is highly dependent on the (spatial)discretization being using.

The second possibility is a convergence to a zero domain size solution, which in my variational method for time usually indicated an equilibrium of the system. I.e. the two possibilities were either an equilibrium in time (but still a periodic orbit in space), or a spatial equilibrium.

After more investigation it turns out that the spatial variational code is indeed not working yet. Tried to put resulting orbits into time integrator in order to reproduce the

result and got an unmatching solution. figure 7 is my newest "results". It uses a $\overline{rpo}_{16.31}$ of the Kuramoto-Sivashinsky equation as the initial condition. I think I was deceived by how nice it looks I suppose...couldn't find any errors today that could enable reproduction via time integration.

As an additional test I put the solutions into Burak's symmetry reduced time integrator to verify whether the "solution" in figure 7 was a relative periodic orbit but alas there was no luck; there is some other error that I haven't been able to identify as of yet. Found another negative sign error, updating figure **??**.

Applied changes based on implementation from spatial variational method to my torus finding code but it seems I jumped the gun as I cannot reproduce any orbits found by spatial newton descent via time integration.

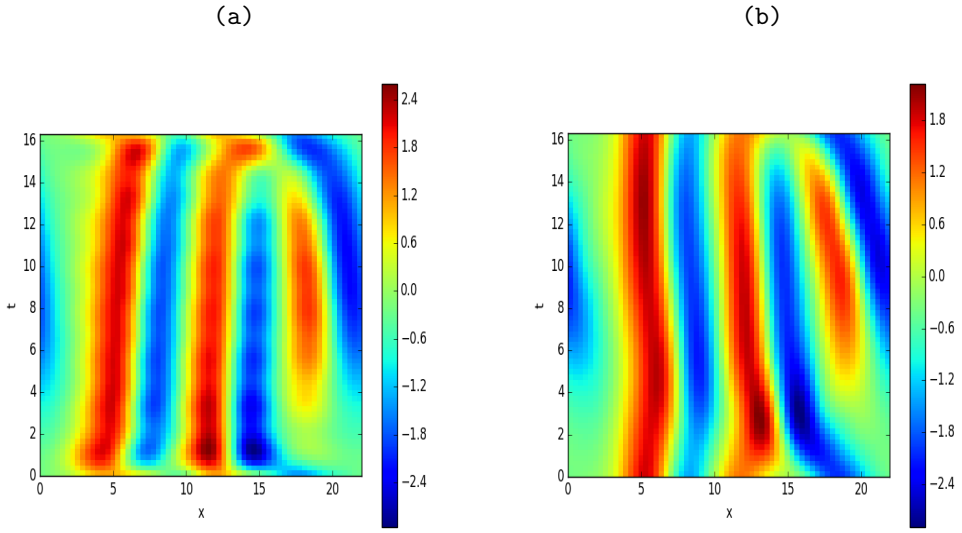

Figure 1: (a) Initial condition of the 16-by-16 space-by-time discretization of $\overline{rpo}_{16.31}$ ($L = 22$) for spatial variational newton descent of the Kuramoto-Sivashinsky equation (b)Resulting "spatial periodic orbit" (temporal equilibrium), with final spatial extent of $L = 21.9394614064$

updated figure 7, figure **??**, and uploaded figure 8 to display current results.

After trying numerous small modifications to the variational method's system of equations from ref. [34] including implementing factors involving the coordinate involved in defining the in-slice time and other quantities involving the slice tangents I didn't get any improvements over what I already had. I still think the problem has to do with the slice time so I went over the derivation of the equations and I believe I found a correction that can be made.

The derivation of the system of equations used to find the fictitious time corrections to the initial guess loop involves substitution of the partial derivative of the rescaling factor with respect to fictitious time. The way that this is done is via the definition

$$\lambda_n = \frac{\Delta t_n}{\Delta s_n}\,, \tag{128}$$

where, s is the parameterization variable, here defined with a subscript n to indicate that the parameterization need not be uniform around the initial guess loop (although this is

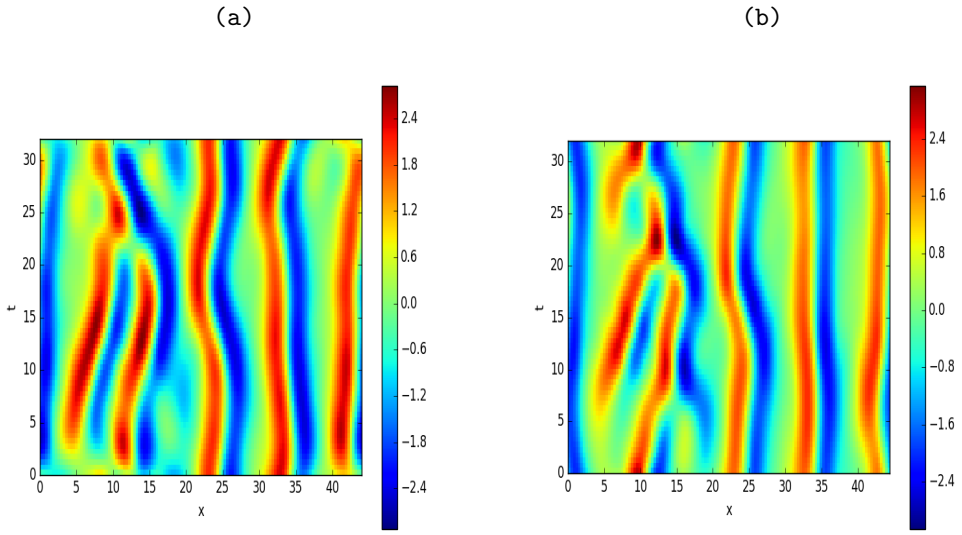

Figure 2: (a) Initial condition of the 32-by-16 space-by-time discretization of a piece of an ergodic trajectory that has been deformed to be periodic in time. $L = 44$. (b) Resulting spatiotemporal periodic orbit, with final spatial extent of $L = 44.3937151766$.

what I work worth as it makes things much easier to program). t in this instance stands for the real dynamical time of the orbit.

If we substitute the definition for the in-slice time, this equation takes the form

$$\lambda_n = \frac{\Delta\hat{(t)}_n (x_1)_n}{\Delta s_n} \tag{129}$$

Going through the almost identical derivation for the system of equations there is one place where I believe they differ. Normally, there is the substitution

$$\delta t_n = \frac{\partial \lambda_n}{\partial \tau} \Delta s_n \delta\tau \tag{130}$$

Normally this is easily generalized in order to produce a uniform rescaling of the period around the orbit, but if the coordinate x_1 is involved I believe that this should take the form

$$\delta t_n = (\frac{\partial \lambda_n}{\partial \tau}(x_1)_n + \frac{\partial (x_1)_n}{\partial \tau}\lambda_n)\Delta s_n \delta\tau \tag{131}$$

As the in-slice time rescaling is coordinate dependent, this formula seems to beg for a description of the general equation that has a coordinate dependent rescaling. This is kind of what I have been stuck on as this is quite different from what I am used to.

For a orbit that is described by M points in time we would need λ_i where $i = 0, ..., M-1$. I'm currently working on implementing this into my current code but it's gotten the best of me so far.

The way I am attempting to solve the problem I am having with in-slice time is as such, for M discretized points representing an in-slice time description of a relative periodic orbit I am introducing M different time rescaling quantities λ_m. In this way the original

variational method equation changes from

$$\left[D - \lambda diag(A_0, ..., A_{M-1}), \quad -v\right] \left[\begin{array}{c} \delta\tilde{x} \\ \delta\lambda \end{array} \right] = \delta\tau \left[\begin{array}{c} \lambda v - \tilde{v} \end{array} \right], \tag{132}$$

to

$$\begin{aligned} &\left[D - diag(\lambda_0 * A_0, ..., \lambda_{M-1} * A_{M-1}), \quad -diag(v_0, ..., v_{M-1})\right] \left[\begin{array}{c} \delta\tilde{x} \\ \delta\lambda_m \end{array} \right] \\ &= \delta\tau \left[\begin{array}{c} diag(\lambda_m)v - \tilde{v} \end{array} \right], \end{aligned} \tag{133}$$

In this way the correction mentioned in (131) is not being resolved but I want to try to see if more flexibility in the rescaling of in-slice time with respect to fictitious time evolution is sufficient before getting into an equation I derived; it's also a simpler step whose changes could be carried over to the full in-slice time description I probably will need.

Wrote some new code for symmetry reduced spatiotemporal fixed point finding; currently it seems a little strange to me because I only know how to quotient the spatial translation symmetry in spatial Fourier space as opposed to spatiotemporal (double) Fourier space.

The way that this I am attempting this, are with the equations, in direct-matrix notation:

Using the definition of the reduced velocity in ti me, (**??**), restated here for reference,

$$\hat{v} = (X \cdot \mathbf{u}) * v - (\dot{Y} \cdot v) * (I_M \otimes T) \cdot \mathbf{u}, \tag{134}$$

We then find the equivalent spatiotemporal system by Fourier transforming in time by means of application of F_t linear operator denoting the transform and moving everything to one side. Here, the $\mathbf{u}$ stands for a description of the initial solution that is only Fourier in space.

$$W \cdot F_t \cdot \mathbf{u} - F_t \cdot ((X \cdot \mathbf{u}) * v - (\dot{Y} \cdot v) * (I_M \otimes T) \cdot \mathbf{u}) = 0 \tag{135}$$

The stability matrix resulting from this symmetry reduced mapping is therefore,

where $\hat{A}$ is in reference to the equation previously defined (**??**).

$$J = W \cdot F_t - F_t \cdot \hat{A} \tag{136}$$

The part that troubled me for a while was how to implement symmetry reduction of the spatial translation symmetry in a spatiotemporal way as opposed to what I have now. I haven't been able to come up with anything good yet but it just seems like a little bit of a hack job to reduce the symmetry before taking things to the spatiotemporal stage. Because I am trying to find fixed points after symmetry reduction I imagined that this case should be exactly be like symmetry reduction of a relative equilibria but I haven't found a good way of treating the equations yet.

As a means of cross-checking results I wrote variational newton descent code for time, debugged, and got it working. It follows the "direct-matrix" approach that I have been finding useful. The velocity equation takes the form

$$v = Q_1 \cdot \mathbf{u} - Q_2 \cdot F \cdot ((F^{-1}\mathbf{u}) \star (F^{-1}\mathbf{u})) \tag{137}$$

and the stability matrix is therefore

$$A = Q_1 - 2 * Q_2 * \cdot F \cdot diag(F^{-1}\mathbf{u}) \cdot F^{-1} \tag{138}$$

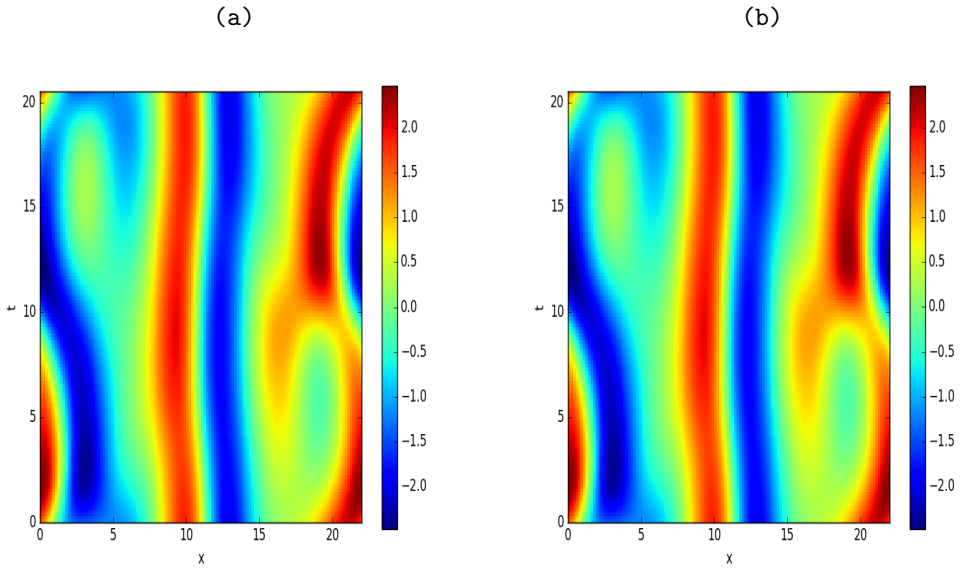

Figure 3: (a) Initial condition of the 32-by-32 space-by-time discretization of pre-periodic orbit10.2: $(L_0, T_0) = (L_0, 2T_{p_0}) = (22, 20.5057459345)$. (b) Resulting spatiotemporal fixed point $(L_p, 2T_p) = (22.0000104401, 20.5057499188)$

figure 6 is a comparison between resulting orbits from both space and time variational newton descent. The general structure of the resulting orbit is preserved even though the periods are quite different. At first I believed that these were supporting evidence that something is going right but now I am confused. The spatial domain resulting from the *spatial* newton descent is largely unchanged, meaning that the orbit that results should be a unique solution with a unique period; but the *time* variational newton descent changes the period somewhat drastically, and because the spatial domain size is fixed this might mean that I am finding the same solution but this is contradictory because the periods are so different. Also good evidence that there is something wrong is that the *time* newton descent is taking a relative periodic orbit initial condition and seemingly changing it into a periodic orbit? I don't know what to think of it but I am glad that I rewrote the time newton descent code as it'll be a good stepping stone into whether or not I need to rewrite the equations in a symmetry reduced form in order to find relative periodic orbits or not. I think that's where I'm headed at least.

Began parsing through the literature on (variational) newton descent, specifically refs. [13, 35] and for invariant tori ref. [36].

$$\frac{\partial x}{\partial \tau}(s+\omega,\tau) + \frac{\partial x}{\partial s}(s+\omega,\tau)\frac{\partial \omega}{\partial \tau}(\tau) - J(x(s,\tau))\frac{\partial x}{\partial \tau}(s,\tau) = f(x(s,\tau)) - x(s+\omega,\tau), \quad (139)$$

There is a storm in the distance however, as this general procedure is ruined for the spatial problem. As we know from the chronotopic literature refs. [18, 37, 38, 55], that iteration in space typically does not converge to the same attractor as iteration in time, and generally corresponds to a strange repeller. Therefore I cannot hope to form an initial guess loop from using a Poincaré section in the spatial direction, as typically all of my Fourier coefficients go off to infinity before a recurrence is found.

My idea to remedy this is to actually use *time* integration to form a initial guess loop for

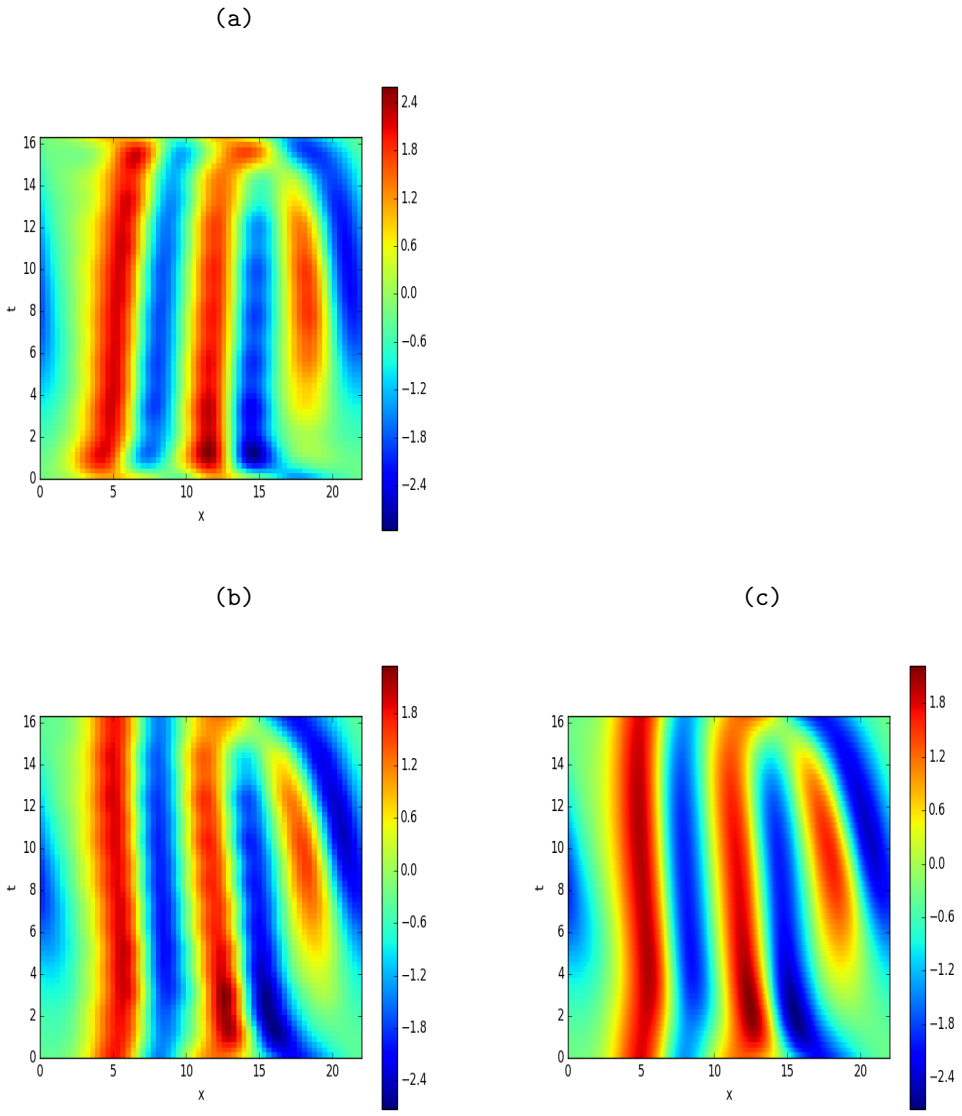

Figure 4: (a) Initial condition of the 16-by-16 space-by-time discretization of relative periodic orbit16.31. $L_0 = 22$. (b) Resulting periodic orbit after variational newton descent in time $L = 22, T = 15.7444884386$, (c) resulting periodic orbit after variational newton descent in space.

applying newton descent in *space*. If I integrate a spatially periodic initial condition in time, by virtue of the spatial periodicity there is a close recurrence in the spatial direction (close and not exact only due to discretization I believe). If I've thought about this the right way. It's the smartest way I can think of to generate an initial condition for the spatial newton descent (**??**) given that my spatial integration code is ill-behaved. If my *spatial* code was working and there is no lapse in my rationale then it might actually have been a way to produce smooth initial guess loops for the *time* direction newton descent code.

Past two days have been spent making changes to newton descent and testing those changes. For any changes that I make that translate generically to any system I try to test them with Rössler first before applying them to antisymmetric subspace $\mathbb{U}^+$ of Kuramoto-Sivashinsky. The other changes that are unique to antisymmetric Kuramoto-Sivashinsky must of course be tested in that realm. The changes are looking promising, as I can find longer periodic orbits in the Rössler system now, however the period seems to be slightly off (Integrating the solution after application of newton descent yields a solution that is periodic but overlaps). This is my number one suspect for why the newton descent code continues to stall at the moment; however, for the Rössler system although it is much slower for longer periodic orbits, it still converges very quickly once $F^2 < 1$.

spatial newton descent stalls out around cost functional value $\mathcal{F}^2 \approx 10^{-8}$. Final configuration space velocity field is a highly oscillating very non-physical type solution that looks like the result of aliasing; I believe I need to incorporate more Fourier modes as a first step to fix the issue, or the number of discretization points, although doing so leads me to run out of memory

I realized that I should not have to worry about implementing a slicing condition in the spatial version of variational newton descent; All that was required in the time case was to reduce the symmetry associated with the marginal direction parallel to velocity, i.e. a Poincaré section. I didn't worry about the spatial translational invariance and it was able to converge to a solution just fine in the time case.

In this line of thought, because space and time have their roles reversed, I should only have to take into consideration the translational invariance in the spatial direction and not the SO(2) symmetry in the (now periodic) time direction.

I also changed the definition of the stability matrix elements that arise due to the nonlinearity in hopes this will fix my problems; All in all, things are working *much* better than they were even when compared to yesterday, although the convergence properties are still not where they need to be in order to say I "found" an new solution yet (For an initial condition whose initial cost functional value is $\mathcal{F}_0^2 \approx 5$ my code is able to reduce it to $\mathcal{F}_\tau^2 \approx 10^{-1}$). I'm currently testing my code with discretized versions of pre-periodic orbit10.2, but I am going to try to see what happens to an more general initial condition next.

There also might be a smarter way of choosing a constraint that enables better convergence, as opposed to the "first coordinate" hyperplane (i.e. the first Fourier mode in most systems). I'm currently playing around with using a hyperplane condition on the "more dynamical" variables which is a hasty and crude name not to be taken seriously. What I mean by this is that in (**??**) the spatial derivatives of the Fourier coefficients of $u^{(3)}$, which represents the third derivative, are much more complex than the other derivatives, so perhaps using a hyperplane condition on one of these coordinates would be better; this hasn't seemed to be the case yet.

Realized I made a small mistake when thinking about using Fourier transforms along the parameterization direction in order to approximate the loop tangent space (145). I thought

that I would have to somehow permute the elements (145) of the "Loop Vector" (vector that encodes the parameterization of initial condition for periodic orbit search). The reasoning behind this was in order to use differentiate with respect to a parameterization variable s, I would need the elements to be in sequential order with respect in parameterization variable s, in order to multiply by vector $i\vec{m}$, where m is the conjugate variable (in a Fourier transform sense) to s. This is ***not*** the case, as I can merely exploit the Kronecker outer product to produce a diagonal matrix such that along the diagonal there are M duplicates of each element of $\vec{m}$

We are essentially diagonalizing a sparse matrix for $\mathcal{O}(M\,(nlog(n)))$ flops from taking M Fourier transforms of length n = power of 2. This is all well and good, but I think that there might be complications from the stability matrices; I need to go through the calculation, but the naive way to write the stability matrices in their new representation is: $\tilde{\mathbf{A}} = \mathbf{FAF}^*$, where F is a unitary matrix representing the discrete Fourier transform.

When you include the amount of flops needed to produce the product of these matrices, I don't think the benefits outweigh the costs *unless* a much smaller discretization can be used due to the convergence of Fourier coefficients (i.e. a truncation in the parameterization variable).

Next is the representation of the fictitious time evolution as a system of linear equations, similar to (**??**), which is restated here for comparison to the new system of equations.

The old linear system is given by,

$$\begin{bmatrix} M & -v \end{bmatrix} \begin{bmatrix} \delta\tilde{x} \\ \delta\lambda \end{bmatrix} = \delta\tau \begin{bmatrix} \lambda v - \tilde{v} \end{bmatrix}, \tag{140}$$

where $M = D - \lambda Diag(A_n)$ with D being the finite difference matrix, and A_n a block diagonal matrix containing stability matrices.

Now, the equations the same form, with new variables described by over-bars

$$\begin{bmatrix} \bar{M} & -\bar{v} \end{bmatrix} \begin{bmatrix} \delta\tilde{\bar{x}} \\ \delta\bar{\lambda} \end{bmatrix} = \delta\tau \begin{bmatrix} \lambda\hat{\bar{v}} - \tilde{\bar{v}} \end{bmatrix}, \tag{141}$$

where $\bar{M} = \mathbf{F}Diag(i\vec{m})\mathbf{F}^* \otimes \mathbf{I}_d - \lambda Diag(A_n)$ and $\bar{v} = \mathcal{F}(v)$, $\tilde{\bar{v}} = (Diag(i\vec{m}) * \tilde{\bar{x}}$. [3]

Spatial Newton Descent Rewrote the main body of the fictitious time evolution loop to hopefully deal with memory management a bit better, but still getting memory issues. Waiting on latest Arnoldi iteration to finish before using terminal to do calculations.

Waffling between implementation of least squares solver for pseudoinverse variational newton descent.

GMRES seems to be locked by memory. Also tried to implement QR decomposition as in TrefethenTrefethen97 but trying to stick to pseudoinverse and least squares solvers as they typically work better; also keeping track of large matrices is a downside.

The best results, (i.e. better than square matrix problem, but still not good enough) was with SciPy's LSQR algorithm, which, in the paper that it is based on ref. [52], describes it as a "conjugate-gradient-like" algorithm, with better stability. I haven't gotten into the nitty gritty as of yet.

A useful class of numerical methods often used in optimization [6, 15] are known as *descent methods*. In our context, optimization denotes the numerical process of finding

[3] Matt: I changed this such that the only difference between my current code and this formulation is the calculation of approximate tangent space via Fourier methods.

minimizers of a scalar valued cost function which vector valued inputs, the spatiotemporal Fourier modes and spatiotemporal parameters. Broadly speaking, descent methods are numerical methods which iteratively solve unconstrained optimization. These methods accomplish this by one way or another providing a direction to step in which monotonically decreases the value of the cost functional, hence "descent" (assuming a non-negative scalar cost function). The method with which to compute the descent direction is the distinguishing property between descent methods. In the limit of an infinitesimal step size, the iterative descent can be characterized as a fictitious flow with respect to a fictitious time [36]. The advantages of descent methods are that they do not require the construction nor the inversion of any matrices. The computational and memory requirements are relatively cheap in comparison to direct methods but the trade off is the rate of convergence. The convergence of a descent method is guaranteed but only in the impractical limit of infinite fictitious time. There are some tools for improving the rate of convergence such as preconditioning operators which will be discussed in a later section.

$$\frac{\partial x}{\partial \tau}(s+\omega,\tau)+\frac{\partial x}{\partial s}(s+\omega,\tau)\frac{\partial \omega}{\partial \tau}(\tau)-J(x(s,\tau))\frac{\partial x}{\partial \tau}(s,\tau)=f(x(s,\tau))-x(s+\omega,\tau), \quad (142)$$

The old linear system is given by,

$$\begin{bmatrix} M & -v \end{bmatrix}\begin{bmatrix} \delta\tilde{x} \\ \delta\lambda \end{bmatrix}=\delta\tau\begin{bmatrix} \lambda v-\tilde{v} \end{bmatrix}, \quad (143)$$

where $M = D - \lambda Diag(A_n)$ with D being the finite difference matrix, and A_n a block diagonal matrix containing stability matrices.

Now, the equations the same form, with new variables described by over-bars

$$\begin{bmatrix} \bar{M} & -\bar{v} \end{bmatrix}\begin{bmatrix} \delta\tilde{\bar{x}} \\ \delta\bar{\lambda} \end{bmatrix}=\delta\tau\begin{bmatrix} \lambda\hat{\bar{v}}-\tilde{\bar{v}} \end{bmatrix}, \quad (144)$$

where $\bar{M} = \mathbf{F}Diag(i\vec{m})\mathbf{F}^* \otimes \mathbf{I}_d - \lambda Diag(A_n)$ and $\bar{v} = \mathcal{F}(v)$, $\tilde{\bar{v}} = (Diag(i\vec{m}) * \tilde{\bar{x}}$. [4] e are essentially diagonalizing a sparse matrix for $\mathcal{O}(M\,(nlog(n)))$ flops from taking M Fourier transforms of length n = power of 2. This is all well and good, but I think that there might be complications from the stability matrices; I need to go through the calculation, but the naive way to write the stability matrices in their new representation is: $\tilde{\mathbf{A}} = \mathbf{FAF}^*$, where F is a unitary matrix representing the discrete Fourier transform.

When you include the amount of flops needed to produce the product of these matrices, I don't think the benefits outweigh the costs *unless* a much smaller discretization can be used due to the convergence of Fourier coefficients (i.e. a truncation in the parameterization variable).

Next is the representation of the fictitious time evolution as a system of linear equations, similar to (**??**), which is restated here for comparison to the new system of equations.

The best results, (i.e. better than square matrix problem, but still not good enough) was with SciPy's LSQR algorithm, which, in the paper that it is based on ref. [52], describes it as a "conjugate-gradient-like" algorithm, with better stability. I haven't gotten into the nitty gritty as of yet.

[4] Matt: I changed this such that the only difference between my current code and this formulation is the calculation of approximate tangent space via Fourier methods.

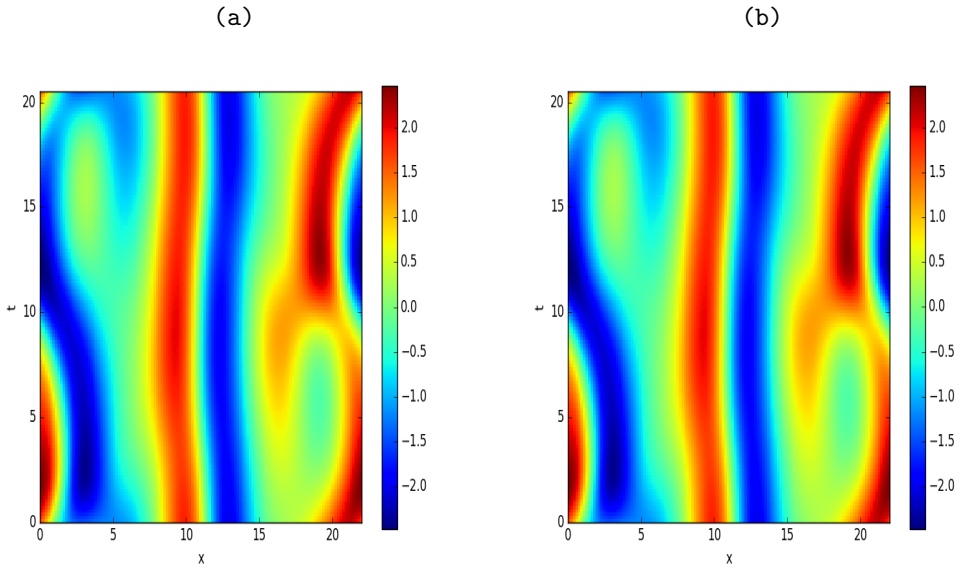

Figure 5: (a) Initial condition of the 32-by-32 space-by-time discretization of pre-periodic orbit10.2: $(L_0, T_0) = (L_0, 2T_{p_0}) = (22, 20.5057459345)$. (b) Resulting spatiotemporal fixed point $(L_p, 2T_p) = (22.0000104401, 20.5057499188)$

While waiting for Arnoldi iteration to finish so that I could begin testing the spatial variational newton descent without fear of memory problems I was trying to think about the best way to use (**??**), which I will restate here.

First, $\mathbf{u}_{k,\ell}$ represents matrix elements, so it makes sense to rewrite the equation as a matrix equation. Define matrices $\mathbf{Q}_1 \equiv Diag(-q_k^2 + q_k^4)$, $\mathbf{W} \equiv Diag(i\omega_\ell)$, $\mathbf{Q}_2 \equiv Diag(\frac{iq_k}{2})$, and let the two dimensional FFT be represented by matrix multiplication $\mathbf{U} = \mathbf{F}_M \mathbf{u} \mathbf{F}_N$, where the matrix elements $U_{k,\ell} = \mathbf{u}_{k,\ell}$.

and therefore we have an equation of the form: $f_{k,\ell}(\mathbf{u}_{k,\ell}) - \mathbf{u}_{k,\ell} = 0$, where the Jacobian matrix is given by the fourth rank tensor that arises from taking partial derivatives with respect to $\mathbf{u}_{k,\ell}$ More to be derived in the future, hoping to make headway into finding tori; I can't tell if this equation is going to be useful or if I should really be working towards deriving and learning variational newton descent equations for finding tori similar to Lan, Chandre and Cvitanović [36].

Application of the spatial newton descent code to ergodic trajectories that have been deformed to be periodic in time were resulting in the "falling into equilibrium" problem, this was due to a bug where the wrong temporal system sizes were being used.

Application of spatial newton descent on pre-periodic orbit10.2 results in a reduced cost functional but seems rather obstinate in regards to convergence. Luckily, the approximate loop seems to fluctuate around spatial extent $L = 22$. I think this is a good indication as it means the spatial newton descent is capturing the spatial geometry of pre-periodic orbit10.2. That is to say, even while reducing the cost functional the solution doesn't want to betray itself, as it originates from the spatial system size $L = 22$.

I tried whether it be the error tolerances, step sizes (variable or constant), initial condition discretizations, least squares solvers, pseudoinverse or regular inverse methods, hypersurface constraints, matrix preconditioners, etc, did not help the converge properties. By examining the corrections being applied to deform the loop, specifically the maximum

correction applied in each step it seems most of the steps are modifying the "period" i.e. the spatial extent of the initial condition the most. There might be some way to discourage this with an additional condition on the rescaling factor λ that matches the magnitudes of the approximate tangent space to the actual tangent space.

I believe I got it working finally, however, the results so far aren't as interesting as I had dreamed. I finished this at the end of the day so I didn't get to test it too much, but so far there are two resulting possibilities. First, with a time periodic initial condition, i.e. one of the periodic orbits in time of Kuramoto-Sivashinsky, when allowing for spatial domain changes and changes to the temporal Fourier coefficients, the solution (only tested one so far) converged to one of the temporal equilibria of the Kuramoto-Sivashinsky system. This I believe is an indication that my code is indeed working, even though this was usually a sign of numerical issues when searching for periodic orbits in time using the variational methods, (i.e. the only way to reduce the cost functional $\mathcal{F}^2 = \lambda v - \tilde{v}$ is to send v and $\tilde{v}$ to 0. The reason I believe this is still valid is because the spatial derivatives of the equilibrium state found are nonzero; i.e. one of the "spatial periodic orbits" I have found is indeed the temporal equilibrium of the system.

After more investigation it turns out that the spatial variational code is indeed not working yet. Tried to put resulting orbits into time integrator in order to reproduce the result and got an unmatching solution. figure 7 is my newest "results". It uses a $\overline{rpo}_{16.31}$ of the Kuramoto-Sivashinsky equation as the initial condition. I think I was deceived by how nice it looks I suppose...couldn't find any errors today that could enable reproduction via time integration.

As an additional test I put the solutions into Burak's symmetry reduced time integrator to verify whether the "solution" in figure 7 was a relative periodic orbit but alas there was no luck; there is some other error that I haven't been able to identify as of yet. Found another negative sign error, updating figure **??**.

As a means of cross-checking results I wrote variational newton descent code for time, debugged, and got it working. It follows the "direct-matrix" approach that I have been finding useful. The velocity equation takes the form

figure 6 is a comparison between resulting orbits from both space and time variational newton descent. The general structure of the resulting orbit is preserved even though the periods are quite different. At first I believed that these were supporting evidence that something is going right but now I am confused. The spatial domain resulting from the *spatial* newton descent is largely unchanged, meaning that the orbit that results should be a unique solution with a unique period; but the *time* variational newton descent changes the period somewhat drastically, and because the spatial domain size is fixed this might mean that I am finding the same solution but this is contradictory because the periods are so different. Also good evidence that there is something wrong is that the *time* newton descent is taking a relative periodic orbit initial condition and seemingly changing it into a periodic orbit? I don't know what to think of it but I am glad that I rewrote the time newton descent code as it'll be a good stepping stone into whether or not I need to rewrite the equations in a symmetry reduced form in order to find relative periodic orbits or not. I think that's where I'm headed at least.

updated figure 7, figure **??**, and uploaded figure 8 to display current results.

After trying numerous small modifications to the variational method's system of equations from ref. [34] including implementing factors involving the coordinate involved in defining the in-slice time and other quantities involving the slice tangents I didn't get any improvements over what I already had. I still think the problem has to do with the slice

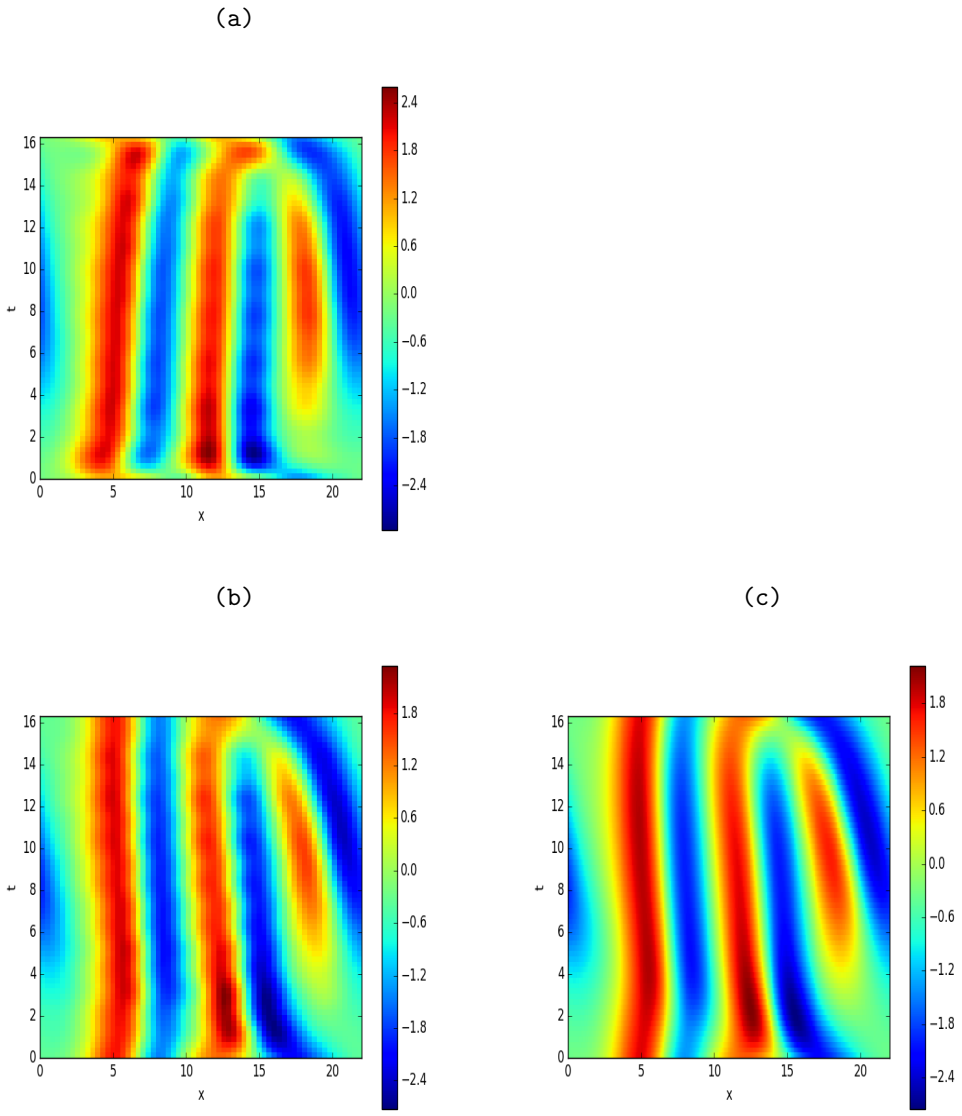

Figure 6: (a) Initial condition of the 16-by-16 space-by-time discretization of relative periodic orbit16.31. $L_0 = 22$. (b) Resulting periodic orbit after variational newton descent in time $L = 22, T = 15.7444884386$, (c) resulting periodic orbit after variational newton descent in space.

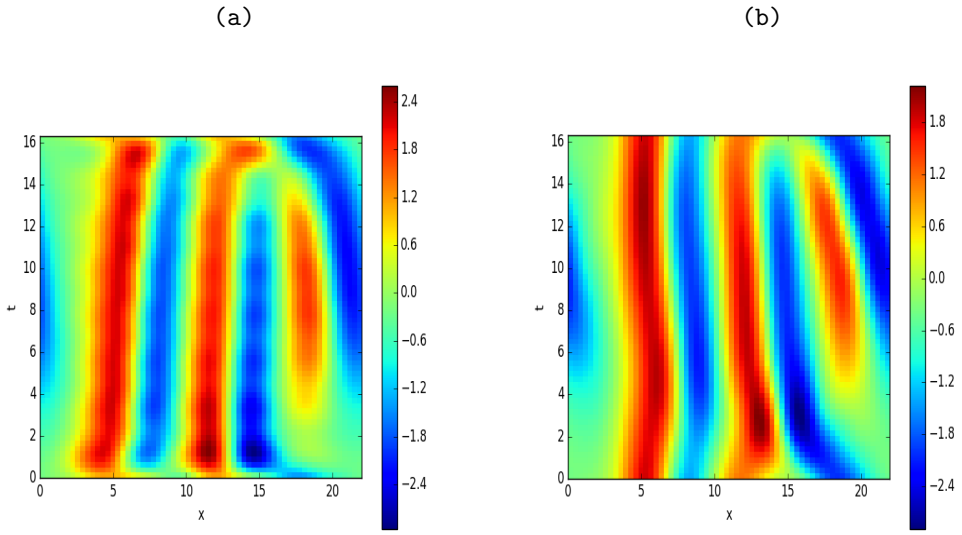

Figure 7: (a) Initial condition of the 16-by-16 space-by-time discretization of $\overline{rpo}_{16.31}$ ($L = 22$) for spatial variational newton descent of the Kuramoto-Sivashinsky equation (b)Resulting "spatial periodic orbit" (temporal equilibrium), with final spatial extent of $L = 21.9394614064$

time so I went over the derivation of the equations and I believe I found a correction that can be made.

$$D_{\mathcal{F}} = \mathbf{P}^{-1}\mathbf{F}^{-1}\mathbf{Diag}(is)\mathbf{FP} \quad \text{where,}$$

$\mathbf{F}$ = Block diagonal matrix composed of Fourier transform matrices (i.e. to Fourier transform each of the Fourier coefficient series with respect to parameterization variable), $\mathbf{P}$ = Permutation matrix to reorder in specific way to enable easy Fourier transforms.

Another main challenge is how to implement a slice condition to deal with translational invariance. Typically this is dealt with when the spatial Fourier series is being used, and therefore it is easier to represent a hypersurface that eliminates this marginal direction; in the spatial newton descent code (this is what I call using (**??**) with variational newton descent) I am trying to eliminate the translational freedom but it's not as straightforward as the first Fourier mode slice; as the first Fourier mode slice in this case would eliminate time translations. I've been looking towards some of the papers about invariant tori and their "phase conditions" as a possible means of escape.

I realized that I should not have to worry about implementing a slicing condition in the spatial version of variational newton descent; All that was required in the time case was to reduce the symmetry associated with the marginal direction parallel to velocity, i.e. a Poincaré section. I didn't worry about the spatial translational invariance and it was able to converge to a solution just fine in the time case.

I thought that I would have to somehow permute the elements (145) of the "Loop Vector" (vector that encodes the parameterization of initial condition for periodic orbit search). The reasoning behind this was in order to use differentiate with respect to a parameterization variable s, I would need the elements to be in sequential order with respect in parameterization variable s, in order to multiply by vector $i\vec{m}$, where m is the conjugate

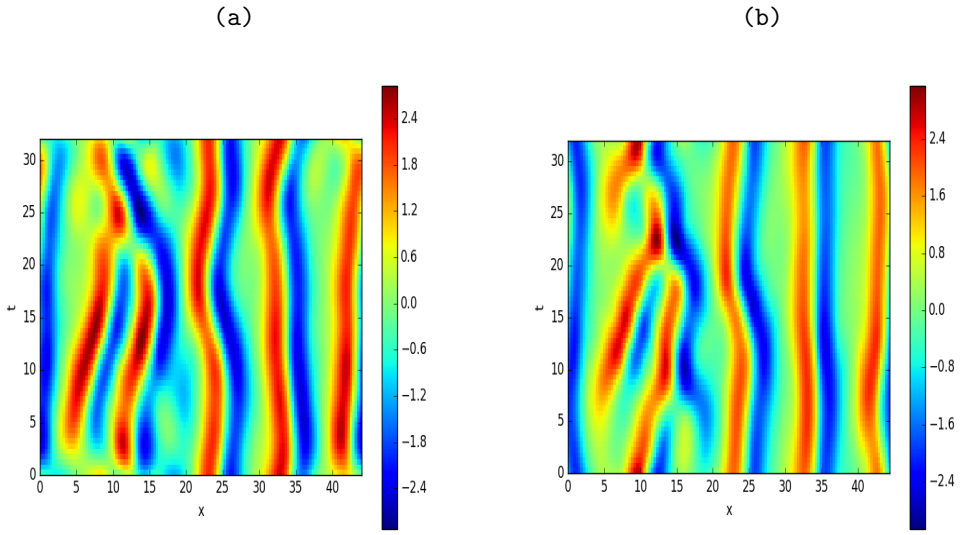

Figure 8: (a) Initial condition of the 32-by-16 space-by-time discretization of a piece of an ergodic trajectory that has been deformed to be periodic in time. $L = 44$. (b) Resulting spatiotemporal periodic orbit, with final spatial extent of $L = 44.3937151766$.

variable (in a Fourier transform sense) to s. This is ***not*** the case, as I can merely exploit the Kronecker outer product to produce a diagonal matrix such that along the diagonal there are M duplicates of each element of $\vec{m}$

We are essentially diagonalizing a sparse matrix for $\mathcal{O}(M\,(nlog(n)))$ flops from taking M Fourier transforms of length n = power of 2. This is all well and good, but I think that there might be complications from the stability matrices; I need to go through the calculation, but the naive way to write the stability matrices in their new representation is: $\tilde{\mathbf{A}} = \mathbf{FAF}^*$, where F is a unitary matrix representing the discrete Fourier transform.

Next is the representation of the fictitious time evolution as a system of linear equations, similar to (**??**), which is restated here for comparison to the new system of equations.

The old linear system is given by,

$$\begin{bmatrix} M & -v \end{bmatrix} \begin{bmatrix} \delta\tilde{x} \\ \delta\lambda \end{bmatrix} = \delta\tau \begin{bmatrix} \lambda v - \tilde{v} \end{bmatrix}, \tag{145}$$

where $M = D - \lambda Diag(A_n)$ with D being the finite difference matrix, and A_n a block diagonal matrix containing stability matrices.

Now, the equations the same form, with new variables described by over-bars

$$\begin{bmatrix} \bar{M} & -\bar{v} \end{bmatrix} \begin{bmatrix} \delta\tilde{\tilde{x}} \\ \delta\bar{\lambda} \end{bmatrix} = \delta\tau \begin{bmatrix} \lambda\hat{\tilde{v}} - \tilde{\tilde{v}} \end{bmatrix}, \tag{146}$$

where $\bar{M} = \mathbf{F}Diag(i\vec{m})\mathbf{F}^* \otimes \mathbf{I}_d - \lambda Diag(A_n)$ and $\bar{v} = \mathcal{F}(v)$, $\tilde{\tilde{v}} = (Diag(i\vec{m}) * \tilde{\tilde{x}}$. [5]

Next is the representation of the fictitious time evolution as a system of linear equations, similar to (**??**), which is restated here for comparison to the new system of equations.

[5] Matt: I changed this such that the only difference between my current code and this formulation is the calculation of approximate tangent space via Fourier methods.

The old linear system is given by,

$$\begin{bmatrix} M & -v \end{bmatrix} \begin{bmatrix} \delta\tilde{x} \\ \delta\lambda \end{bmatrix} = \delta\tau \begin{bmatrix} \lambda v - \tilde{v} \end{bmatrix}, \tag{147}$$

where $M = D - \lambda Diag(A_n)$ with D being the finite difference matrix, and A_n a block diagonal matrix containing stability matrices.

Now, the equations the same form, with new variables described by over-bars

$$\begin{bmatrix} \bar{M} & -\bar{v} \end{bmatrix} \begin{bmatrix} \delta\tilde{\tilde{x}} \\ \delta\bar{\lambda} \end{bmatrix} = \delta\tau \begin{bmatrix} \lambda\hat{\tilde{v}} - \tilde{\tilde{v}} \end{bmatrix}, \tag{148}$$

where $\bar{M} = \mathbf{F} Diag(i\vec{m})\mathbf{F}^* \otimes \mathbf{I}_d - \lambda Diag(A_n)$ and $\bar{v} = \mathcal{F}(v)$, $\tilde{\tilde{v}} = (Diag(i\vec{m}) * \tilde{\tilde{x}}$. [6]

[6] Matt: I changed this such that the only difference between my current code and this formulation is the calculation of approximate tangent space via Fourier methods.

APPENDIX B

DISCRETE LAGRANGIAN METHODS

I messed around with a calculation but just came to the same conclusion I had previously reached, namely if the adjoint variable v is chosen to be the Kuramoto-Sivashinsky equation then the variational derivative of the formal Lagrangian in the Ibragimov sense corresponds to the adjoint descent direction. This is perhaps why it works so well for me. i.e.the variational derivative obeys the equation

To avoid confusion, the analysis is different than the study of "discrete Lagrangian systems", commonly seen in the context of variational time integrators refs. [43, 44].

Therefore, if possible, we want to find an alternative type of analysis which is as useful without having to change our spatiotemporal formulation. There are two avenues of pursuit towards this endeavor. The first is known as Hill's formula [2]. It discusses how the determinant of a finite matrix of the Hessian of an action functional of a discrete Lagrangian system, can be related to the eigenvalues of the monodromy matrix corresponding to a critical point of said action functional, which represents a critical point. As stated in [2], Hill proposed this formula without any proof of convergence of the determinant (his derivation utilized an infinite dimensional Hessian), but a rigorous proof was later presented by Poincaré. The application of this formula requires a discrete Lagrangian system, which means we need equations which have a Lagrangian structure. The cost functional as described by (**??**), although a scalar function of our system variables, does not have the correct Lagrangian structure that is needed, so instead we introduce the concept of *formal Lagrangians* refs. [22, 25, 32]. In this context the Lagrangian structure is imposed by construction, allowing for a valid application of Hill's formula. The interpretation of this application is slightly confusing however as the Hessian and monodromy matrix are now functions of the "original" velocity field and its partial derivatives but also a collection of adjoint variables.

$$\begin{aligned} \frac{\delta L(u,v)}{\delta u} &= -v_t + v_{xx} + v_{xxxx} - uv_x \\ \frac{\delta L(u,-F(u))}{\delta u} &= F_t - F_{xx} - F_{xxxx} + uF_x \equiv -J^\dagger F \end{aligned} \tag{149}$$

Following sect. 2 *Quasi-self-adjoint equations* of Ibragimov [22] (which does not reference refs. [20, 21]), we can write the *formal Lagrangian* of the Kuramoto-Sivashinsky equation to derive the spatiotemporal adjoint equations in terms of the original spatiotemporal field $u(x,t)$, and then one is free to use whatever representation suits the user in discretization; the cavaet is that numerically it seems better to use a real valued representation Fourier representation for the adjoint descent. [1]

Ibragimov notation, for the Kuramoto-Sivashinsky case: the independent variable is denoted by x. The dependent variable is u, used together with its first-order partial derivative $u_{(1)}$:

$$u_{(1)} = \{u_i^\alpha\}, \quad u_i^\alpha = D_i(u^\alpha),$$

[1] 2018-05-08 Predrag: Ibragimov [22] is included in Archives of ALGA **4**.

and higher-order derivatives $u_{(2)}, \ldots, u_{(4)}$, where

$$u_{(2)} = \{u^\alpha_{ij}\}, \quad u^\alpha_{ij} = D_i D_j(u^\alpha), \ldots,$$

$$u_{(s)} = \{u^\alpha_{i_1 \cdots i_s}\}, \quad u^\alpha_{i_1 \cdots i_s} = D_{i_1} \cdots D_{i_s}(u^\alpha).$$

Here D_i is the total differentiation with respect to x^i :

$$D_i = \frac{\partial}{\partial x^i} + u^\alpha_i \frac{\partial}{\partial u^\alpha} + u^\alpha_{ij} \frac{\partial}{\partial u^\alpha_j} + \cdots . \tag{150}$$

Using the definition for the *formal Lagrangian* $\mathcal{L}$,

$$\mathcal{L} \equiv v \left[u_t + u_{xx} + u_{xxxx} + u u_x \right], \tag{151}$$

and then taking the functional derivative,

$$\frac{\delta \mathcal{L}}{\delta u} = 0 . \tag{152}$$

The surviving terms from this functional derivative are

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial u} &= v u_x \\ -\partial_t \frac{\partial \mathcal{L}}{\partial u_t} &= -v_t \\ -\partial_x \frac{\partial \mathcal{L}}{\partial u_x} &= -v u_x - u v_x \\ \partial_x^2 \frac{\partial \mathcal{L}}{\partial u_{xx}} &= v_{xx} \\ \partial_x^4 \frac{\partial \mathcal{L}}{\partial u_{xxxx}} &= v_{xxxx}, \end{aligned} \tag{153}$$

where the sum of these terms equals (152) and hence must be zero. The resultant adjoint equation is ($\pm v u_x$ terms cancel),

$$-v_t + v_{xx} + v_{xxxx} - u v_x = 0 . \tag{154}$$

If we take the adjoint variable to be the Kuramoto-Sivashinsky equation,

$$F \equiv v = -\left[u_t + u_{xx} + u_{xxxx} + u u_x \right] , \tag{155}$$

we arrive at the equation which *I claim* provides adjoint descent direction without explicit construction of the gradients matrix J,

$$-J^\dagger F = (\partial_t + \partial_x^2 + \partial_x^4) F + u \partial_x F \tag{156}$$

(I believe the negative sign in (155) is motivated so that the functional derivative is strictly decreasing or zero).

Numerical evidence is suggestive as the real valued adjoint descent is working better than before when I was trying to reverse engineer $J^\dagger F$ in a matrix-free way.

I messed around with a calculation but just came to the same conclusion I had previously reached, namely if the adjoint variable v is chosen to be the Kuramoto-Sivashinsky equation

then the variational derivative of the formal Lagrangian in the Ibragimov sense corresponds to the adjoint descent direction. This is perhaps why it works so well for me. i.e.the variational derivative obeys the equation

$$\begin{aligned} \frac{\delta L(u,v)}{\delta u} &= -v_t + v_{xx} + v_{xxxx} - uv_x \\ \frac{\delta L(u,-F(u))}{\delta u} &= F_t - F_{xx} - F_{xxxx} + uF_x \equiv -J^\dagger F \end{aligned} \tag{157}$$

Lagrangian systems are conservative dynamical systems which have a variational formulation. To understand the relation between the discrete time Hamiltonian and Lagrangian formulations, one needs to understand the discrete mapping generating function, such as (**??**).

[2]

[3]

Consider a cat map [1] of form

$$\begin{pmatrix} q_{t+1} \\ p_{t+1} \end{pmatrix} = A \begin{pmatrix} q_t \\ p_t \end{pmatrix} \qquad \mod 1\,, \qquad A = \begin{pmatrix} s-1 & 1 \\ s-2 & 1 \end{pmatrix}, \tag{158}$$

with both q_t and p_t in the unit interval, A a linear, state space (area) preserving map of a 2-torus onto itself, and $s = \mathrm{tr}\, A > 2$ an integer. Implement explicitly, as in (**??**), the mod 1 operation by introducing m^q and m^p winding numbers,

$$\begin{pmatrix} q_{t+1} \\ p_{t+1} \end{pmatrix} = A \begin{pmatrix} q_t \\ p_t \end{pmatrix} - \begin{pmatrix} m^q_{t+1} \\ m^p_{t+1} \end{pmatrix}. \tag{159}$$

This is a non-autonomous, time-forced Hamiltonian equation of motion of form (**??**,**??**):

$$q_{t+1} = q_t + p_{t+1} + ({s_{t+1}}^p - {s_{t+1}}^q) \tag{160}$$

$$p_{t+1} = p_t + (s-2)q_t - {s_{t+1}}^p\,, \tag{161}$$

with the force and the corresponding potential energy given by

$$P(q_t) = -\frac{dV(q_t)}{dq_t} = (s-2)q_t - {s_{t+1}}^p\,, \tag{162}$$

$$V(q_t) = \frac{1}{2}(2-s)\, q_t^2 + {s_{t+1}}^p q_t\,. \tag{163}$$

As always, the Lagrangian, or, in the parlance of discrete time dynamics, the *generating function* $L(q_i, q_{i+1})$, is given by the difference of the kinetic and potential energies, where in the literature [2, 41, 42, 45] there are different choices of the instant in time at which $V(q)$ is be evaluated. We define the generating function as

$$L(q_t, q_{t+1}) = \frac{1}{2}p_{t+1}^2 - V(q_t)\,.$$

[2] 2019-08-05 Predrag: In preparing this summary we have found expositions of Lagrangian dynamics for discrete time systems by MacKay, Meiss and Percival [42, 45], and Li and Tomsovic [39] particulary helpful.

[3] 2018-02-16 Predrag: We need a simple explanation for why the 2-dimensional $1 - A^n$ and the linearization of the periodic orbit $2n$-dimensional Hessian matrix give the same multipliers.

Next one eliminates momenta in favor of velocities, using (160)

$$\begin{aligned} L(q_t, q_{t+1}) &= \frac{1}{2}(q_{t+1} - q_t - {s_{t+1}}^p + {s_{t+1}}^q)^2 + \frac{s-2}{2} q_t^2 - {s_{t+1}}^p q_t \\ &= \frac{1}{2} q_{t+1}^2 + \frac{s-1}{2} q_t^2 - q_t q_{t+1} \\ &\quad -q_{t+1}{s_{t+1}}^p + q_{t+1}{s_{t+1}}^q - q_t {s_{t+1}}^q + \text{constant}\,. \end{aligned} \tag{164}$$

And this generating function satisfies (174).

Consider a symplectic ("area preserving") map acting on phase space

$$x_{t+1} = M(x_t)\,, \qquad x_t = (q_t, p_t)$$

that maps x_t to x_{t+1} while preserving the symplectic area.

A *path* is any set of successive *configuration space* points

$$\{q_i\} = \{q_t, q_{t+1}, \cdots, q_{t+k}\}\,. \tag{165}$$

In a Lagrangian system each path of finite length in the configuration space is assigned an *action*. [4]

To get the action of an orbit from time t_0 to t_n, we only need to sum (164) over intermediate time steps:

$$W(q_{t_0}, q_{t_0+1}, \ldots, q_{t_n-1}, q_{t_n}) = \sum_{t=t_0}^{t_n-1} L(q_t, q_{t+1})\,. \tag{166}$$

For example, in a discrete-time one-degree-of-freedom Lagrangian system with the configuration coordinate q_i at the discrete time i, and *generating function* ("Lagrangian density") $L(q_i, q_{i+1})$, the action of path $\{q_i\}$ is

$$W_{t,t+k} \equiv \sum_{i=t}^{t+k-1} L(q_i, q_{i+1})\,, \tag{167}$$

For 1-dof systems, the geometrical interpretation of the action $W_{t,t+k}$ is that $L(q_t, q_{t+1})$ is, up to an overall constant, the phase-space area below the p_t to p_{t+k} graph for the (q_t, q_{q+k}) path in the (q, p) phase plane. [5]

Denoting the derivatives of the generating function $L(q, q')$ as

$$\begin{aligned} L_1(q, q') &= \frac{\partial}{\partial q} L(q, q')\,, \quad L_2(q, q') = \frac{\partial}{\partial q'} L(q, q') \\ L_{12}(q, q') &= L_{21}(q, q') = \frac{\partial^2}{\partial q \partial q'} L(q, q')\,, \end{aligned} \tag{168}$$

the *momenta* are given by [42, 45]

$$p_n = -L_1(q_n, q_{n+1})\,, \quad p_{n+1} = L_2(q_n, q_{n+1})\,. \tag{169}$$

[4] 2019-08-04 Predrag: repeat of text in catLagrang.tex

[5] 2016-11-11, 2018-09-26 Predrag: What follows is (initially) copied from Li and Tomsovic [39], *Exact relations between homoclinic and periodic orbit actions in chaotic systems* arXiv source file, then merged with the MacKay-Meiss-Percival action principle refs. [42, 45].

The twist condition

$$\partial p_{n+1}/\partial q_n \neq 0 \text{ for all } p_{n+1}, q_n\,, \tag{170}$$

ensures that

$$L_{12}(q_n, q_{n+1}) \neq 0\,. \tag{171}$$

We distinguish a *path* (165), which is any set of successive points $\{q_n\}$ in the configuration space, from the *orbit segment* $M^k(x_n)$ from x_n to x_{n+k}, a set of successive *phase space* points

$$\{x_i\} = \{x_n, x_{n+1}, \cdots, x_{n+k}\}\,. \tag{172}$$

that extremizes the *action* (167), with momenta given by (169). In other words, not only q_n, but also p_n have to align from phase space point to phase space point [53],

$$\frac{\partial}{\partial q_n}\left(L(q_{n-1}, q_n) + L(q_n, q_{n+1})\right) = 0\,. \tag{173}$$

Any finite path for which the action is stationary with respect to variations of the segment keeping the endpoints fixed, is called an orbit segment or *trajectory* [14]. Infinite paths for which each finite segment is an orbit segment are called *orbits*.

Given by Keating [29], for the 1-dimensional cat map (**??**), the action of a one-step orbit (which is the generating function) from (x_t, p_t) to (x_{t+1}, p_{t+1}) can be written as (**??**). And the map (159) can be generated using [42, 45]:

$$p_t = -\partial L(x_t, x_{t-1})/\partial x_t\,, \quad p_{t+1} = \partial L(x_t, x_{t-1})/\partial x_{t+1} \tag{174}$$

[6]

Setting the first variation of the action δW to 0 we get:

$$\frac{\partial W}{\partial x_t} = \frac{\partial L(x_t, x_{t+1})}{\partial x_t} + \frac{\partial L(x_{t-1}, x_t)}{\partial x_t} = 0 \tag{175}$$

$$\Rightarrow -x_{t-1} + s x_t - x_{t+1} = {s_{t+1}}^x - {s_t}^x + {s_t}^p\,. \tag{176}$$

This is the Percival-Vivaldi 2-step difference equation of the cat map with $s_t = {s_{t+1}}^x - {s_t}^x + {s_t}^p$.

Using (164–166) we can compute the action of any finite trajectory. For a trajectory $\ldots x_{t-1} x_t x_{t+1} x_{t+2} \ldots$, the action can be written as:

$$W(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top H \boldsymbol{x} - \boldsymbol{s}^\top \boldsymbol{x}\,, \tag{177}$$

where $\boldsymbol{x}$ and $\boldsymbol{s}$ are column vectors,

$$\boldsymbol{x} = \begin{bmatrix} \vdots \\ x_{t-1} \\ x_t \\ x_{t+1} \\ x_{t+2} \\ \vdots \end{bmatrix}, \quad \boldsymbol{s} = \begin{bmatrix} \vdots \\ s_{t-1} \\ s_t \\ s_{t+1} \\ s_{t+2} \\ \vdots \end{bmatrix}, \tag{178}$$

[6] Han 2019-08-01: (**??**) is given by Keating [29] but I cannot find the derivation of this generating function in that paper and the papers referred [28, 54]. The following derivation of generating function is from our blog.

and the Hessian H is a Toeplitz matrix

$$H = \begin{bmatrix} \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \ddots & s & -1 & 0 & 0 & \dots & 0 & 0 & \ddots \\ \ddots & -1 & s & -1 & 0 & \dots & 0 & 0 & \ddots \\ \ddots & 0 & -1 & s & -1 & \dots & 0 & 0 & \ddots \\ \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots \\ \ddots & 0 & 0 & \dots & \dots & \dots & s & -1 & \ddots \\ \ddots & 0 & 0 & \dots & \dots & \dots & -1 & s & \ddots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix} . \tag{179}$$

For an orbit with finite length, we need to know the bc's to find the action at boundaries. Note that the action computed in this way will not have the constant terms in (164). The matrix H has same effect as $(s-\Box-2)$ where the $\Box$ is the discrete one-dimensional Laplacian defined in (**??**).

[7] To explore this we will introduce what is sometimes referred to as the *formal Lagrangian* refs. [3, 22–25, 32, 61]

$$\mathcal{L} = vG(u(x,t), \dots, u_{(n)}(x,t)) . \tag{180}$$

This construction introduces the adjoint variable v, which is quitessentially a Lagrange multiplier as the form of (180) implies. Although relatively simplistic in its construction, the corresponding Euler-Lagrange equations of the formal Lagrangian (180) provide us with the Kuramoto-Sivashinsky equation (**??**) and its adjoint (182). The Euler-Lagrange equations can be produced by taking the variational derivative (181) of (180) with respect to either u and v separately.

$$\frac{\delta}{\delta u} = \frac{\partial}{\partial u} + \sum_{n=1}^{\infty} (-1)^s D_{i_1} \dots D_{i_n} \frac{\partial}{\partial u_{i_1 \dots i_n}} \tag{181}$$

which is the operator that produces the Euler-Lagrange equations of the corresponding Lagrangian [24].

The recovery of the original equation is self evident as $\frac{\delta \mathcal{L}}{\delta v} = \frac{\partial \mathcal{L}}{\partial v} = G$. For the adjoint equation we have

$$\frac{\delta \mathcal{L}}{\delta u} = G^*(x,t,u,v,v_t,v_x,v_{xx},v_{xxxx}) = -v_t + v_{xx} + v_{xxxx} - uv_x \tag{182}$$

both (182) and (**??**) create what is known as the *optimality system* of equations [19], show in (183) for completeness.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial v} &\equiv G = u_t + u_{xx} + u_{xxxx} + uu_x \\ \frac{\partial \mathcal{L}}{\partial u} &\equiv G^* = -v_t + v_{xx} + v_{xxxx} - uv_x \end{aligned} \tag{183}$$

[7] 2019-09-11 Predrag: I am thinking of moving everything from (180) to (206) into a Chapter of your thesis, remarking here briefly that Ibragimov methods seem to have not worked for you?

The adjoint variable v is to eventually be replaced by a function of u and its derivatives such that the system of equations (183) is solved whenever u is a solution. The main method to determine the explicit form of this substitution is to find the form of v which transforms the adjoint equation into the original equation. If the substitution $v = u$ satisfies this requirement, then the equation is said to be *self adjoint*. Specifically, the operator resultant from linearization of (**??**) is the actual object with this property. Other than self-adjointness there are also the notions of quasi-self adjoint equations and nonlinearly adjoint equations refs. [23–25, 61]. The substitutions that define quasi self-adjoint equations and nonlinearly self-adjoint equations are functions $v = \phi(u)$ and $v = \phi(x^i, u, u^{(n)})$ that transform (182) into (**??**) (the term $u^{(n)}$ represents all derivatives up to order n). Upon substitution, the self-adjoint condition can be written explicitly as

$$G^*(x, t, u, \phi(x, t, u, u^{(n)})) = \lambda G(x, t, u)\,, \tag{184}$$

For the adjoint equation (182) the only solution that guarantees the adjoint equation (182) is satisfied when u is a solution is $v = c$ where c is a constant. This is determined by making the substitution $v = \phi(x^i, u, u^{(n)})$ and solving the set of equations similar to the determining equations which produced the generators of the Lie algebra of infinitesimal symmetries. Specifically, the result of matching terms in (184) can be summarized by two of the resulting equations,

$$\begin{aligned} \phi_u &= \lambda \\ \phi_u &= -\lambda\,, \end{aligned} \tag{185}$$

which in turn implies that λ is equal to zero. This might seem like a red flag, the invariant 2-tori of the Kuramoto-Sivashinsky equation are critical points of (180) and this seems to indicate that is not possible; but $v = G(u)$ is actually a representation of the trivial solution when u is a solution because $G(u) = 0$, so there is no contradiction.

There is an another whose Euler-Lagrange equations are the same. Following the analysis of Burgers' equation and the prescription of [26] we find an alternative form of the Lagrangian for the Kuramoto-Sivashinsky equation to be

$$\mathcal{L} = \frac{1}{2}(vu_t - uv_t) - u_x v_x + u_{xx} v_x x + \frac{u}{3}(vu_x - uv_x) \tag{186}$$

There are a number of mathematical and numerical techniques available to our doubly periodic variational problem and not the equivalent dynamical system. For instance, we can utilize Lie group analysis to derive continuous symmetries pertaining to our Lagrangian. y Noether's Theorem these symmetries imbue the corresponding Euler-Lagrange equations with conservation laws [49]. On the numerical front we can utilize spatiotemporal two dimensional spectral methods (collocation methods, technically).

Pertaining to our system of equations, (**??**) and (182). There are a number of different paths we can take for this type of analysis but for now we present the investigation of whether or not there are any non-trivial conservation laws involving the Euler-Lagrange equations resulting from the Lagrangian density (180). As a note we could write (180) in an alternate manner, and then search for variational symmetries [50]. These would be imparted onto (**??**) and (182) because they are the corresponding Euler-Lagrange equations.

We already know the equations with which we intend to work so we may begin after introducing some notation. In order to derive conserved quantities using the machinery of [21] we first need to find the vector fields that span the Lie algebra of infinitesimal

symmetries of our equations. Our description follows Theorem 2.36 from [50] in terms of notation.

To begin, take an arbitrary vector field defined on the space $X \times U$ which contains the independent variables x^i and dependent variables u^α such that

$$\mathbf{v} = \xi^i(x^i, u^\alpha)\frac{\partial}{\partial x^i} + \phi_\alpha(x^i, u^\alpha)\frac{\partial}{\partial u^\alpha}\,, \tag{187}$$

where summation is implied by repeated indices. To create a vector field applicable to our equations, we need to "prolong" (187) or perform a "jet prolongation" to the jet space of the same order as our equations,n. Informally this just means extending the definition (187) to the same order as the equations being studied. The general formula for the prolongation to the nth jet bundle is

$$\mathrm{pr}^{(n)}v = v + \phi^J_\alpha \frac{\partial}{\partial u^\alpha_J}\,, \tag{188}$$

where the coefficients ϕ^J_α are given by

$$\phi^J_\alpha(x, u^{(n)} = D_J(\phi_\alpha - \xi^i u^\alpha_i) + \xi^i u^\alpha_{J,i}. \tag{189}$$

We can now begin to apply this to our equation of interest, the Kuramoto-Sivashinsky equation, (**??**). Starting with the prolongation of the general vector field, We need the fourth prolongation which seems like a lot of work (there is a coefficient for every combination of partial derivatives, and each higher order coefficient becomes more involved due to increased numbers of differentiation operations). Luckily, we already know that we are going to apply the vector field to the Kuramoto-Sivashinsky equation such that instead of calculating all $2 + 4 + 8 + 16$ jet prolongation coefficients (all combinations of t, x derivatives of order one, two, three and four) we only need the coefficients which accompany vectors $\frac{\partial}{\partial u_J}$ which appear in the Kuramoto-Sivashinsky equation. Namely, $\{\phi^J\} = \{\phi^t, \phi^x, \phi^{xx}, \phi^{xxxx}\}$, which in turn creates the vector field specific to the Kuramoto-Sivashinsky equation

$$\begin{aligned}\mathrm{pr}\, v^{(4)} &= \epsilon(x,t,u)\frac{\partial}{\partial x} + \tau(x,t,u)\frac{\partial}{\partial t} + \phi(x,t,u)\frac{\partial}{\partial u} + \phi^t(x,t,u^{(1)})\frac{\partial}{\partial u_t} \\ &+ \phi^x(x,t,u^{(1)})\frac{\partial}{\partial u_x} + \phi^{xx}(x,t,u^{(2)})\frac{\partial}{\partial u_{xx}} + \phi^{xxxx}(x,t,u^{(4)})\frac{\partial}{\partial u_{xxxx}}\,.\end{aligned} \tag{190}$$

All other higher order terms will annihilate when acting on the Kuramoto-Sivashinsky equation. Note that the higher the "order" of the coefficient, the higher the order of the jet bundle that the coefficients depend on. This can be seen by the definition of the coefficients (189), the higher the "order" of the coefficients, the more derivatives are taken to define it. Now that we have the general form of the vector field we can begin to derive the *infinitesimal generators* which span the Lie algebra. To accomplish this, we will derive the *determining equations* which are produced by applying (187) to the system of differential equations and equating to zero, that is

$$\mathrm{pr}\, v^{(4)}(G(u^{(\alpha)}(x,t), u^{(\alpha)}_{(1)}(x,t), \ldots, u^{(\alpha)}_{(n)}(x,t))) = 0\,. \tag{191}$$

Performing this operation yields

$$\phi^t + \phi^{xx} + \phi^{xxxx} + u\phi^x + \phi u_x = 0\,. \tag{192}$$

We finally are forced to derive the coefficients ϕ^J and to include as many details as possible we will write the exact formulas needed to derive them as well as the long form expressions that they are equal to.

$$\begin{aligned}
\phi^t &= D_t(\phi(x,t,u) - \epsilon(x,t,u)u_x - \tau(x,t,u)u_t) + \tau(x,t,u)u_t t + \epsilon(x,t,u)u_x t \\
\phi^x &= D_x(\phi(x,t,u) - \epsilon(x,t,u)u_x - \tau(x,t,u)u_t) + \tau(x,t,u)u_t x + \epsilon(x,t,u)u_x x \\
\phi^{xx} &= D_x^2(\phi(x,t,u) - \epsilon(x,t,u)u_x - \tau(x,t,u)u_t) + \tau(x,t,u)u_{txx} + \epsilon(x,t,u)u_{xxx} \\
\phi^{xxxx} &= D_x^4(\phi(x,t,u) - \epsilon(x,t,u)u_x - \tau(x,t,u)u_t) + \tau(x,t,u)u_{txxx} + \epsilon(x,t,u)u_{xxxx}
\end{aligned} \tag{193}$$

where D_t and D_x represent *total differentiation* operators,

$$D_i = \frac{\partial}{\partial x^i} + u_i \frac{\partial}{\partial u} + u_{ii} \frac{\partial}{\partial u_i} + \dots \tag{194}$$

the long form expressions from each of these are

$$\phi^t = u_t^2 \left(-\tau_u\right) - \tau_t u_t - u_t u_x \epsilon_u - \epsilon_t u_x + u_t \phi_u + \phi_t \tag{195}$$

$$\phi^x = -u_t \tau_u u_x - u_t \tau_x + u_x^2 \left(-\epsilon_u\right) - u_x \epsilon_x + u_x \phi_u + \phi_x \tag{196}$$

$$\begin{aligned}
\phi^{\mathrm{xx}} &= -u_t u_x^2 \tau_{\mathrm{uu}} - 2u_t u_x \tau_{\mathrm{xu}} - u_t \tau_u u_{\mathrm{xx}} \\
&- u_t \tau_{\mathrm{xx}} + u_x^3 \left(-\epsilon_{\mathrm{uu}}\right) + u_x^2 \phi_{\mathrm{uu}} \\
&- 2\tau_u u_x u_{\mathrm{xt}} - 2u_{\mathrm{xt}} \tau_x - 2u_x^2 \epsilon_{\mathrm{xu}} \\
&+ 2u_x \phi_{\mathrm{xu}} - 3u_x u_{\mathrm{xx}} \epsilon_u - u_x \epsilon_{\mathrm{xx}} \\
&- 2u_{\mathrm{xx}} \epsilon_x + u_{\mathrm{xx}} \phi_u + \phi_{\mathrm{xx}}
\end{aligned} \tag{197}$$

$$\begin{aligned}
\phi^{\mathrm{xxxx}} &= -4u_t u_x u_{\mathrm{xxx}} \tau_{\mathrm{uu}} - 3u_t u_{\mathrm{xx}}^2 \tau_{\mathrm{uu}} - 6u_t u_x^2 u_{\mathrm{xx}} \tau_{\mathrm{uuu}} - u_t u_x^4 \tau_{\mathrm{uuuu}} \\
&- 12u_t u_x u_{\mathrm{xx}} \tau_{\mathrm{xuu}} - 4u_t u_x^3 \tau_{\mathrm{xuuu}} - 6u_t u_x^2 \tau_{\mathrm{xxuu}} - 4u_t u_x \tau_{\mathrm{xxxu}} - 4u_t u_{\mathrm{xxx}} \tau_{\mathrm{xu}} \\
&- 6u_t u_{\mathrm{xx}} \tau_{\mathrm{xxu}} - u_t \tau_u u_{\mathrm{xxxx}} - u_t \tau_{\mathrm{xxxx}} - 12u_x u_{\mathrm{xt}} u_{\mathrm{xx}} \tau_{\mathrm{uu}} \\
&- 15u_x u_{\mathrm{xx}}^2 \epsilon_{\mathrm{uu}} - 6u_x^2 u_{\mathrm{xxt}} \tau_{\mathrm{uu}} - 10u_x^2 u_{\mathrm{xxx}} \epsilon_{\mathrm{uu}} + 4u_x u_{\mathrm{xxx}} \phi_{\mathrm{uu}} \\
&+ 3u_{\mathrm{xx}}^2 \phi_{\mathrm{uu}} - 4u_x^3 u_{\mathrm{xt}} \tau_{\mathrm{uuu}} - 10u_x^3 u_{\mathrm{xx}} \epsilon_{\mathrm{uuu}} + 6u_x^2 u_{\mathrm{xx}} \phi_{\mathrm{uuu}} \\
&+ u_x^5 \left(-\epsilon_{\mathrm{uuuu}}\right) + u_x^4 \phi_{\mathrm{uuuu}} - 12u_x^2 u_{\mathrm{xt}} \tau_{\mathrm{xuu}} - 12u_x u_{\mathrm{xt}} \tau_{\mathrm{xxu}} \\
&- 12u_x u_{\mathrm{xxt}} \tau_{\mathrm{xu}} - 16u_x u_{\mathrm{xxx}} \epsilon_{\mathrm{xu}} - 24u_x^2 u_{\mathrm{xx}} \epsilon_{\mathrm{xuu}} + 12u_x u_{\mathrm{xx}} \phi_{\mathrm{xuu}} \\
&- 4u_x^4 \epsilon_{\mathrm{xuuu}} + 4u_x^3 \phi_{\mathrm{xuuu}} - 18u_x u_{\mathrm{xx}} \epsilon_{\mathrm{xxu}} - 6u_x^3 \epsilon_{\mathrm{xxuu}} + 6u_x^2 \phi_{\mathrm{xxuu}} \\
&- 4\tau_u u_x u_{\mathrm{xxxt}} - 4u_{\mathrm{xxxt}} \tau_x - 4u_x^2 \epsilon_{\mathrm{xxxu}} + 4u_x \phi_{\mathrm{xxxu}} - 5u_x u_{\mathrm{xxxx}} \epsilon_u - u_x \epsilon_{\mathrm{xxxx}} \\
&- 4u_{\mathrm{xxxx}} \epsilon_x - 12u_{\mathrm{xt}} u_{\mathrm{xx}} \tau_{\mathrm{xu}} - 4\tau_u u_{\mathrm{xt}} u_{\mathrm{xxx}} - 4u_{\mathrm{xt}} \tau_{\mathrm{xxx}} \\
&- 12u_{\mathrm{xx}}^2 \epsilon_{\mathrm{xu}} + 4u_{\mathrm{xxx}} \phi_{\mathrm{xu}} - 6\tau_u u_{\mathrm{xx}} u_{\mathrm{xxt}} - 6u_{\mathrm{xxt}} \tau_{\mathrm{xx}} \\
&+ 6u_{\mathrm{xx}} \phi_{\mathrm{xxu}} - 10u_{\mathrm{xx}} u_{\mathrm{xxx}} \epsilon_u - 6u_{\mathrm{xxx}} \epsilon_{\mathrm{xx}} - 4u_{\mathrm{xx}} \epsilon_{\mathrm{xxx}} \\
&+ u_{\mathrm{xxxx}} \phi_u + \phi_{\mathrm{xxxx}}
\end{aligned} \tag{198}$$

upon substitution into (192) we can separate the terms by coefficients of monomials which gives us the determining equations as previously mentioned

$$
\begin{gathered}
\phi_t + \phi_{\mathrm{xx}} + \phi_{\mathrm{xxxx}} = 0 \\
-4\tau_x = 0 \\
-6\tau_{\mathrm{xx}} = 0 \\
-2\tau_x - 4\tau_{\mathrm{xxx}} = 0 \\
-4\epsilon_x + \tau_t + \tau_{\mathrm{xx}} + \tau_{\mathrm{xxxx}} = 0 \\
4\phi_{\mathrm{xu}} - 6\epsilon_{\mathrm{xx}} = 0 \\
-4\tau_u = 0 \\
4\tau_{\mathrm{xu}} = 0 \\
-2\epsilon_x - 4\epsilon_{\mathrm{xxx}} + \tau_t + \tau_{\mathrm{xx}} + \tau_{\mathrm{xxxx}} + 6\phi_{\mathrm{xxu}} = 0 \\
-6\tau_u = 0 \\
-12\tau_{\mathrm{xu}} = 0 \\
6\tau_{\mathrm{xxu}} = 0 \\
4\tau_{\mathrm{xu}} - 10\epsilon_u = 0 \\
-12\epsilon_{\mathrm{xu}} + 6\tau_{\mathrm{xxu}} + 3\phi_{\mathrm{uu}} = 0 \\
3\tau_{\mathrm{uu}} = 0 \\
3\tau_{\mathrm{uu}} = 0 \\
\phi - \epsilon_t - \epsilon_{\mathrm{xx}} - \epsilon_{\mathrm{xxxx}} + 2\phi_{\mathrm{xu}} + 4\phi_{\mathrm{xxxu}} = 0 \\
-4\tau_u = 0 \\
-12\tau_{\mathrm{xu}} = 0 \\
-2\tau_u - 12\tau_{\mathrm{xxu}} = 0 \\
-4\epsilon_u + 2\tau_{\mathrm{xu}} + 4\tau_{\mathrm{xxxu}} = 0 \\
4\phi_{\mathrm{uu}} - 16\epsilon_{\mathrm{xu}} = 0 \\
4\tau_{\mathrm{uu}} = 0 \\
-2\epsilon_u - 18\epsilon_{\mathrm{xxu}} + 2\tau_{\mathrm{xu}} + 4\tau_{\mathrm{xxxu}} + 12\phi_{\mathrm{xuu}} = 0 \\
-12\tau_{\mathrm{uu}} = 0 \\
12\tau_{\mathrm{xuu}} = 0 \\
4\tau_{\mathrm{uu}} = 0 \\
12\tau_{\mathrm{xuu}} - 15\epsilon_{\mathrm{uu}} = 0 \\
-2\epsilon_{\mathrm{xu}} - 4\epsilon_{\mathrm{xxxu}} + \phi_{\mathrm{uu}} + 6\phi_{\mathrm{xxuu}} = 0 \\
-6\tau_{\mathrm{uu}} = 0 \\
-12\tau_{\mathrm{xuu}} = 0
\end{gathered}
\tag{199}
$$

$$
\begin{gathered}
\tau_{\mathrm{uu}}+6 \tau_{\mathrm{xxuu}}=0 \\
-10 \epsilon_{\mathrm{uu}}=0 \\
-24 \epsilon_{\mathrm{xuu}}+\tau_{\mathrm{uu}}+6 \tau_{\mathrm{xxuu}}+6 \phi_{\mathrm{uuu}}=0 \\
6 \tau_{\mathrm{uuu}}=0 \\
6 \tau_{\mathrm{uuu}}=0 \\
-\epsilon_{\mathrm{uu}}-6 \epsilon_{\mathrm{xxuu}}+4 \phi_{\mathrm{xuuu}}=0 \\
-4 \tau_{\mathrm{uuu}}=0 \\
4 \tau_{\mathrm{xuuu}}=0 \\
4 \tau_{\mathrm{xuuu}}-10 \epsilon_{\mathrm{uuu}}=0 \\
\phi_{\mathrm{uuuu}}-4 \epsilon_{\mathrm{xuuu}}=0 \\
\tau_{\mathrm{uuuu}}=0 \\
\tau_{\mathrm{uuuu}}=0 \\
-\epsilon_{\mathrm{uuuu}}=0 \\
\phi_x=0 \\
\tau_x=0 \\
\tau_x=0 \\
-\epsilon_x+\tau_t+\tau_{\mathrm{xx}}+\tau_{\mathrm{xxxx}}=0 \\
4 \tau_{\mathrm{xu}}=0 \\
6 \tau_{\mathrm{xxu}}=0 \\
3 \tau_{\mathrm{uu}}=0 \\
2 \tau_{\mathrm{xu}}+4 \tau_{\mathrm{xxxu}}=0 \\
4 \tau_{\mathrm{uu}}=0 \\
12 \tau_{\mathrm{xuu}}=0 \\
\tau_{\mathrm{uu}}+6 \tau_{\mathrm{xxuu}}=0 \\
6 \tau_{\mathrm{uuu}}=0 \\
4 \tau_{\mathrm{xuuu}}=0 \\
\tau_{\mathrm{uuuu}}=0 \\
\tau_x=0
\end{gathered}
$$

While initially intimidating, these equations can be solved by noticing the lower order equations such as $\tau_x = \tau_u = 0$ which means that τ can only be a function of t. Following this reasoning we find that in fact

$$
\begin{aligned}
\tau(x,t,u) &= \tau = c_1 \\
\epsilon(x,t,u) &= \epsilon(t) = c_3 t + c_1 \\
\phi(x,t,u) &= \phi = c_3 \,,
\end{aligned} \tag{200}
$$

such that the Lie algebra of infinitesimal symmetries is spanned by

$$
\begin{aligned}
v_1 &= \partial_x \\
v_2 &= \partial_t \\
v_3 &= t\partial_x + \partial_u \,,
\end{aligned} \tag{201}
$$

which are the generators of space and time translations, and Galilean transformations. This is not surprising, as these symmetries have been previously described [9]. The reason why this calculation was pursued in the first place was to see if there were any "hidden" continuous symmetries afforded by a spatiotemporal formulation that were not present when

the problem was viewed as a dynamical system. This is true for *discrete symmetries*, but unfortunately not so for continuous symmetries. To carry the calculation through to finality we need to know the prolongations of (201) and their extensions to the adjoint variables present in (182), as the Lie algebra needs to be extended to account for both Euler-Lagrange equations.

The prolongations of (201) result in

$$\begin{aligned} \mathrm{pr}\, v_1 &= y_1 = \partial_x \\ \mathrm{pr}\, v_2 &= y_2 = \partial_t \\ \mathrm{pr}\, v_3 &= y_3 = \partial_x + \partial_u - u_x \partial_{u_t}\,. \end{aligned} \tag{202}$$

We can now derive the extended versions of (202) such that we can apply them to the formal Lagrangian (180). Once again we deploy the machinery of Ibragimov to extend (202) to the adjoint variables. Unfortunately it seems that the symmetries were too simple to actually have extensions to the adjoint variables, but we can still go forward with the conservation law calculations regardless. Both Ibragimov [21] and Olver [50] work through the proof that there is a conserved vector (as Ibragimov names it) such that its divergence provides a conservation law (technically infinite number of conservation laws because they are equations involving PDE solutions). The components of the conserved vector (one for each independent variable) are given by

$$\begin{aligned} C^i &= \xi^i \mathcal{L} + W^\alpha [\frac{\partial \mathcal{L}}{\partial u^\alpha_i} - D_j \frac{\partial \mathcal{L}}{\partial u^\alpha_{ij}} + D_j D_k \frac{\partial \mathcal{L}}{\partial u^\alpha_{ijk}} - + D_j D_k D_l \frac{\partial \mathcal{L}}{\partial u^\alpha_{ijkl}}] \\ &+ D_j(W^\alpha)[\frac{\partial \mathcal{L}}{\partial u^\alpha_{ij}} - D_k \frac{\partial \mathcal{L}}{\partial u^\alpha_{ijk}} + D_k D_l \frac{\partial \mathcal{L}}{\partial u^\alpha_{ijkl}}] \\ &+ D_j D_k(W^\alpha)[\frac{\partial \mathcal{L}}{\partial u^\alpha_{ijk}} - D_l \frac{\partial \mathcal{L}}{\partial u^\alpha_{ijkl}} + D_k D_j \frac{\partial \mathcal{L}}{\partial u^\alpha_{ikj}} - + D_k D_j D_l \frac{\partial \mathcal{L}}{\partial u^\alpha_{ikjl}}] \\ &+ D_j D_k D_l(W^\alpha)[\frac{\partial \mathcal{L}}{\partial u^\alpha_{ijkl}}]\,, \end{aligned} \tag{203}$$

where the equation has been extended to include all possible non-zero terms in the context of the Kuramoto-Sivashinsky equation, W^α is shorthand for $\phi^\alpha + \xi^i u^\alpha_i$. Applying this to our generators yields one unique conservation law which we shall now detail.

For the Galilean transformation generator $\mathrm{pr}\, v_3 = t\partial_x + \partial_u - u_x \partial_{u_t}$ the components equal

$$\begin{aligned} C^x &= t * \mathcal{L} + W[\frac{\partial \mathcal{L}}{\partial u_x} - D_x \frac{\partial \mathcal{L}}{\partial u_{xx}} - D_x^3 \frac{\partial \mathcal{L}}{\partial u_{xxxx}}] \\ &+ D_x(W)[\frac{\partial \mathcal{L}}{\partial u_{xx}} + D_x^2 \frac{\partial \mathcal{L}}{\partial u_{xxxx}}] \\ &+ D_x^2(W)[-D_x \frac{\partial \mathcal{L}}{\partial u_{xxxx}}] \\ &+ D_x^3(W)[\frac{\partial \mathcal{L}}{\partial u_{xxxx}}] \\ C^t &= 0 * \mathcal{L} + (1 - t u_x)[\frac{\partial \mathcal{L}}{\partial u_t}]\,. \end{aligned} \tag{204}$$

Both expressions simplify to [8] [9]

$$\begin{aligned} C^x &= t(u_t v + u_x v_x + u_x v_{xxx} + u_{xxx} v_x + u_{xx} v_{xx}) + uv - v_x - v_{xxx} \\ C^t &= (1 - t u_x) v \,, \end{aligned} \tag{205}$$

such that the conservation law is given by the divergence

$$\begin{aligned} D_x(C^x) + D_t(C^t) &= 0 \\ &= v_t - v u_x - u_x v_t t + u v_x + v u_x - v_{xx} - v_{xxxx} \\ &\quad + t(v_x(u_t + u_{xx} + u_{xxxx}) + u_x(v_{xx} + v_{xxxx}) + 2D_x(u_{xx} v_{xx})) \\ &= t(v_x(u_t + u_{xx} + u_{xxxx}) + u_x(-v_t + v_{xx} + v_{xxxx}) + 2D_x(u_{xx} v_{xx})) \\ &= 2D_x(u_{xx} v_{xx}) \,. \end{aligned} \tag{206}$$

This analysis is only relevant if there are non-trivial solutions to the adjoint equation (182), because otherwise one does not know how to evaluate (206). As we claimed previously, the only solutions to (182) are constant in nature; implying that the conserved vector (206) is unfortunately a trivial conservation law. There are a number of reasons why we believe this analysis fails to yield anything useful, the most obvious being that the Lie algebra of infinitesimal symmetries is too simple. [10]

[8] 2019-09-11 Predrag: not sure about v_{tt} term...

[9] 2019-09-12 Matt: Accidentally wrote down the divergence $D_t C^t$ instead of C^t

[10] 2019-09-11 Predrag: I am thinking of moving everything from (180) to here into a Chapter of your thesis, remarking here briefly that Ibragimov methods seem to have not worked for you?

APPENDIX C

ORBITHUNTER

One of the largest hindrances to collaboration that I have experienced is the fact that everyone has their own custom numerical codes. The lack of consistency in programming language, conventions, notation, documentation, etc., dramatically slow down the scientific process. Not only do these codes tend to be esoteric and quite confusing to adopt, I have found they are also very difficult to modify or make additions to. My solution to this was to write a programming package, "orbithunter", in the Python programming language. Python was chosen primarily because I believe that it is the friendliest language. With improvements in the scientific computing packages such as NumPy and SciPy, numerical computations are no long prohibitively slow in Python when compared to, for example, C++. Also, with the usage of jupyter notebooks, computations become nimble and efficient when provided with an API with similar qualities. Orbithunter serves as a high-level API to perform spatiotemporal calculations described in this thesis for the Kuramoto-Sivashinsky equation. The beauty and my proudest achievement is that (barring some inevitable bugs that come with expanding any programming package) the utilities that enable users to clip, glue, etc. extend to any N-dimensional field equation. Not only do all of the methods generalize to other equations, it is also easy to write the necessary code to make these additions, if the framework is followed. This framework consists of the following: in order to be able to make use of all orbithunter utilities, one must write a module which creates a subclass of the orbithunter class Orbit. Namely, this entails writing about thirty or so functions which are able to compute the relevant numerical quantities such as the governing equations , the matrix-vector products , etc. The functions which must be written at very clearly documented as methods of the parent Orbit class. The bonus is that there is no assumption of the basis being used; all transform methods are custom to the subclass being written. Therefore, even if you disagree on, for instance, using a spatiotemporal Fourier mode basis, you can still utilize the package and its utilities, although I cannot guarantee that the methods will not lose their meaning if one strays too far from the spatiotemporal path.

The API itself is designed after some of the most popular Python packages in scientific computing: Pandas, NumPy, SciPy. This enables the user to apply high level functions and ideas to perform complex calculations as opposed to digging through the details of how the entire package is setup.

In addition to the increased user friendliness that orbithunter presents, it is also faster than my previous research code and presents the user with a myriad of options for the purpose of numerical optimization.

Introduction In order to expand the user base and influence of our spatiotemporal formulation I converted my research code into a Python package that can be distributed as an open source project on Github. The original code fell into the classic pitfall of being only well known to the author. The newer version of the code is incredibly user friendly and tries to model itself after very popular python packages

The following package is a result of the complete rewrite of my research code to maximize user friendliness and computational speed. Because this is a complete refactoring of my research code there are likely still a few bugs. It also should be noted that this is a completely independent endeavor of somebody with a Physics background; there might be some details or choices in the code that are suboptimal. This is why this code will be available as an open source github repository.

Main benefits of using this package. Plugging in your own Torus class. It has all been vectorized and so it's pretty fast. It's incredibly user-friendly. It has many different numerical methods available, although there are a number that can be used only if constraints are included as they require square linear systems (such as SciPy's implementation of GMRES).

The other main goal of this project was to make the code modular; obviously the code has been written around the Kuramoto-Sivashinsky equation but it can be tailored to encompass any equation and any dimensional torus. The main requirements to interface with the current package are the following:

A function which returns a class instance in terms of real-valued spectral basis; it doesn't necessarily have to be Fourier, but the function is required to be called this for compatibility sake. This might be changed upon release just to prevent confusion; it's annoying but it would only take the renaming of functions. Note that for a system with $D+1$ dimensional space-time, this must take a physical field to D dimensional spectral basis, even though it acts on the entire $D+1$ dimensional torus. This was done because it is natural for those who simulate fluid turbulence; most codes are spectral in nature and so these transforms are already implemented.

A function which returns a class instance in terms of real-valued spatiotemporal spectral basis. The requirements follow the same as the spatial transform.

A function which calculates the equations in the spatiotemporal Fourier coefficients

References

[1] V. I. Arnol'd and A. Avez, *Ergodic Problems of Classical Mechanics* (Addison-Wesley, Redwood City, 1989).

[2] S. V. Bolotin and D. V. Treschev, "Hill's formula", Russ. Math. Surv. **65**, 191 (2010).

[3] A. Borzi and V. Schulz, *Computational Optimization of Systems Governed by Partial Differential Equations* (SIAM, Philadelphia, 2011).

[4] A. Bouras and V. Frayssé, "Inexact matrix-vector products in Krylov methods for solving linear systems: A relaxation strategy", SIAM J. Matrix Anal. Appl. **26**, 660–678 (2005).

[5] J. P. Boyd, *Chebyshev and Fourier Spectral Methods*, 2nd ed. (Dover, New York, 2000).

[6] S. Boyd and L. Vandenberghe, *Convex Optimization* (Cambridge Univ. Press, Cambridge, 2004).

[7] P. N. Brown and Y. Saad, "Hybrid Krylov methods for nonlinear systems of equations", SIAM J. Sci. Statist. Comput. **11**, 450–481 (1990).

[8] P. N. Brown, "A local convergence theory for combined inexact-Newton/finite-difference projection methods", SIAM J. Numer. Anal. (1987).

[9] N. B. Budanur, P. Cvitanović, R. L. Davidchack, and E. Siminos, "Reduction of the SO(2) symmetry for spatially extended dynamical systems", Phys. Rev. Lett. **114**, 084102 (2015).

[10] C. Canuto, M. Y. Hussaini, A. Quateroni, and T. A. Zhang, *Spectral methods in fluid dynamics* (Springer, New York, 1988).

[11] T. F. Chan and K. R. Jackson, "Nonlinearly preconditioned Krylov subspace methods for discrete Newton algorithms", SIAM J. Sci. Statist. Comput. **5**, 533–542 (1984).

[12] K. T. Chu, "A direct matrix method for computing analytical Jacobians of discretized nonlinear integro-differential equations", J. Comput. Phys. **228**, 5526–5538 (2009).

[13] P. Cvitanović and Y. Lan, Turbulent fields and their recurrences, in Correlations and Fluctuations in QCD : Proceedings of 10. International Workshop on Multiparticle Production, edited by N. Antoniou (2003), pp. 313–325.

[14] P. Cvitanović, R. Artuso, R. Mainieri, G. Tanner, and G. Vattay, *Chaos: Classical and Quantum* (Niels Bohr Inst., Copenhagen, 2018).

[15] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations* (SIAM, Philadelphia, 1996).

[16] J. van den Eshof and G. L. G. Sleijpen, "Inexact Krylov subspace methods for linear systems", SIAM J. Matrix Anal. Appl. **26**, 125–153 (2004).

[17] D. G. Fox and S. A. Orszag, "Pseudospectral approximation to two-dimensional turbulence", J. Comput. Phys. **11**, 612–619 (1973).

[18] G. Giacomelli, S. Lepri, and A. Politi, "Statistical properties of bidimensional patterns generated from delayed and extended maps", Phys. Rev. E **51**, 3939–3944 (1995).

[19] M. Gunzburger, *Perspectives in Flow Control and Optimization* (SIAM, 2002).

[20] N. H. Ibragimov, "Integrating factors, adjoint equations and Lagrangians", J. Math. Anal. Appl. **318**, 742–757 (2006).

[21] N. H. Ibragimov, "A new conservation theorem", J. Math. Anal. Appl. **333**, 311–328 (2007).

[22] N. H. Ibragimov, "Quasi-self-adjoint differential equations", Arch. ALGA **4**, 55–60 (2007).

[23] N. H. Ibragimov, "Nonlinear self-adjointness and conservation laws", J. Phys. A **44**, 432002 (2011).

[24] N. H. Ibragimov, "Nonlinear self-adjointness in constructing conservation laws", Arch. ALGA **7** (2011).

[25] N. H. Ibragimov, "Conservation laws and non-invariant solutions of anisotropic wave equations with a source", Nonlinear Anal. Real World Appl. **40**, 82–94 (2018).

[26] N. H. Ibragimov and T. Kolsrud, "Lagrangian approach to evolution equations: symmetries and conservation laws", Nonlin. Dyn. **36**, 29–40 (2004).

[27] A.-K. Kassam and L. N. Trefethen, "Fourth-order time-stepping for stiff PDEs", SIAM J. Sci. Comput. **26**, 1214–1233 (2005).

[28] J. P. Keating, "Asymptotic properties of the periodic orbits of the cat maps", Nonlinearity **4**, 277 (1991).

[29] J. P. Keating, "The cat maps: quantum mechanics and classical motion", Nonlinearity **4**, 309–341 (1991).

[30] R. M. Kerr, "Higher-order derivative correlations and the alignment of small-scale structure in isotropic numerical turbulence", J. Fluid. Mech. **153**, 31–58 (1985).

[31] D. Knoll and D. Keyes, "Jacobian-free Newton-Krylov methods: a survey of approaches and applications", J. Comput. Phys. **193**, 357–397 (2004).

[32] M. Kraus and O. Maj, "Variational integrators for nonvariational partial differential equations", Physica D **310**, 37–71 (2015).

[33] S. Krist and T. A. Zang, Numerical simulation of channel flow transition: Resolution requirements and the structure of hairpin vortices, tech. rep. (NASA, 1987).

[34] Y. Lan, Dynamical systems approach to $1-d$ spatiotemporal chaos – A cyclist's view, PhD thesis (School of Physics, Georgia Inst. of Technology, Atlanta, 2004).

[35] Y. Lan and P. Cvitanović, "Variational method for finding periodic orbits in a general flow", Phys. Rev. E **69**, 016217 (2004).

[36] Y. Lan, C. Chandre, and P. Cvitanović, "Variational method for locating invariant tori", Phys. Rev. E **74**, 046206 (2006).

[37] S. Lepri, A. Politi, and A. Torcini, "Chronotopic Lyapunov analysis. I. A detailed characterization of 1D systems", J. Stat. Phys. **82**, 1429–1452 (1996).

[38] S. Lepri, A. Politi, and A. Torcini, "Chronotopic Lyapunov analysis. II. Towards a unified approach", J. Stat. Phys. **88**, 31–45 (1997).

[39] J. Li and S. Tomsovic, "Exact relations between homoclinic and periodic orbit actions in chaotic systems", Phys. Rev. E **97**, 022216 (2017).

[40] V. López, P. Boyland, M. T. Heath, and R. D. Moser, "Relative periodic solutions of the complex Ginzburg-Landau equation", SIAM J. Appl. Dyn. Syst. **4**, 1042–1075 (2006).

[41] R. S. Mackay and J. D. Meiss, "Linear stability of periodic orbits in Lagrangian systems", Phys. Lett. A **98**, 92–94 (1983).

[42] S. MacKay, J. D. Meiss, and I. C. Percival, "Transport in Hamiltonian systems", Physica D **13**, 55–81 (1984).

[43] J. E. Marsden and M. West, "Discrete mechanics and variational integrators", Acta Numerica **10**, 357–514 (2001).

[44] J. Marsden, S. Pekarsky, S. Shkoller, and M. West, "Variational methods, multisymplectic geometry and continuum mechanics", J. Geom. Phys. **38**, 253–284 (2001).

[45] J. D. Meiss, "Symplectic maps, variational principles, and transport", Rev. Mod. Phys. **64**, 795–848 (1992).

[46] J. Meza, A modification to the GMRES method for ill-conditioned linear systems, tech. rep. (Sandia National Laboratories, 1995).

[47] F. Montigny-Rannou, "Effect of "aliasing" on spectral method solution of Navier-Stokes equations", Rech. Aerosp. **2**, 365–41 (1982).

[48] R. D. Moser, P. Moin, and L. A., "A spectral numerical method for the Navier-Stokes equations with applications to Taylor-Couette flow", J. Comput. Phys. **52**, 524–544 (1983).

[49] E. Noether, "Der Endlichkeitssatz der Invarianten endlicher Gruppen", Math. Ann. **77**, 89–92 (1915).

[50] P. J. Olver, *Applications of Lie Groups to Differential Equations* (Springer, New York, 1998).

[51] S. A. Orszag, "Computation of pseudospectral and spectral approximations", Stud. Appl. Math. **51**, 253–259 (1972).

[52] C. C. Paige and M. A. Saunders, "LSQR: An algorithm for sparse linear equations and sparse least squares", ACM Trans. Math. Softw. **8**, 43–71 (1982).

[53] I. Percival and F. Vivaldi, "A linear code for the sawtooth and cat maps", Physica D **27**, 373–386 (1987).

[54] I. Percival and F. Vivaldi, "Arithmetical properties of strongly chaotic motions", Physica D **25**, 105–130 (1987).

[55] A. Politi, A. Torcini, and S. Lepri, "Lyapunov exponents from node-counting arguments", J. Phys. IV **8**, 263 (1998).

[56] G. L. G. Sleijpen, J. van den Eshof, and M. B. van Gijzen, "Restarted gmres with inexact matrix–vector products", in *Lecture Notes in Computer Science*, edited by Z. Li, L. Vulkov, and J. Waśniewski (Springer, Berlin, 2005), pp. 494–502.

[57] M. Tabor, *Chaos and Integrability in Nonlinear Dynamics: An Introduction* (Wiley, New York, 1989).

[58] D. Viswanath, "Recurrent motions within plane Couette turbulence", J. Fluid Mech. **580**, 339–358 (2007).

[59] D. Viswanath, "The critical layer in pipe flow at high Reynolds number", Philos. Trans. Royal Soc. A **367**, 561–576 (2009).

[60] Q. Wang, R. Hu, and P. J. Blonigan, "Least Squares Shadowing sensitivity analysis of chaotic limit cycle oscillations", J. Comput. Phys. **267**, 210–224 (2014).

[61] L. Wei and Y. Wang, "Symmetry analysis, conserved quantities and applications to a dissipative DGH equation", J. Diff. Equ. **266**, 3189–3208 (2019).