



UNIVERSITY OF LEEDS

Modelling Unstable Kolmogorov Flow via Spectral Methods, Transition to 2D Turbulence

Alexander Robert Wan

Supervisor: Dr. Cédric Beaume

MATH5000M

**Submitted in accordance with the requirements for the degree
of MSc. Mathematics**

The University of Leeds

Faculty of Engineering and Physical Sciences

School of Mathematics

20/08/26

Project Summary

The project starts with the introduction of turbulence, where the sources [14, 17, 1] were introduced. The introduction then summarises the ideas used in Kolmogorov flow using [6, 10, 13, 2].

The Kolmogorov system was then defined and non-dimensionalised using standard techniques, the final system was represented by the vorticity equation.

Linear stability analysis was then carried out on the vorticity equation, using [10, 8] to validate results. The critical Reynolds number for unstable perturbation was found to be $Re_c = \sqrt{2}$. An eigenvalue problem for the growth of perturbations was also obtained, which was approximated through convergent analysis.

Using [3, 9, 16, 15, 4] spectral methods was motivated. Similarly using [3, 9, 5] the Adams-Bashforth and Adams-Moulton time stepping methods were motivated.

Next the spectral methods, and time stepping methods were applied to the Kolmogorov system, thus producing the iterative numerical scheme, and advection methods. The simulation methods were also motivated here, discussing grid size and optimisation.

The Results section was then included, where [11] provided a definition for stability on the smallest domain size. This section showed the results of running the Kolmogorov solver, and illustrated the transition into turbulence of the system.

Finally the project was discussed, including achievement and limitations.

AI Integrity

I confirm that I did not make use of generative AI during the completion of this project.

Abstract

Through the use of spectral methods in conjunction with the Adams-Bashforth, and Adams-Moulton time stepping schemes, a numerical scheme for solving the Kolmogorov system iteratively was derived. Additionally through stability analysis a critical Reynolds number $Re_c = \sqrt{2}$, for which the steady state Kolmogorov flow becomes unstable, is derived. By utilising the Kolmogorov system solver over different domain sizes transition to turbulence is observed.

Acknowledgements

Before progressing onto the mathematics, I would first like to express my gratitude to my supervisor Cédric Beaume, whose guidance and support I am thankful for. This project has been a really wonderful experience.

I would additionally like to thank my Mum, and Grandma, for supporting me throughout my MSc year, it would not have been possible without them.

Finally I will thank my girlfriend Khushi for her reassurance throughout this project.

Contents

1	Kolmogorov Flow	1
1.1	Introduction	1
1.2	The Kolmogorov System	3
1.3	Non-dimensionalisation of the Kolmogorov System	3
1.4	Vorticity Formulation	6
2	Bifurcations from a Steady Flow	8
2.1	Linearised Vorticity	8
2.2	Normal Mode Solutions	10
2.3	Convergence of the Bifurcation Curve	12
2.4	Solutions for the Matrix $\mathbf{M}_1$	14
2.5	Critical Reynolds Convergence	15
3	Spectral Methods	19
3.1	Spectral Differentiation	19
3.2	The Fast Fourier Transform	21
3.3	Spectral Accuracy	21
3.4	The 2D DFT	22
3.5	Spectral Aliasing	23
4	Time Stepping and Numerical Stability	26
4.1	Numerical Stability	26
4.2	Non-Linear Term	28
4.3	Time Derivative	30

5	Simulation Methods for K-Flow	31
5.1	Kolmogorov Spectral Scheme	31
5.2	The Non-Linear Advection Term	33
5.3	Simulation Approach	34
6	Results	37
6.1	Diagnostic Quantities	37
6.2	The Trivial Domain (1,1)	38
6.3	Simulation on (2,1)	38
6.3.1	Instability on (2,1) with $Re = 25$	42
6.3.2	Instability on (2,1) with $Re = 100$	43
6.3.3	Flow Reversal on (2,1)	44
6.4	Simulation on (2,2)	45
6.4.1	Instability and Turbulence on (2,2) for $Re = 100$	46
6.5	Simulation on (4,4)	48
6.5.1	Instability on (4,4) with $Re = 2.5$	48
6.5.2	Instability on (4,4) with $Re = 5$	50
6.5.3	Instability and Turbulence on (4,4) with $Re = 25$	51
6.6	Simulation on (16,16)	54
6.6.1	Instability on (16,16) with $Re = 2.5$	54
6.6.2	Instability and Turbulence on (16,16) with $Re = 100$	56
6.7	Conclusions on Transition to Turbulence	57
7	Discussion	59
	References	60
A	Appendix	62
A.1	Stability Analysis	62
A.1.1	Function: <code>growth()</code>	62
A.1.2	Function: <code>growth_mesh()</code>	63
A.1.3	Re_c Convergence as $n \rightarrow \infty$	63
A.1.4	Re_c Convergence as $\Delta S \rightarrow 0$	64
A.1.5	Function: <code>point_growth()</code>	65

A.2	Spectral Methods	66
A.2.1	Spectral Accuracy in 1D	66
A.3	Time-Stepping in Python	67
A.3.1	Solving the Heat Equation	67
A.4	Kolmogorov Simulation	69
A.4.1	Function: <code>advection()</code>	69
A.4.2	Function: <code>aliasing_mask()</code>	69
A.4.3	Validating <code>advection()</code>	69
A.4.4	Function: <code>Kolmogorov_simulation()</code>	70
A.4.5	Validating Steady State for <code>Kolmogorov_simulation()</code>	72
A.5	Results	74
A.5.1	Mesh Convergence Study	74
A.5.2	Calculating $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$	75
A.5.3	Directional Energy Calculator	76
A.5.4	Line Plots for (2,1)	77
A.5.5	Contour Plots for (2,1)	79
A.5.6	Plots for (2,2)	80
A.5.7	Line Plots for (4,4)	82
A.5.8	Contour Plots for (4,4)	84
A.5.9	Plots for (16,16)	85

List of Figures

1.1	Turbulence illustrated on a vorticity field (Appendix A.5.9).	1
2.1	Bifurcation curve (2.31) for the matrix $\mathbf{M}_1$, plotted in Python.	15
2.2	Plot showing $ Re_c^n - Re_c^N $ against n , where $N = 40$, $M = P = 10$, $\Delta S = 0.01$, computed and plotted in Python (Appendix A.1.3).	17
2.3	Plot showing $ Re_c^h - Re_c^H $ against ΔS_h , where $N = 20$, $M = P = 10$, $H = 40$, computed and plotted in Python (Appendix A.1.4)	17
3.1	Equidistributed Γ periodic grid $x_j = j\Gamma/N$, $j = 0, 1, \dots, N-1$ with data values f_j and periodic boundary condition $x_0 = x_N$, reproduced from [9].	20
3.2	Scatter plot showing the Root Mean Squared Error (RMSE) of approximation (3.11), with respect to the analytical derivative (3.10), for different grid resolu- tions N . Results computed and plotted on a logarithmic RMSE axis in Python (Appendix A.2.1) using methods from [9].	22
3.3	Modes $\cos(6x)$ and $\cos(2x)$ plotted on the periodic domain $x \in [0, 2\pi]$, with the vertical dashed lines representing the grid $x_j = 2\pi j/N$, $j = 0, \dots, N-1$ for $N = 8$. 24	
4.1	Implicit and explicit solutions to the heat equation, evolved using finite differences in time, on a spatially periodic domain $x, t \in [0, 2\pi] \times [0, 5]$ discretised spatially and temporally. The system has an initial profile $u_0 = \exp(-\cos^2(x/2))$, and parameters $D = 1$, $\Delta t = 1.065 \times 10^{-2}$, $N = 30$, $\Gamma = 2\pi$. Results computed and graphed using Python (Appendix A.3.1).	28

5.1	Scatter plot showing the Root Mean Squared Error (RMSE) of approximation (5.15), for the test function $\psi = \exp(\cos x + \cos y)$, at different grid resolutions $N = N_x = N_y$. Results computed and plotted on a logarithmic RMSE axis in Python (Appendix A.4.3).	34
5.2	Root Mean Squared Error (RMSE) of Kolmogorov system velocity field with respect to the steady state field $\omega_s = -\cos(y)$, as the system evolves with time, plotted on a logged RMSE axis. Vorticity field evolved using “ <code>Kolmogorov_simulation()</code> ” with $N = 128, a_x = 1, a_y = 1, Re = 1, \Delta t = 0.1, t_{max} = 10$ with an initial <code>state=False</code> . Computed and plotted on a logged RMSE axis in python (Appendix A.4.5).	35
6.1	Plots showing $\tilde{\zeta}^t$ over time for different spatial and temporal meshes. Calculated and plotted using Python (Appendix A.5.1).	39
6.2	Plots showing $\tilde{\zeta}^t$ over time for different spatial and temporal meshes. Calculated and plotted using Python (Appendix A.5.1).	40
6.3	Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on $(2, 1)$, with $N = 128$ and $Re = 2.5$. Calculated in Python using <code>perturbation_energy.py</code> and plotted with code in Appendix A.5.4.	41
6.4	Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on $(2, 1)$, with $N = 128$ and $Re = 5$. Calculated in Python using <code>perturbation_energy.py</code> , and plotted with code in Appendix A.5.4.	42
6.5	Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on $(2, 1)$, with $N = 128$ and $Re = 25$. Calculated in Python using <code>perturbation_energy.py</code> , and plotted with code in Appendix A.5.4.	43
6.6	Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on $(2, 1)$, with $N = 128$ and $Re = 100$. Calculated in Python using <code>perturbation_energy.py</code> , and plotted with code in Appendix A.5.4.	44
6.7	Contourf plots of the vorticity field of $(2, 1)$, for $Re = 2.5, 5, 25, 100$ at $t = 1999$. Plotted in Python (Appendix A.5.5).	45
6.8	Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on $(2, 2)$, with $N = 128$ and $Re = 100$. Calculated in Python using <code>perturbation_energy.py</code> , and plotted with code in Appendix A.5.6.	46

6.9	Contour plots of the vorticity field of the Kolmogorov system at $t = 50, 75, 100, 1000$ for $Re = 100$ on $(2, 2)$, plotted using Python (Appendix A.5.6)	47
6.10	Plot showing $\tilde{\zeta}(t)$ on $(4, 4)$ with $N = 128$ and $Re = 2.5$ on a logged vertical axis. The black dashed line is a tangent with gradient 2λ , representing the expected growth rate (a) $\lambda = 0.035\dots$ indicative of the $k_x = 0.25$ mode, and (b) $\lambda = 0.038\dots$ indicative of the $k_x = 0.5$ mode. Calculated in Python using <code>perturbation_energy.py</code> , and plotted by code in Appendix A.5.7.	49
6.11	Mean absolute time derivative $\tilde{\Delta}(t)$ for $Re = 2.5$ on $(4, 4)$ with $N = 128$, evolved with time. Calculated in Python using <code>perturbation_energy.py</code> , and plotted by code in Appendix A.5.7.	49
6.12	Contourf plot of the vorticity field on $(4, 4)$ with $Re = 2.5$ at $t = 2000$. Plotted in Python (Appendix A.5.8).	50
6.13	Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on $(4, 4)$, with $N = 128$ and $Re = 5$. Calculated in Python using <code>perturbation_energy.py</code>	51
6.14	Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on $(4, 4)$, with $N = 128$ and $Re = 25$. Calculated in Python using <code>perturbation_energy.py</code>	52
6.15	Contour plots of the vorticity field of the Kolmogorov system at $t = 50, 60, 75, 200$ for $Re = 25$ on $(4, 4)$, plotted using Python (Appendix A.5.8).	53
6.16	Plot showing $\tilde{\Delta}(t)$ for the Kolmogorov system simulated on $(16, 16)$, with $N = 128$ and $Re = 2.5$, evolved with time. Results plotted on a logged vertical axis in Python (Appendix A.5.9).	54
6.17	Contour plots of the vorticity field of the Kolmogorov system at $t = 300, 1000, 1250, 1500$ for $Re = 2.5$ on $(16, 16)$, plotted using Python (Appendix A.5.9)	55
6.18	Plot showing $\tilde{\Delta}(t)$ for the Kolmogorov system simulated on $(16, 16)$, with $N = 128$ and $Re = 100$, evolved with time. Results plotted on a logged vertical axis in Python (Appendix A.5.9).	56
6.19	Contour plots of the vorticity field of the Kolmogorov system at $t = 40, 55, 75, 200$ for $Re = 100$ on $(16, 16)$, plotted using Python (Appendix A.5.9)	57

Chapter 1

Kolmogorov Flow

1.1 Introduction

Turbulence is a phenomenon defined by swirling motions in a fluid that occur irregularly in time and space [14]. The understanding of turbulent motion is paramount in a number of scientific fields, from oceanography, to engineering and astrophysics [14]. Figure 1.1 illustrates a turbulent vorticity field, the complexity of the flow is unmistakeable. How does this complex flow arise?

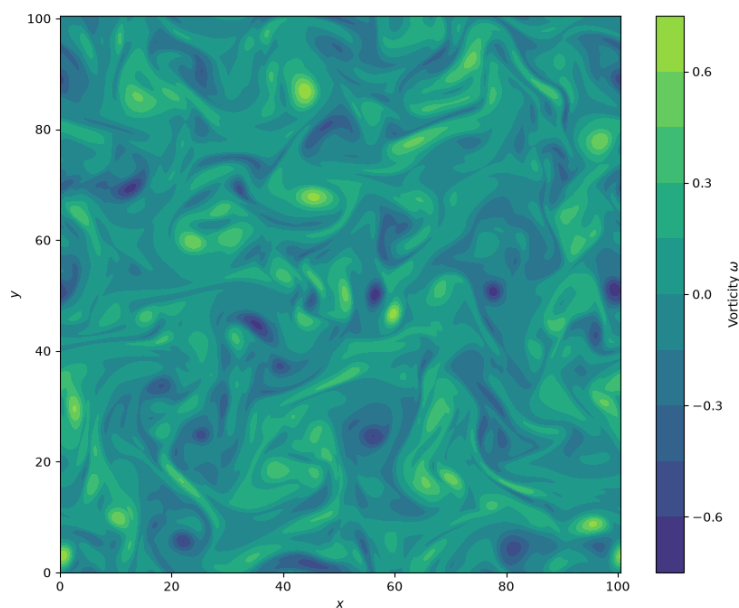


Figure 1.1: Turbulence illustrated on a vorticity field (Appendix A.5.9).

Turbulent eddies play an important role in maintaining the atmospheres global circulation, and synoptic scale weather systems [17]. This same phenomenon of turbulence also occurs on the surface of stars, in the formation of nebulae, as an efficient cooling mechanism in nuclear engineering, and even in the process by which you mix a cup of tea [1]. It is clear that the world around us is surrounded by turbulence.

During the late 1950s, A.N. Kolmogorov developed an interest in this study of turbulence. Much of his work became focused on the flow $u = \sin(y)$, which resulted from unidirectional sinusoidal forcing on a 2 dimensional, incompressible fluid [6]. This flow came to be known as **Kolmogorov flow**, and since its introduction it has been the subject of many studies, aiming to better understand instability and turbulence in shear flows [6].

Many of these studies rely on the use of simulation methods to model Kolmogorov flow, for example [10] who simulated Kolmogorov flow in 3 dimensions, utilising spectral methods. Numerous experimental studies into Kolmogorov flow have also been conducted, such as [2], where Kolmogorov flow was investigated on a cylindrical surface.

Kolmogorov flow, and turbulence, have also found applications outside of purely fluid dynamics research. For instance [13] investigated the use of turbulent Kolmogorov flow as a method of fast encryption, utilising pseudo random initial conditions, in conjunction with the chaos of the system to obtain computationally secure product ciphers.

In this paper we endeavour to solve the Kolmogorov system using simulation methods. We will start by discussing the Kolmogorov system, which applies a sinusoidal body force to a fluid, over an infinite domain. We will then discuss the stability of the Kolmogorov system to perturbations, and the critical Reynolds number for which instability can occur. Through a survey of spectral methods, and time stepping techniques we will motivate the final time stepping scheme that will allow us to simulate the Kolmogorov system. Finally we will discuss the results of simulating the Kolmogorov system over a range of domain shapes, with a focus on the transition of the system into turbulence.

1.2 The Kolmogorov System

The Kolmogorov system is obtained by adding the sinusoidal forcing term $\chi \sin(2\pi y/L_y) \mathbf{e}_x$ to the RHS of the Navier–Stokes equations:

$$\rho \frac{D\mathbf{u}}{Dt} = -\nabla p + \mu \nabla^2 \mathbf{u} + \chi \sin\left(\frac{2\pi y}{L_y}\right) \mathbf{e}_x \quad (1.1)$$

$$\nabla \cdot \mathbf{u} = 0 \quad (1.2)$$

where χ must be dimensionally consistent:

$$[\chi] = [\rho] \cdot L \cdot T^{-2} = M \cdot L^{-2} \cdot T^{-2}. \quad (1.3)$$

1.3 Non-dimensionalisation of the Kolmogorov System

The objective of this paper is to understand how changing parameters effects the stability of the Kolmogorov system, thus to gain optimal insights here it is beneficial to reduce the number of parameters in (1.1) and (1.2) by non-dimensionalisation. Ultimately we find that we can reduce the number of parameters from 4 to 1. The standard approach is applied here, expressing each variable in the system as a non-dimensional variant multiplied by some dimensional scaling quantity. We will start by setting $\mathbf{u} = U\mathbf{u}^*$, $\mathbf{x} = X\mathbf{x}^*$ in order to make (1.2) dimensionless:

$$\frac{1}{X} \nabla^* \cdot (U\mathbf{u}^*) = \frac{U}{X} \nabla^* \cdot \mathbf{u}^* = 0. \quad (1.4)$$

By dividing through U/X we obtain the non-dimensional Continuity equation, the form is identical to the dimensional Continuity equation as a result of equivalent scaling in x and y :

$$\nabla^* \cdot \mathbf{u}^* = 0. \quad (1.5)$$

We can now make the Navier–Stokes equation dimensionless, similarly to non-dimensionalising the Continuity equation we set the variables in (1.1) to $\mathbf{u} = U\mathbf{u}^*$, $\mathbf{x} = X\mathbf{x}^*$, $t = Xt^*/U$, $p =$

Pp^* , thus the Navier–Stokes equation becomes

$$\rho \frac{\partial(U\mathbf{u}^*)}{\partial(\frac{X}{U}t^*)} + U\mathbf{u}^* \cdot (\frac{1}{X}\nabla(U\mathbf{u}^*)) = -\frac{1}{X}\nabla^*(Pp^*) + \frac{\mu}{X^2}\nabla^{*2}(\mathbf{U}\mathbf{u})^* + \chi \sin\left(\frac{2\pi Xy^*}{L_y}\right)\mathbf{e}_x. \quad (1.6)$$

By application of the chain rule we can move the dimensional constants outside the derivatives:

$$\frac{\rho U^2}{X} \left(\frac{\partial \mathbf{u}^*}{\partial t^*} + \mathbf{u}^* \cdot \nabla^* \mathbf{u}^* \right) = -\frac{P}{X}\nabla^* p^* + \frac{\mu U}{X^2}\nabla^{*2} \mathbf{u}^* + \chi \sin\left(\frac{2\pi Xy^*}{L_y}\right)\mathbf{e}_x. \quad (1.7)$$

Multiplying by $X/\rho U^2$ makes each term in the equation dimensionless, thus simplifying the form:

$$\frac{\partial \mathbf{u}^*}{\partial t^*} + \mathbf{u}^* \cdot \nabla^* \mathbf{u}^* = -\frac{P}{\rho U^2}\nabla^* p^* + \frac{\mu}{\rho U X}\nabla^{*2} \mathbf{u}^* + \frac{\chi X}{\rho U^2} \sin\left(\frac{2\pi Xy^*}{L_y}\right)\mathbf{e}_x. \quad (1.8)$$

We can check that each term is dimensionless though dimensional analysis:

$$\left[\frac{\partial \mathbf{u}^*}{\partial t^*} \right] = 1 \quad (1.9)$$

$$[\mathbf{u}^* \cdot \nabla^* \mathbf{u}^*] = 1 \quad (1.10)$$

$$\left[\frac{P}{\rho U^2} \nabla^* p^* \right] = \frac{[P]}{[\rho][U^2]} = \frac{M \cdot L^{-1} \cdot T^{-2}}{M \cdot L^{-3} \cdot L^2 \cdot T^{-2}} = 1 \quad (1.11)$$

$$\left[\frac{\mu}{\rho U X} \nabla^{*2} \mathbf{u}^* \right] = \frac{[\mu]}{[\rho][U][X]} = \frac{M \cdot L^{-1} \cdot T^{-1}}{M \cdot L^{-3} \cdot L \cdot T^{-1} \cdot L} = 1 \quad (1.12)$$

$$\left[\frac{\chi X}{\rho U^2} \sin\left(\frac{2\pi Xy^*}{L_y}\right) \right] = \frac{[\chi][X]}{[\rho][U^2]} = \frac{M \cdot L^{-2} \cdot T^{-2} \cdot L}{M \cdot L^{-3} \cdot L^2 \cdot T^{-2}} = 1. \quad (1.13)$$

The terms in (1.8) are all dimensionless, thus we can now make choices for U , X and P which are the scaling parameters for our non-dimensional variables. Immediately and obvious scaling choice is $P = \rho U^2$ as it simplifies the pressure term significantly. We will also define the Reynolds number here $Re = \mu/\rho U X$ as another dimensionless quantity:

$$\frac{\partial \mathbf{u}^*}{\partial t^*} + \mathbf{u}^* \cdot \nabla^* \mathbf{u}^* = -\nabla^* p^* + \frac{1}{Re}\nabla^{*2} \mathbf{u}^* + \frac{\chi X}{\rho U^2} \sin\left(\frac{2\pi Xy^*}{L_y}\right)\mathbf{e}_x. \quad (1.14)$$

We could make a number of choices for X and U here, however by considering the scale of steady state solutions to (1.14) we can make some intuitive choices for X and U . The steady state flow is referred to as Kolmogorov flow, this state has several main features that are helpful

in identifying the solution: firstly the flow is steady, thus solutions remain constant over time and $\partial \mathbf{u}^*/\partial t^* = \mathbf{0}$. Secondly the Kolmogorov forcing occurs only along x^* , hence it is reasonable to conclude that there is no vertical flow $v^* = 0$.

By considering solutions to the Continuity equation (1.5) when $v^* = 0$ we find $\partial u^*/\partial x^* = 0$, thus steady state solutions are axisymmetric in x^* and it follows that $\frac{\partial p^*}{\partial x^*} = 0$. Applying the conditions $\frac{\partial u^*}{\partial t^*} = \frac{\partial p^*}{\partial x^*} = v^* = 0$ to 1.14 in 2D yields:

$$\frac{1}{Re} \frac{\partial^2 u^*}{\partial y^{*2}} + \frac{\chi X}{\rho U^2} \sin\left(\frac{2\pi X y^*}{L_y}\right) = 0. \quad (1.15)$$

Through expanding the Reynolds number, and solving for u^* with $u^*(0) = u^*(L_y/X)$ we obtain:

$$u^* = \frac{\chi L_y^2}{4\pi^2 U \mu} \sin\left(\frac{2\pi X y^*}{L_y}\right). \quad (1.16)$$

By choosing $U = \chi L_y^2 / 4\pi^2 \mu$ we can simplify the form of this solution, $u^* = \sin(2\pi X y^* / L_y)$. Substituting our choice for U back into (1.14) and then dropping the star notation yields:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{u} + \frac{X^2}{Re} \frac{4\pi^2}{L_y^2} \sin\left(\frac{2\pi X y}{L_y}\right) \mathbf{e}_x. \quad (1.17)$$

Here we have 2 obvious choices for X , we will discuss the advantages and disadvantages of both. Choosing $X = L_y$ significantly simplifies the periodicity of the Kolmogorov forcing to $y \in [0, 1]$, however this choice does not offer the cleanest formulation possible. The resulting non-dimensional Navier–Stokes equation with Kolmogorov forcing for $X = L_y$ becomes:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{u} + \frac{4\pi^2}{Re} \sin(2\pi y) \mathbf{e}_x. \quad (1.18)$$

If instead we choose $X = L_y/2\pi$ the resulting Kolmogorov forcing is periodic on $y \in [0, 2\pi]$, which is conceptually more complex than periodicity on $y \in [0, 1]$. The formulation of the resulting non-dimensional equation is however more mathematically elegant than (1.18), and since this paper will frequently manipulate the Kolmogorov system it is favourable to make the

choice $X = L_y/2\pi$, thus we obtain the following system in 2D:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{u} + \frac{1}{Re} \sin(y) \mathbf{e}_x \quad (1.19)$$

$$\nabla \cdot \mathbf{u} = 0. \quad (1.20)$$

Given that we sought solutions earlier, we can determine the steady state velocity for the non-dimensional Kolmogorov system to be:

$$\mathbf{u}_s = (\sin(y), 0, 0). \quad (1.21)$$

1.4 Vorticity Formulation

Recall that we are interested in 2D flows resulting from the Kolmogorov system, such flows are of the form $\mathbf{u} = (u(x, y), v(x, y), 0)$. We can therefore make use of the vorticity formulation of the Navier–Stokes equation, simplifying the vectorised equation into a scalar equation for vorticity. Another advantage of the vorticity equation is the absence of $-\nabla p$, by curling the Navier–Stokes equation this term vanishes, thus reducing the number of variables.

First we define the vorticity of the flow as $\omega = \nabla \times \mathbf{u}$, where the vorticity of 2D flows of the form $\mathbf{u} = (u(x, y), v(x, y), 0)$ becomes $\omega = (0, 0, \omega)$. By applying the methods of [12] we curl (1.19) and utilise incompressibility to obtain the vorticity equation for a 2D flow:

$$\frac{\partial \omega}{\partial t} + \mathbf{u} \cdot \nabla \omega = \frac{1}{Re} \nabla^2 \omega - \frac{1}{Re} \cos(y). \quad (1.22)$$

This equation still contains the vector $\mathbf{u} = (u(x, y), v(x, y), 0)$, to simplify this we can define the stream-function ψ which encodes the velocity profile of the flow as a scalar:

$$u = \frac{\partial \psi}{\partial y}, \quad v = -\frac{\partial \psi}{\partial x}. \quad (1.23)$$

The relationship between the vorticity and the stream function is $\omega = -\nabla^2 \psi$ [12], thus the final

system of equations describing the vorticity of the Kolmogorov system reads:

$$\frac{\partial \omega}{\partial t} + \frac{\partial \psi}{\partial y} \frac{\partial \omega}{\partial x} - \frac{\partial \psi}{\partial x} \frac{\partial \omega}{\partial y} = \frac{1}{Re} \nabla^2 \omega - \frac{1}{Re} \cos(y) \quad (1.24)$$

$$\omega = -\nabla^2 \psi. \quad (1.25)$$

Curling the steady state $\mathbf{u}_s$ recovers the steady state solution:

$$\omega_s = -\cos(y) \quad (1.26)$$

and thus solving (1.25) for the steady state stream function, with the periodic boundary conditions $\psi_s(0) = \psi_s(L_y)$, yields:

$$\psi_s = -\cos(y). \quad (1.27)$$

Chapter 2

Bifurcations from a Steady Flow

The purpose of this section is to determine the stability of the steady state Kolmogorov flow to small perturbations, here we aim to define conditions on the Reynolds number for which the onset of instability occurs in addition to growth rates for specific Fourier modes once the system has become unstable. This will be achieved through the study of bifurcation curves, something that [8] investigated similarly for a 2D domain through the use of continued fractions, and [10] later refined for a 3D periodic domain by approaching the modal coupling as a singular matrix problem. The techniques used in this section will borrow heavily from [10], applying their methods to a 2D system.

2.1 Derivation of the Linearised Vorticity Equation

We must obtain a linear equation which describes the growth of perturbations from the steady state, this is the basis of our stability analysis, where instability is characterised by perturbation growth and stability by perturbation decay. By applying standard linearising techniques we derive a linear equation equivalent to the one used by [8], differing as a result of non-dimensionalisation which yields an equation with the Reynolds number as the only parameter. The first step is to introduce a perturbation into the steady states, by the way of an extra term scaled by a small parameter $\varepsilon \ll 1$:

$$\begin{aligned}\omega(x, y, t) &= \omega_s(x, y, t) + \varepsilon \tilde{\omega}(x, y, t) \\ \psi(x, y, t) &= \psi_s(x, y, t) + \varepsilon \tilde{\psi}(x, y, t).\end{aligned}\tag{2.1}$$

Injecting (2.1) into the non-dimensional vorticity equation yields the perturbation equation:

$$\begin{aligned} \frac{\partial \omega_s}{\partial t} + \frac{\partial \psi_s}{\partial y} \frac{\partial \omega_s}{\partial x} - \frac{\partial \psi_s}{\partial x} \frac{\partial \omega_s}{\partial y} + \varepsilon \left[\frac{\partial \tilde{\omega}}{\partial t} + \frac{\partial \tilde{\psi}}{\partial y} \frac{\partial \omega_s}{\partial x} + \frac{\partial \psi_s}{\partial y} \frac{\partial \tilde{\omega}}{\partial x} - \frac{\partial \tilde{\psi}}{\partial x} \frac{\partial \omega_s}{\partial y} - \frac{\partial \psi_s}{\partial x} \frac{\partial \tilde{\omega}}{\partial y} \right] \\ + \varepsilon^2 \left[\frac{\partial \tilde{\psi}}{\partial y} \frac{\partial \tilde{\omega}}{\partial x} - \frac{\partial \tilde{\psi}}{\partial x} \frac{\partial \tilde{\omega}}{\partial y} \right] = \frac{1}{Re} \nabla^2 \omega_s - \frac{1}{Re} \cos(y) + \frac{\varepsilon}{Re} \nabla^2 \tilde{\omega}. \end{aligned} \quad (2.2)$$

The $O(1)$ terms contain only the steady state variables, and are thus automatically satisfied by the original vorticity equation so they can be removed leaving only smaller terms. We also divide the resulting equation by ε as it simplifies the form:

$$\frac{\partial \tilde{\omega}}{\partial t} + \frac{\partial \tilde{\psi}}{\partial y} \frac{\partial \omega_s}{\partial x} + \frac{\partial \psi_s}{\partial y} \frac{\partial \tilde{\omega}}{\partial x} - \frac{\partial \tilde{\psi}}{\partial x} \frac{\partial \omega_s}{\partial y} - \frac{\partial \psi_s}{\partial x} \frac{\partial \tilde{\omega}}{\partial y} + \varepsilon \left[\frac{\partial \tilde{\psi}}{\partial y} \frac{\partial \tilde{\omega}}{\partial x} - \frac{\partial \tilde{\psi}}{\partial x} \frac{\partial \tilde{\omega}}{\partial y} \right] = \frac{1}{Re} \nabla^2 \tilde{\omega}. \quad (2.3)$$

As $\varepsilon \ll 1$ at the onset of instability a reasonable approximation is to ignore $O(\varepsilon)$ terms, this approximation is only valid whilst the perturbations are small thus solutions to this equation are only indicative of initial instability growth:

$$\frac{\partial \tilde{\omega}}{\partial t} + \frac{\partial \tilde{\psi}}{\partial y} \frac{\partial \omega_s}{\partial x} + \frac{\partial \psi_s}{\partial y} \frac{\partial \tilde{\omega}}{\partial x} - \frac{\partial \tilde{\psi}}{\partial x} \frac{\partial \omega_s}{\partial y} - \frac{\partial \psi_s}{\partial x} \frac{\partial \tilde{\omega}}{\partial y} = \frac{1}{Re} \nabla^2 \tilde{\omega}. \quad (2.4)$$

Having linearised the vorticity equation it is now important to linearise $\omega = -\nabla^2 \psi$, this is done again by injecting the perturbed variables (2.1) into the equation, which reads:

$$\omega_s + \varepsilon \tilde{\omega} = -\nabla^2 \psi_s - \varepsilon \nabla^2 \tilde{\psi}. \quad (2.5)$$

Again the $O(1)$ terms vanish here as they are automatically satisfied by the steady state, dividing by ε simplifies the form:

$$\tilde{\omega} = -\nabla^2 \tilde{\psi}. \quad (2.6)$$

Now that we have derived linearised equations for vorticity, as well as the equation relating vorticity and the stream function, we can substitute the steady state functions $\omega_s = \psi_s = -\cos(y)$ to obtain equations governing instability growth from the Kolmogorov flow steady

state:

$$\frac{\partial \tilde{\omega}}{\partial t} + \sin(y) \frac{\partial}{\partial x} [\tilde{\omega} - \tilde{\psi}] = \frac{1}{Re} \nabla^2 \tilde{\omega} \quad (2.7)$$

$$\tilde{\omega} = -\nabla^2 \tilde{\psi}. \quad (2.8)$$

Combining these 2 equations produces a linearised equation almost identical to the one used by [8], the only difference being the simplified non-dimensional form:

$$\frac{\partial}{\partial t} \nabla^2 \tilde{\psi} + \sin(y) \frac{\partial}{\partial x} [\tilde{\psi} + \nabla^2 \tilde{\psi}] = \frac{1}{Re} \nabla^4 \tilde{\psi} \quad (2.9)$$

2.2 Normal Mode Solutions to the Linearised Vorticity Equation

By observing that (2.9) is linear with respect to time, we can make the remark that $\tilde{\psi}(x, y, t) = f(x, y)e^{\lambda t}$. Additionally the system is periodic in the $\mathbf{e}_x$ direction and autonomous with respect to x , thus permitting the normal mode expansion $\tilde{\psi}(x, y, t) = g(y, t)e^{ik_x x}$. We can now make the substitution $\tilde{\psi}(x, y, t) = \hat{\psi}(y)e^{ik_x x}e^{\lambda t}$.

$$\lambda \left(\frac{d^2}{dy^2} - k_x^2 \right) \hat{\psi} + ik_x \sin(y) \left(\frac{d^2}{dy^2} + 1 - k_x^2 \right) \hat{\psi} = \frac{1}{Re} \left(\frac{d^2}{dy^2} - k_x^2 \right)^2 \hat{\psi} \quad (2.10)$$

The resulting equation is not autonomous in y thus coupling occurs, consequently the same normal mode expansion argument cannot be used. Instead we use the periodicity of the system in the $\mathbf{e}_y$ direction to expand $\hat{\psi}(y)$ as a Fourier series, similarly to [10]:

$$\hat{\psi}(y) = \sum_{k_y=-\infty}^{\infty} c_{k_y} e^{ik_y y} \quad (2.11)$$

Substitution of the Fourier series into (2.10) yields:

$$\sum_{k_y=-\infty}^{\infty} \left[-\lambda(k_x^2 + k_y^2) + ik_x \sin(y)(1 - k_x^2 - k_y^2) \right] c_{k_y} e^{ik_y y} = \sum_{k_y=-\infty}^{\infty} \frac{1}{Re} (k_x^2 + k_y^2)^2 c_{k_y} e^{ik_y y} \quad (2.12)$$

Moving all terms to the LHS, making the substitution $\sin(y) = (e^{iy} - e^{-iy})/2i$, and expanding

yields:

$$\sum_{k_y=-\infty}^{\infty} \left[-\lambda(k_x^2 + k_y^2)e^{ik_y y} - \frac{1}{Re}(k_x^2 + k_y^2)^2 e^{ik_y y} + \frac{k_x}{2}(1 - k_x^2 - k_y^2)c_{k_y}e^{i(k_y+1)y} - \frac{k_x}{2}(1 - k_x^2 - k_y^2)c_{k_y}e^{i(k_y-1)y} \right] = 0 \quad (2.13)$$

where modal coupling arises as a result of the product of complex exponentials similarly to [10]. We want all of the terms in the summation to share a common factor of $e^{ik_y y}$, so we may use the orthogonality of complex exponentials to isolate a recurrence relation for c_{k_y} . Shifting the index of the $e^{ik_y \pm 1}$ terms by ∓ 1 , allows us to represent all the terms in the infinite sum that share the common factor of $e^{ik_y y}$:

$$\sum_{k_y=-\infty}^{\infty} \left[-\lambda(k_x^2 + k_y^2)c_{k_y} + \frac{k_x}{2}(1 - k_x^2 - (k_y - 1)^2)c_{k_y-1} - \frac{k_x}{2}(1 - k_x^2 - (k_y + 1)^2)c_{k_y+1} - \frac{1}{Re}(k_x^2 + k_y^2)^2 c_{k_y} \right] e^{ik_y y} = 0 \quad (2.14)$$

The complex exponentials $e^{ik_y y}$, $k_y \in \mathbb{Z}$ form an orthogonal set, thus multiplication by $\overline{e^{i\eta y}}$ and integration over $y \in [0, 2\pi]$ yields:

$$\sum_{k_y=-\infty}^{\infty} \left[-\lambda(k_x^2 + k_y^2)c_{k_y} + \frac{k_x}{2}(1 - k_x^2 - (k_y - 1)^2)c_{k_y-1} - \frac{k_x}{2}(1 - k_x^2 - (k_y + 1)^2)c_{k_y+1} - \frac{1}{Re}(k_x^2 + k_y^2)^2 c_{k_y} \right] \int_0^{2\pi} e^{ik_y y} \overline{e^{i\eta y}} dy = 0 \quad (2.15)$$

where the integral is only a non-zero and a constant if $k_y = \eta$. Thus setting the summand equal to 0, and isolating $\lambda(k_x^2 + k_y^2)c_{k_y}$ on the LHS yields:

$$\lambda(k_x^2 + k_y^2)c_{k_y} = \frac{k_x}{2}(1 - k_x^2 - (k_y - 1)^2)c_{k_y-1} - \frac{1}{Re}(k_x^2 + k_y^2)^2 c_{k_y} - \frac{k_x}{2}(1 - k_x^2 - (k_y + 1)^2)c_{k_y+1} \quad \forall k_y \in \mathbb{Z} \quad (2.16)$$

This result corroborates what was derived by [10, 8]. Further simplification requires division by $k_x^2 + k_y^2$, to do this we must consider how $\tilde{\psi}$ behaves when $k_x = k_y = 0$, to avoid dividing by 0. The wave numbers $k_x = k_y = 0$ represent a mode with infinite wavelength, which is therefore a constant. Since $\tilde{\psi}$ is defined up to a constant we can set $1/(k_x^2 + k_y^2)$ to any value when $k_x = k_y = 0$ without loss of information, we will choose $1/(k_x^2 + k_y^2) = 0$. Dividing (2.16)

by $k_x^2 + k_y^2$, and further simplifying algebraically yields:

$$\lambda c_{k_y} = k_x \left(\frac{k_y}{k_x^2 + k_y^2} - \frac{1}{2} \right) c_{k_y-1} - \frac{1}{Re} (k_x^2 + k_y^2) c_{k_y} + k_x \left(\frac{k_y}{k_x^2 + k_y^2} + \frac{1}{2} \right) c_{k_y+1} \quad (2.17)$$

We can arrange this recurrence relation as a tridiagonal eigenvalue problem for λ :

$$\lambda \begin{pmatrix} \vdots \\ c_{k_y-1} \\ c_{k_y} \\ c_{k_y+1} \\ \vdots \end{pmatrix} = \mathbf{M} \begin{pmatrix} \vdots \\ c_{k_y-1} \\ c_{k_y} \\ c_{k_y+1} \\ \vdots \end{pmatrix} \quad (2.18)$$

With the tridiagonal, doubly infinite matrix:

$$\mathbf{M} = \begin{pmatrix} \ddots & & & & \\ \ddots & \beta_{k_y-1} & \gamma_{k_y-1} & & \\ & \alpha_{k_y} & \beta_{k_y} & \gamma_{k_y} & \\ & & \alpha_{k_y+1} & \beta_{k_y+1} & \ddots \\ & & & \ddots & \ddots \end{pmatrix} \quad (2.19)$$

and coefficients:

$$\alpha_{k_y} = k_x \left(\frac{k_y}{k_x^2 + k_y^2} - \frac{1}{2} \right), \quad \beta_{k_y} = -\frac{k_x^2 + k_y^2}{Re}, \quad \gamma_{k_y} = k_x \left(\frac{k_y}{k_x^2 + k_y^2} + \frac{1}{2} \right) \quad (2.20)$$

2.3 Convergence of the Bifurcation Curve

Computing the eigenvalues of $\mathbf{M}$ directly is impossible as a result of its infinite size, fortunately we can approximate eigenvalues of the infinite matrix by applying the methods of [10], taking eigenvalues of the matrix:

$$\mathbf{M}_n = \begin{pmatrix} \beta_{-n} & \gamma_{-n} & & & \\ \alpha_{-n+1} & \beta_{-n+1} & \ddots & & \\ & \ddots & \ddots & \gamma_{n-1} & \\ & & \alpha_n & \beta_n & \end{pmatrix} \quad (2.21)$$

as $n \rightarrow \infty$, where $\mathbf{M}_n \rightarrow \mathbf{M}$ at this limit. This approximation relies on the convergence of the eigenvalues of (2.21), given as solutions to the characteristic equation $P_n(k_x, Re, \lambda) = 0$, where $P_n(k_x, Re, \lambda) = \det(\mathbf{M}_n - \lambda \mathbf{I})$. It is important to note that the growth of unstable modes is determined by only the largest eigenvalue.

The bifurcation curve of the Kolmogorov system in 3D was proven to converge as $n \rightarrow \infty$ by [10]. We will adapt this method to show convergence of the bifurcation curve for the 2D Kolmogorov system. We seek solutions to the characteristic equation $P_n(k_x, Re) = 0$ that converge as $n \rightarrow \infty$, with $P_n(k_x, Re) = \det(\mathbf{M}_n)$, where choosing $\lambda = 0$ forces solutions to define a bifurcation curve on $k_x, Re \in \mathbb{R}^2$. It is shown by [10] that we can use the decomposition $P_n(k_x, Re) = Q_n(k_x, Re) \prod_{j=-n}^n \beta_j$, given that $\beta_j \neq 0$, where:

$$Q_n(k_x, Re) = \begin{vmatrix} 1 & \frac{\gamma_{-n}}{\beta_{-n}} & & \\ \frac{\alpha_{-n+1}}{\beta_{-n+1}} & 1 & \ddots & \\ & \ddots & \ddots & \frac{\gamma_{n-1}}{\beta_{n-1}} \\ & & \frac{\alpha_n}{\beta_n} & 1 \end{vmatrix} \quad (2.22)$$

thus solutions to $Q_n(k_x, Re) = 0$ must also satisfy $P_n(k_x, Re) = 0$. The case $\beta_j = 0$ only occurs for $j = k_x = 0$. We can show that $\lambda \leq 0$ always at $k_x = 0$ by considering the matrix (2.19), where its coefficients above and below the leading diagonal $\alpha_{k_y} = \gamma_{k_y} = 0$ for all k_y . As a result the characteristic polynomial can be written $P_n(0, Re) = (\beta_0 - \lambda) \prod_{j=1}^n (\beta_1 - \lambda)(\beta_{-1} - \lambda)$, where $\beta_0 = 0$ results in the largest eigenvalue being $\lambda = 0$. Thus the system is always marginally stable at $k_x = 0$, and proof of convergence here is not necessary.

It is shown by [10] that solutions to $Q_n(k_x, Re) = 0$ converge if $\sum_{n=1}^{\infty} |\Lambda_{\pm n}|$ are convergent series, where $\Lambda_n = \frac{\alpha_{n+1}\gamma_n}{\beta_{n+1}\beta_n}$. By substituting the coefficients (2.20) we obtain:

$$\Lambda_n = \frac{Re^2 k_x^2 (1 - k_x^2 - n^2)(2n + k_x^2 + n^2)}{4 (k_x^2 + n^2)^2 (k_x^2 + (n+1)^2)^2} \quad (2.23)$$

where convergence is proven by applying the limit comparison test, given by [7]. The limit comparison test states, $\sum_{n=1}^{\infty} |a_n|$ and $\sum_{n=1}^{\infty} |b_n|$ either both converge or diverge if $\lim_{n \rightarrow \infty} |a_n/b_n| = c$, where $0 < c < \infty$. Setting $a_n = |\Lambda_{\pm n}|$ and choosing the comparison series $b_n = 1/n^4$ yields:

$$\lim_{n \rightarrow \infty} \frac{|\Lambda_n|}{1/n^4} = \frac{Re^2 k_x^2}{4}, \quad \lim_{n \rightarrow \infty} \frac{|\Lambda_{-n}|}{1/n^4} = \frac{Re^2 k_x^2}{4}. \quad (2.24)$$

Here the series $\sum_{n=1}^{\infty} 1/n^4$ is known to converge, thus $\sum_{n=1}^{\infty} |\Lambda_{\pm n}|$ converges by the limit comparison test, given k_x , and solutions to $Q_n(k_x, Re) = 0$ converge. Recall that $Q_n(k_x, Re) = 0$ implies $P_n(k_x, Re) = 0$, as a result solutions to $Q_n(k_x, Re) = 0$ must satisfy $P_n(k_x, Re) = 0$, thus the solutions defining the bifurcation curve on $k_x, Re \in \mathbb{R}^2$ converge as $n \rightarrow \infty$.

2.4 Solutions for the Matrix $\mathbf{M}_1$

We have discussed the convergence of the bifurcation curve $P_n(k_x, Re) = 0$ as $n \rightarrow \infty$. By considering the eigenvalues of $\mathbf{M}_1$, and solving for $\lambda = 0$, we can identify important features of the bifurcation curve. We start by defining:

$$\mathbf{M}_1 = \begin{pmatrix} \beta_{-1} & \gamma_{-1} & 0 \\ \alpha_0 & \beta_0 & \gamma_0 \\ 0 & \alpha_1 & \beta_1 \end{pmatrix}. \quad (2.25)$$

By considering the structure of the matrix coefficients we find $\alpha_{k_y} = -\gamma_{-k_y}$, $\beta_{k_y} = \beta_{-k_y}$, $\gamma_{k_y} = -\alpha_{-k_y}$. Applying this to $\mathbf{M}_1$ yields a simplified matrix that is easier to manipulate:

$$\mathbf{M}_1 = \begin{pmatrix} \beta_1 & -\alpha_1 & 0 \\ \alpha_0 & \beta_0 & -\alpha_0 \\ 0 & \alpha_1 & \beta_1 \end{pmatrix}. \quad (2.26)$$

The characteristic equation of this matrix becomes:

$$(\beta_1 - \lambda)((\beta_0 - \lambda)(\beta_1 - \lambda) + 2\alpha_0\alpha_1) = 0 \quad (2.27)$$

Solving for its eigenvalues yields:

$$\lambda_1 = \beta_1, \quad \lambda_{2\pm} = \frac{1}{2}(\beta_0 + \beta_1 \pm [(\beta_0 - \beta_1)^2 - 8\alpha_0\alpha_1]^{\frac{1}{2}}). \quad (2.28)$$

Expanding the coefficients gives:

$$\lambda_1 = -\frac{k_x^2 + 1}{Re}, \quad \lambda_{2\pm} = \frac{1}{2} \left[-\frac{2k_x^2 + 1}{Re} \pm \left(\frac{1}{Re^2} + 2k_x^2 \frac{1 - k_x^2}{1 + k_x^2} \right)^{\frac{1}{2}} \right]. \quad (2.29)$$

Since $Re \geq 0$ by definition [10] it follows that $\lambda_1, \lambda_{2-} < 0$ always, thus the only eigenvalue capable of producing growth is λ_{2+} . The bifurcation curve of this eigenvalue is obtained by setting $\lambda_{2+} = 0$ and manipulating, to obtain:

$$k_x = 0 \quad (2.30)$$

$$Re = \left[2 \frac{(1 + k_x^2)^2}{1 - k_x^2} \right]^{\frac{1}{2}} \quad (2.31)$$

We see that the bifurcation is real and bounded on the interval $k_x \in (-1, 1)$, where the curve is undefined at $k_x = \pm 1$. $k_x = 0$ is true always, thus we will examine only (2.31). Figure 2.1 shows the bifurcation curve (2.31). It is clear that the curve has a minimum as $k_x \rightarrow 0$, this is the critical Reynolds number for which instability occurs. Taking the limit $k_x \rightarrow 0$ of (2.31) yields the critical Reynolds number $Re_c = \sqrt{2}$.

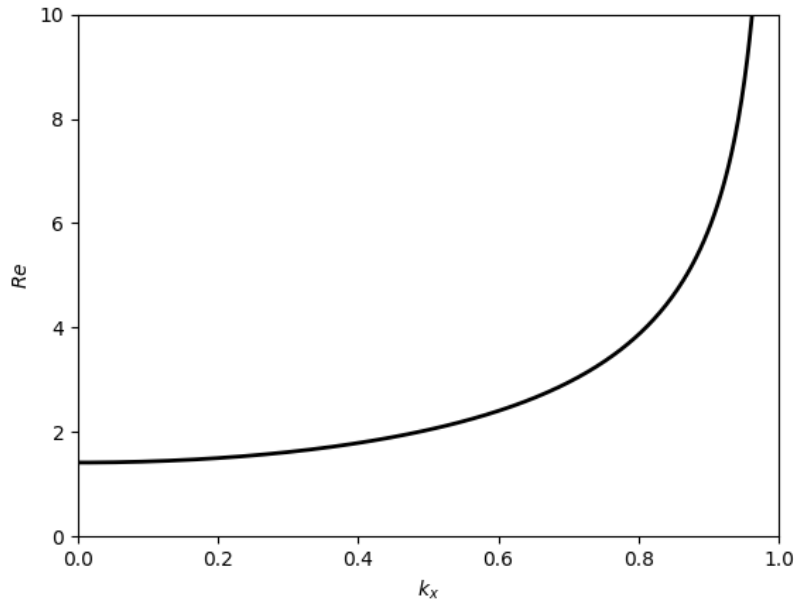


Figure 2.1: Bifurcation curve (2.31) for the matrix $\mathbf{M}_1$, plotted in Python.

2.5 Critical Reynolds Number Convergence Study

We have shown convergence for the bifurcation curve, however in practice it is useful to also understand the growth rates we would expect from specific modes with wave number k_x at a given Re . The function “`growth()`” was created in python (Appendix A.1.1), which returns the maximum eigenvalue of the matrix $\mathbf{M}_n$ for a given n, k_x and Re . We discretise k_x, Re on

the following numerical grid:

$$\begin{aligned}(k_x)_j &= j\Delta S + (k_x)_0, \quad j = 0, \dots, M \\ (Re)_\ell &= \ell\Delta S + (Re)_0, \quad \ell = 0, \dots, P\end{aligned}\tag{2.32}$$

where ΔS is the grid spacing. We now define a new function “`growth_mesh()`” which calls `growth()` at every point on (2.32), thus yielding $\lambda_{j,\ell}^n$ evaluated at every point, for a chosen n .

We can now test the convergence of the critical Reynolds number, as $n \rightarrow \infty$. We first consider (2.32) where $(k_x)_0 = \Delta S$ and $(Re)_0 = \sqrt{2} - P\Delta S/2$, the resulting grid (2.33) is spaced ΔS away from the Re axis, and is centred on $\sqrt{2}$ on the Re axis:

$$\begin{aligned}(k_x)_j &= (1+j)\Delta S, \quad j = 0, \dots, M \\ (Re)_\ell &= \left(\ell - \frac{P}{2}\right)\Delta S + \sqrt{2}, \quad \ell = 0, \dots, P\end{aligned}\tag{2.33}$$

We choose to centre the grid on $Re = \sqrt{2}$ as we wish to examine change from $Re_c = \sqrt{2}$ as n , or ΔS change. Additionally we choose $(k_x)_0 = \Delta S$ since the growth rate $\lambda = 0$ for $k_x = 0$.

We will first show convergence of Re_c as $n \rightarrow \infty$, where we define the critical Reynolds number at a given n as Re_c^n . We evaluate Re_c^n at some $n = N$, and introduce the quantity $|Re_c^n - Re_c^N|$ which can be used to quantify the convergence. The Python script “`Re_c_n.convergence.py`” was created (Appendix A.1.3), which evaluates the largest eigenvalue on (2.33) for a given n , draws the bifurcation curve for $\lambda = 0$, and then applies the `allsegs()` function to obtain the minimum of the bifurcation curve Re_c^n . The script then computes $|Re_c^n - Re_c^N|$ and saves the result to an array. Figure 2.2 shows the result of running “`Re_c_n.convergence.py`” with $N = 40$, $\Delta S = 0.01$ and $M = P = 10$. We see that there is no measurable change in $|Re_c^n - Re_c^N|$ as $n \rightarrow 40$.

We can show a similar result of convergence as $\Delta S \rightarrow 0$. We start by defining $\Delta S_h = \Delta S_1/h$ where $h = 1, \dots, H$. We now define the critical Reynolds number for a given h as Re_c^h , where we define $|Re_c^h - Re_c^H|$ as a measure of convergence. The Python script “`Re_c_h.convergence.py`” was created (Appendix A.1.4), which evaluates the largest eigenvalue on (2.33) for a given h , draws the bifurcation curve for $\lambda = 0$, and then applies the `allsegs()` function to obtain the minimum of the bifurcation curve Re_c^h . The script then computes $|Re_c^h - Re_c^H|$ and saves the result to an array. Figure 2.3 shows the result of running “`Re_c_h.convergence.py`” with

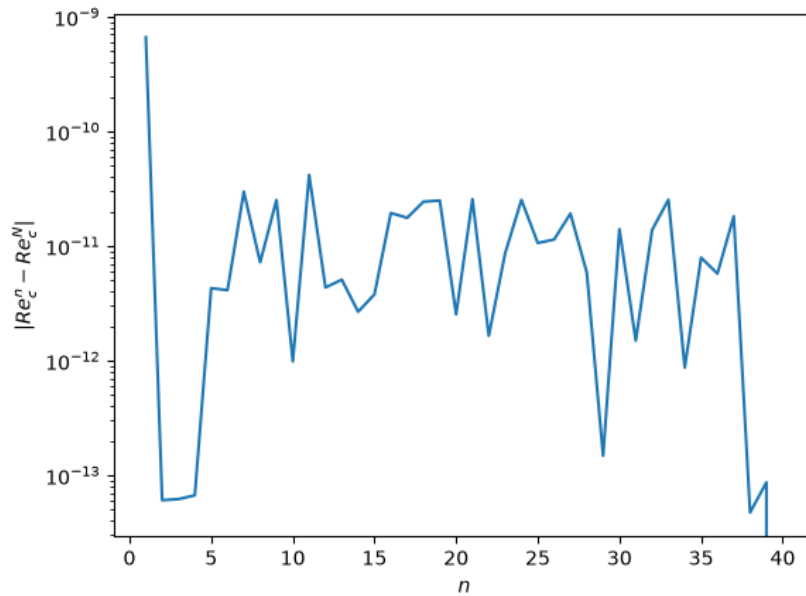


Figure 2.2: Plot showing $|Re_c^n - Re_c^N|$ against n , where $N = 40$, $M = P = 10$, $\Delta S = 0.01$, computed and plotted in Python (Appendix A.1.3).

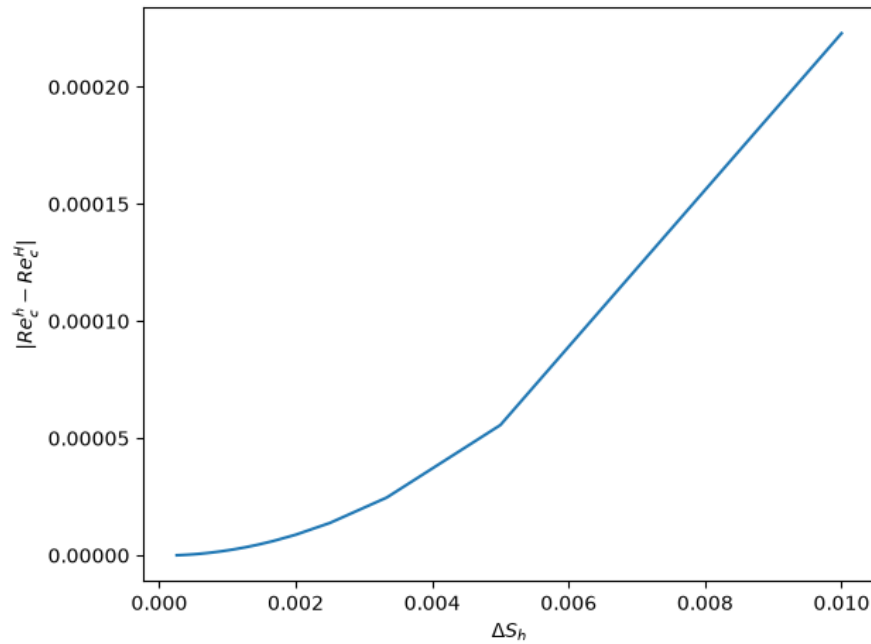


Figure 2.3: Plot showing $|Re_c^h - Re_c^H|$ against ΔS_h , where $N = 20$, $M = P = 10$, $H = 40$, computed and plotted in Python (Appendix A.1.4)

$N = 20, H = 40$ and $M = P = 10$. We see that as $\Delta S \rightarrow 0$ the plot $|Re_c^h - Re_c^H| \rightarrow 0$. Since the value of the critical Reynolds number is unchanging as $n \rightarrow \infty$, and convergent as $\Delta S \rightarrow 0$ we surmise that the critical Reynolds number is $Re_c = \sqrt{2}$ for all n .

Chapter 3

Spectral Methods

Often, differential equations that describe real world phenomena are far too complex to solve analytically. Spectral methods offer numerical solutions to these problems, that are high precision, and computationally inexpensive in comparison to other methods [5] through virtue of the Fast Fourier Transform (FFT). This section will make extensive use of material presented by [9, 3] in order to motivate spectral schemes that we will later apply to the Kolmogorov system. It is important to note, we will be mainly exploring ideas that concern periodic systems, thus ideas such as Chebyshev grids and solutions to boundary value problems will be largely ignored.

3.1 Spectral Differentiation

We will now motivate spectral differentiation through the discretisation of a periodic system. It is first important to understand decomposition of any function, that is periodic over Γ , into the discrete Fourier series given by [3]:

$$f(x) = \sum_{n=-\infty}^{\infty} \hat{f}_n \exp\left(inx \frac{2\pi}{\Gamma}\right). \quad (3.1)$$

The complex Fourier coefficient $\hat{f}_n$ can be obtained through the following integral [15]:

$$\hat{f}_n = \int_0^{\Gamma} f(x) \exp\left(-inx \frac{2\pi}{\Gamma}\right) dx. \quad (3.2)$$

In [9] the spectral derivative is motivated through the spatial derivative of the Inverse Fourier Transform (IFT). Since the IFT arises from the Fourier series as $\Gamma \rightarrow \infty$, shown by [16], we can

generalise this result by spatially differentiating the Fourier series:

$$\frac{df}{dx}(x) = \sum_{n=-\infty}^{\infty} in \frac{2\pi}{\Gamma} \hat{f}_n \exp\left(inx \frac{2\pi}{\Gamma}\right). \quad (3.3)$$

The Fourier coefficient $\hat{f}_n$ exists on the unbounded grid $n \in \mathbb{Z}$ which we will refer to as spectral space. In order to solve systems numerically they must become spatially discretised, we consider the function $f(x)$ evaluated on the periodic grid below, where we have introduced the notation $f(x_j) = f_j$:

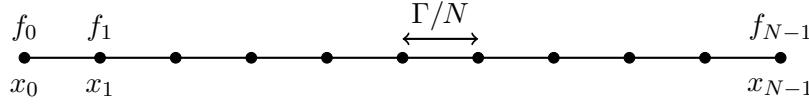


Figure 3.1: Equidistributed Γ periodic grid $x_j = j\Gamma/N$, $j = 0, 1, \dots, N-1$ with data values f_j and periodic boundary condition $x_0 = x_N$, reproduced from [9].

We can first approximate (3.2) on this grid discretely using the trapezoid rule, as noted by [9], to obtain the Discrete Fourier Transform (DFT):

$$\hat{f}_n = \frac{1}{N} \sum_{j=0}^{N-1} f_j \exp\left(-inj \frac{2\pi}{N}\right). \quad (3.4)$$

By discretising the system spatially we bound the possible wave numbers to $n = -N/2 + 1, \dots, N/2$, as wave numbers differing by an integer multiple of N are indistinguishable [9]. By bounding the limits of n in (3.1) we arrive at the Inverse Discrete Fourier Transform (IDFT):

$$f_j = \sum_{n=-N/2+1}^{N/2} \hat{f}_n \exp\left(inj \frac{2\pi}{N}\right). \quad (3.5)$$

Similarly bounding (3.3), and then taking its DFT, yields the definition of the discrete spectral derivative [9]:

$$\frac{d\hat{f}_n}{dx} = i \frac{2\pi n}{\Gamma} \hat{f}_n \quad (3.6)$$

with $n = -N/2 + 1, \dots, N/2$. Applying the IDFT yields an approximation for the derivative:

$$\frac{df_j}{dx} \approx \mathcal{F}^{-1} \left[i \frac{2\pi n}{\Gamma} \hat{f}_n \right] \quad (3.7)$$

where $\mathcal{F}$ is the DFT operator, and $\mathcal{F}^{-1}$ its inverse. For this paper we will be working with real valued functions. It follows from (3.4) that if $f_j \in \mathbf{R}$ then $\hat{f}_n = \hat{f}_n^*$, where $*$ is the complex conjugate, as shown by [3], thus the IDFT can be expressed:

$$f_j = \hat{f}_0 + \sum_{n=1}^{N/2-1} \left[\hat{f}_n \exp\left(inj\frac{2\pi}{N}\right) + \hat{f}_n^* \exp\left(inj\frac{2\pi}{N}\right) \right] + \hat{f}_{N/2}(-1)^j. \quad (3.8)$$

This expanded form (3.8), given by [3], illustrates that the transform of real valued data can be represented by $n = 0, \dots, N/2$, which reduces the size of the grid in spectral space for efficient computing.

3.2 The Fast Fourier Transform

Discovered in 1965 by Cooley and Tukey [4], the FFT is a computationally efficient algorithm that computes the DFT and IDFT. Computing the DFT normally on a grid of N points has a time complexity of $O(N^2)$, whilst the FFT reduces this to a time complexity of $O(N \ln N)$, according to [9]. Utilising FFT massively reduces the run time of simulations, especially for grids with a large number of points. To illustrate the efficiency of the FFT consider a grid with $N = 64$ points, taking the standard DFT takes 4096 computations, whilst the FFT only requires 226 computations making it approximately 15.3 time faster. For computing spectral derivatives the speed of the FFT outperforms other algorithms, such as the spectral differentiation matrix given by [9], whilst providing the same degree of accuracy. The rest of this paper will therefore use the `Scipy.fft` Python library for computing derivatives, by means of spectral methods.

3.3 Spectral Accuracy

We will now illustrate the spectral accuracy through an example. Consider the 2π periodic real valued function:

$$f(x) = \exp\left(\sin\left(\frac{x}{2}\right) \cos\left(\frac{x}{2}\right)\right). \quad (3.9)$$

Which has the analytical derivative:

$$\frac{df}{dx}(x) = \frac{f(x)}{2} \left[\cos^2\left(\frac{x}{2}\right) - \sin^2\left(\frac{x}{2}\right) \right] \quad (3.10)$$

Evaluating $f(x)$ on the grid $x_j = 2\pi j/N$, $j = 0, \dots, N-1$, and applying the spectral method (3.7) yields the approximation:

$$\frac{df_j}{dx} \approx \mathcal{F}^{-1} [in\mathcal{F}(f_j)]. \quad (3.11)$$

where $n = 0, \dots, N/2$.

Figure 3.2 shows the RMSE of approximation (3.7), scaled logarithmically, as the number of grid points N increases. The RMSE initially decreases linearly on the logarithmic axis, which is indicative of exponential decay, thus RMSE is proportional to $\exp(-N)$. This phenomenon is referred to as spectral accuracy [9], and it can be used to validate schemes as being spectral. The approximation gains no additional accuracy once the RMSE reaches $O(10^{-14})$, as a result of computational rounding errors that occur at this scale [9].

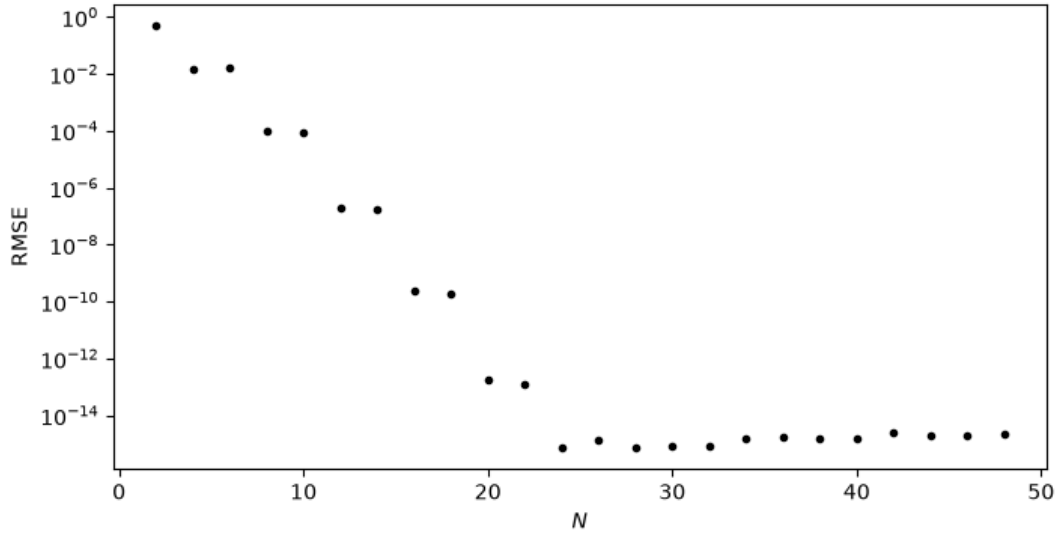


Figure 3.2: Scatter plot showing the Root Mean Squared Error (RMSE) of approximation (3.11), with respect to the analytical derivative (3.10), for different grid resolutions N . Results computed and plotted on a logarithmic RMSE axis in Python (Appendix A.2.1) using methods from [9].

3.4 The 2D Discrete Fourier Transform

With the Kolmogorov system in mind it is crucial to extend the DFT to 2D. We will start by considering a domain with periodicity Γ_x in the x -direction, and periodicity Γ_y in the y -direction

on the equidistributed grid, as given by [3]:

$$\begin{aligned} x_j &= \frac{j\Gamma_x}{M}, \quad j = 0, \dots, M-1 \\ y_\ell &= \frac{\ell\Gamma_y}{N}, \quad \ell = 0, \dots, N-1. \end{aligned} \quad (3.12)$$

We consider the 2D DFT for real valued data, as the simulations in the paper do not involve complex physical quantities. It is shown by [3] that applying the DFT first in the x -direction, and then the y -direction transforms the entire 2D grid into spectral space, thus we have the 2D DFT:

$$\hat{f}_{m,n} = \frac{1}{MN} \sum_{\ell=0}^{N-1} \sum_{j=0}^{M-1} f(x_j, y_\ell) \exp\left(imj\frac{2\pi}{M}\right) \exp\left(-in\ell\frac{2\pi}{N}\right). \quad (3.13)$$

Applying the DFT to real data in the x -direction produces a grid of complex values, and so the second DFT in the y -direction must treat the data as complex, thus $m = 0, \dots, M/2$, and $n = -N/2 + 1, \dots, N/2$. We can similarly define the 2D IDFT:

$$\begin{aligned} f(x_j, y_\ell) &= \sum_{n=-N/2+1}^{N/2} \exp\left(in\ell\frac{2\pi}{N}\right) \left(\hat{f}_{0,n} \dots \right. \\ &\quad \left. + \sum_{m=0}^{M/2} \left[\hat{f}_{m,\ell} \exp\left(imj\frac{2\pi}{M}\right) + \hat{f}_{m,\ell}^* \exp\left(-imj\frac{2\pi}{M}\right) \right] + \hat{f}_{M/2,n}(-1)^j \right). \end{aligned} \quad (3.14)$$

3.5 Spectral Aliasing

Recall that only wave numbers $n = -N/2 + 1, \dots, N/2$ carry information, on a discrete periodic grid, as wave numbers differing by an integer multiple of N become indistinguishable [9, 3]:

$$\exp\left(i(n+N)j\frac{2\pi}{N}\right) = \exp\left(inj\frac{2\pi}{N}\right). \quad (3.15)$$

We can therefore work with the equivalent range of wave numbers $n = 0, \dots, N-1$, where we have switched to inequality notation, as this range is more suitable for motivating aliasing. We now consider modes with wave numbers $n = m + N/2$, with $m < N/2$, similarly to [3]:

$$\exp\left(i\left(\frac{N}{2} + m\right)j\frac{2\pi}{N}\right) = \exp\left(i\left(\frac{N}{2} - m\right)j\frac{2\pi}{N}\right)^*. \quad (3.16)$$

This illustrates the phenomenon of frequency folding, where modes of wave number $N/2 + m$ project back onto modes of wave number $N/2 - m$, where the wave number $N/2$ responsible for this phenomenon is called the Nyquist frequency [3]. The modes of wave number $N/2 \pm m$ are represented by complex conjugate coefficients [3], thus $\hat{f}_{N/2+m} = \hat{f}_{N/2-m}^*$, which can be proven by applying (3.16) to the DFT.

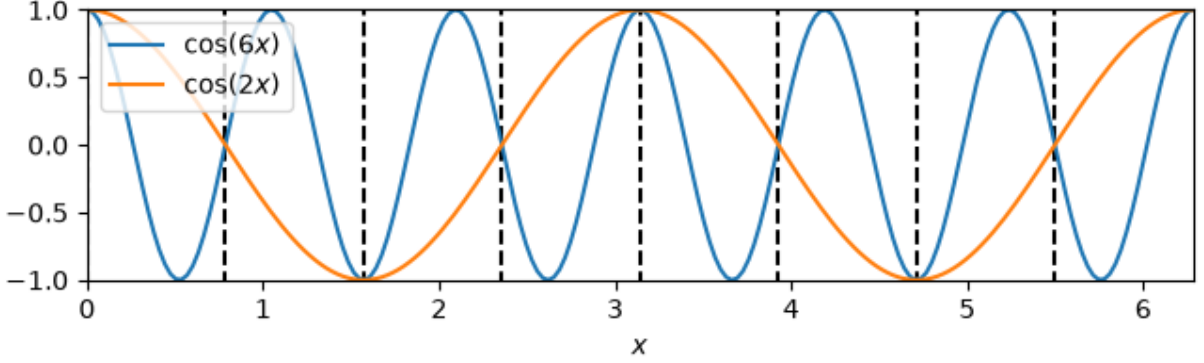


Figure 3.3: Modes $\cos(6x)$ and $\cos(2x)$ plotted on the periodic domain $x \in [0, 2\pi]$, with the vertical dashed lines representing the grid $x_j = 2\pi j/N$, $j = 0, \dots, N-1$ for $N = 8$.

Figure 3.3 demonstrates frequency folding of mode $\cos(6x)$ onto $\cos(2x)$, on a 2π periodic domain with $N = 8$ points. The modes appear mostly separate, intersecting only at points on the grid x_j , this is a result of the mode of wave number $n = 6$ being folded back onto the mode of wave number $n = 2$. Frequency folding results in modes of wave number greater than the Nyquist frequency being misrepresented, [3] refers to this phenomenon as aliasing.

Aliasing is not an issue for systems with linear equations, as the modes never interact with one another, however issues arise with non-linearity. Consider the following product of modes, with non-linearity of order η , according to [3]:

$$\exp\left(inj\frac{2\pi}{N}\right)^\eta = \exp\left(i\eta nj\frac{2\pi}{N}\right). \quad (3.17)$$

The resulting mode of wave number ηn could potentially fold back onto modes with wave number n , or smaller than n . According to [3] the resulting redistribution of energy from high wave number modes, to low wave number modes can be mitigated by applying a Fourier filter. The role of the Fourier filter is to remove modes that fold back onto, or past themselves, after

non linearity is applied. The Fourier filter given by [3] reads:

$$\eta n < \frac{N}{2} + \left(\frac{N}{2} - n \right) \quad (3.18)$$

where $N/2 - n = m$ is the distance from the mode of wave number n to the Nyquist frequency.

Rearranging yields:

$$n < \frac{N}{\eta + 1} \quad (3.19)$$

given by [3]. Here we have discussed frequency folding past the positive Nyquist frequency, however if we had instead discussed wave numbers on the range $-N + 1, \dots, 0$ the notion of the negative Nyquist frequency would have arisen [3]. Repeating the process in this section for the range $-N + 1, \dots, 0$ yields an equivalent Fourier filter condition for modes with negative wave numbers:

$$n > -\frac{N}{\eta + 1} \quad (3.20)$$

Thus for a non-linearity ηn applied to modes of wave-number $n = -N/2 + 1, \dots, N/2$ we must apply Fourier filtering for the positive, and negative wave numbers [3]. We can consider the simple example of a grid with $N = 8$ points, with wave numbers $n = -3, \dots, 4$, and a non-linearity $\eta = 2$. Evaluating the Fourier filter conditions for positive and negative wave numbers we find $|n| < 8/3 = 2.66\dots$, thus modes with wave numbers $n = \pm 3$ and $n = 4$ are omitted from the non-linear term.

Chapter 4

Time Stepping and Numerical Stability

We have discussed how spectral methods can be applied to numerically solve steady partial differential equations (PDEs), however this methodology is not applicable to systems that evolve with time. Numerically modelling systems that evolve over time requires the use of iterative time stepping algorithms, where time is discretised accordingly. This section will illustrate time stepping, by considering simple implicit and explicit finite difference “Euler” schemes, which will motivate the notion of numerical stability. We will then investigate the Adams-Bashforth and Adams-Moulton methods, as these offer an approach to discretise the non-linear advection and time derivative terms, which both appear in the vorticity equation we wish to solve.

4.1 Numerical Stability

In order to understand the stability of an iterative numerical scheme it is best to start with the simplest time stepping method, finite differences. To motivate the notion of numerical stability we consider the diffusion equation, discretised on the periodic grid $x_j = j\Gamma/N$, $j = 0, \dots, N-1$:

$$\frac{\partial u_j}{\partial t} = D \frac{\partial^2 u_j}{\partial x^2}. \quad (4.1)$$

Taking the DFT of the equation, and evaluating the spectral derivative yields:

$$\frac{\partial \hat{u}_n}{\partial t} = -Dk_n^2 \hat{u}_n \quad (4.2)$$

where we have defined the physical wave number $k_n = 2\pi n/\Gamma$ to simplify notation. We wish to evolve the system numerically with time, we thus discretise (4.2) temporally on the grid $t_c = c\Delta t$, $c = 0, \dots, T$. Additionally We introduce the notation $u(t) = u^{(t)}$ to simplify the form. By utilising an approximation for the time derivative we can derive a time stepping scheme for $\hat{u}^{(t)}$ implicitly, or $\hat{u}^{(t+\Delta t)}$ explicitly. Evaluating (4.2) at running time t , and using forward finite difference to approximate the time derivative yields the implicit scheme:

$$\frac{\hat{u}_n^{(t)} - \hat{u}_n^{(t-\Delta t)}}{\Delta t} \approx -Dk_n^2 \hat{u}_n^{(t)}. \quad (4.3)$$

Similarly evaluating (4.2) at running time t and using backward finite difference to approximate the time derivative yields the explicit scheme:

$$\frac{\hat{u}_n^{(t+\Delta t)} - \hat{u}_n^{(t)}}{\Delta t} \approx -Dk_n^2 \hat{u}_n^{(t)}. \quad (4.4)$$

We will first analyse the stability of the implicit scheme (4.3), this is achieved by applying the Von-Neumann stability ansatz $u_j^{(t)} = g^{\Delta t} e^{ikx_j}$ given by [5]:

$$g^{\Delta t} = \frac{1}{1 + \Delta t Dk_n^2}. \quad (4.5)$$

Since $u_j^{(t+\Delta t)} = g^{\Delta t} u_j^{(t)}$ we can determine that stability only occurs when $|g^{\Delta t}| < 1$, as [5] indicates that the solution would grow without bound if $|g^{\Delta t}| > 1$. Evaluating $|g^{\Delta t}| < 1$ for (4.5) gives a stability condition $|1 + \Delta t Dk_n^2| > 1$ which is only true if $\Delta t Dk_n^2 < -2$ and $\Delta t Dk_n^2 > 0$. Since $\Delta t Dk_n^2$ is positive these conditions are always satisfied, hence the implicit scheme is unconditionally stable, which is expected according to [5].

Similarly following the same method for the explicit scheme yields the stability condition $0 < \Delta t Dk_n^2 < 2$. This is a far more limiting condition on $\Delta t Dk_n^2$ as the scheme can become unstable, and the stability is bound to a finite interval $\Delta t Dk_n^2 \in (0, 2)$. We can evaluate an upper bound for the size of Δt , so that the solution is numerically stable. Rearranging $\Delta t Dk_n^2 < 2$ for the

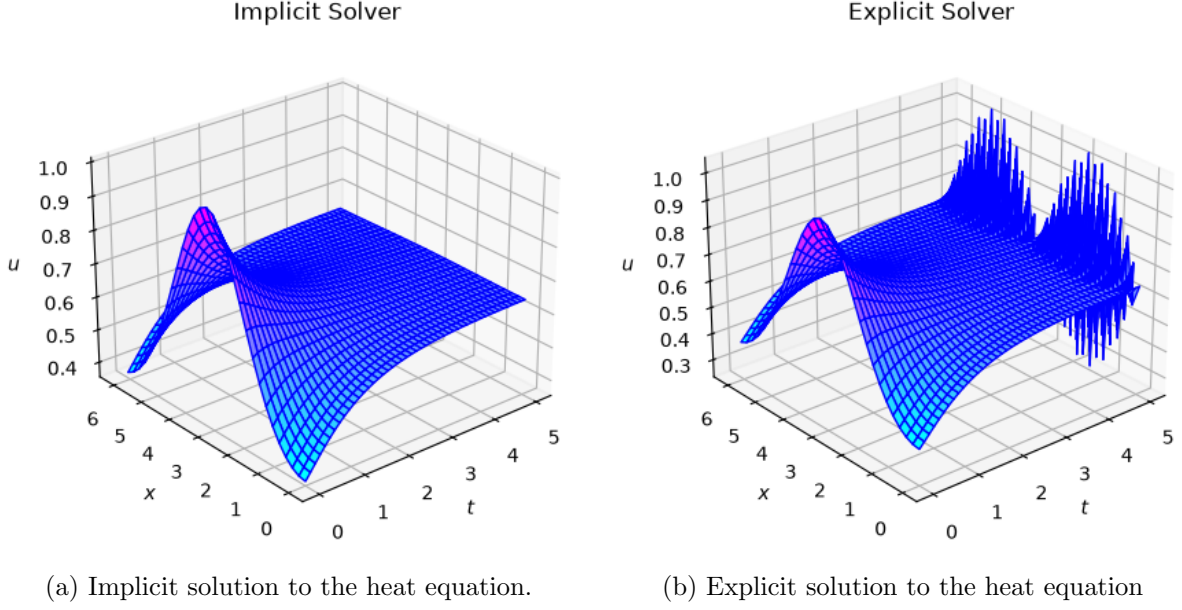


Figure 4.1: Implicit and explicit solutions to the heat equation, evolved using finite differences in time, on a spatially periodic domain $x, t \in [0, 2\pi] \times [0, 5]$ discretised spatially and temporally. The system has an initial profile $u_0 = \exp(-\cos^2(x/2))$, and parameters $D = 1$, $\Delta t = 1.065 \times 10^{-2}$, $N = 30$, $\Gamma = 2\pi$. Results computed and graphed using Python (Appendix A.3.1).

time step, and substituting the maximum wave number $k_{N/2} = \pi N/\Gamma$, similarly to [9], yields:

$$\Delta t < \frac{2\Gamma^2}{DN^2\pi^2}. \quad (4.6)$$

Figure 4.1 shows the results of solving the heat equation implicitly and explicitly. The simulation uses parameters $N = 30$, $\Gamma = 2\pi$ and $D = 1$, thus we can construct the upper bound $\Delta t < 8/30^2 \approx 8.8 \times 10^{-3}$, for which the stability of the explicit scheme fails. The solutions in figure 4.1 use $\Delta t = 1.065 \times 10^{-2}$, thus it is unsurprising that the explicit solution becomes unstable with instabilities that grow over time. The implicit solution remains well behaved, as a result of the unconditional stability of implicit schemes [5]. It is clear that our choice of scheme and Δt is critical to the stability of the solutions, where the choice of implicit methods can increase numerical stability [5].

4.2 Extrapolating the Non-linear Term

Recall that the vorticity equation contains the non-linear advection term $\mathbf{u} \cdot \nabla \omega$, which we will refer to as J henceforth to reduce heavy notation, and the time derivative $\partial\omega/\partial t$. We will evaluate the time derivative implicitly at time t as this improves the stability of the solver,

similarly to [3], as a result J^t is unknown. The non-linearity of the advection term makes it impossible to solve implicitly, thus we choose to extrapolate J^t explicitly using the Adams-Bashforth method. It is worthwhile noting that whilst the implementation here focuses on non-linear advection, this method is valid for linear terms also.

It is shown by [3] that J^t can be extrapolated using a linear combination past iterations:

$$J^t \approx \sum_{j=1}^{\eta} \beta_j J^{(t-j\Delta t)} \quad (4.7)$$

where β is a coefficient, and η is the order of the resulting scheme. Since this method only extrapolates from previous time, any change in dynamics from the steady state will not be detected until after one time-step. The time stepping of our scheme will be small, hence the accuracy of a second order scheme is sufficient here [9]. According to [3] a second order Adams-Bashforth scheme requires $J^{t-\Delta t}$ and $J^{t-2\Delta t}$, which are expanded as a Taylor series based on t :

$$J^{t-\Delta t} = J^t + \frac{(-1)^1 \Delta t}{1!} \frac{\partial J^t}{\partial t} + O(\Delta t^2) \quad (4.8)$$

$$J^{t-2\Delta t} = J^t + \frac{(-2)^1 \Delta t}{1!} \frac{\partial J^t}{\partial t} + O(\Delta t^2). \quad (4.9)$$

We can now construct a linear combination of $J^{t-\Delta t}$ and $J^{t-2\Delta t}$:

$$\beta_1 J^{t-\Delta t} + \beta_2 J^{t-2\Delta t} = (\beta_1 + \beta_2) J^t - \frac{(\beta_1 + 2\beta_2) \Delta t}{1!} \frac{\partial J^t}{\partial t} + O(\Delta t^2). \quad (4.10)$$

By choosing coefficients, such that $\beta_1 + \beta_2 = 1$ and $\beta_1 + 2\beta_2 = 0$, we preserve J^t whilst eliminating the first derivative term [3]. Solving yields $\beta_1 = 2$, $\beta_2 = -1$. Substituting these values into (4.10) gives the second order scheme:

$$2J^{t-\Delta t} - J^{t-2\Delta t} = J^t + O(\Delta t^2) \quad (4.11)$$

which produces an approximation J^t with error $O(\Delta t^2)$.

4.3 Extrapolating the Time Derivative

We previously discussed how we would evaluate the time derivative implicitly. This can be done using the Adams-Moulton method, which is the implicit counterpart to the Adams-Bashforth method. Through much the same method as the previous section we apply the Taylor series expansion, however this time we wish to preserve the time derivative term. The Adams-Moulton method for approximating the time derivative requires 3 time-steps, as shown by [3], owing to the factor of Δt in front of the time derivative in the Taylor series. Expanding ω^t , $\omega^{t-\Delta t}$ and $\omega^{t-2\Delta t}$ as Taylor series yields:

$$\omega^t = \omega^t \quad (4.12)$$

$$\omega^{t-\Delta t} = \omega^t + \frac{(-1)^1 \Delta t}{1!} \frac{\partial \omega^t}{\partial t} + \frac{(-1)^2 \Delta t^2}{2!} \frac{\partial^2 \omega^t}{\partial t^2} + O(\Delta t^3) \quad (4.13)$$

$$\omega^{t-2\Delta t} = \omega^t + \frac{(-2)^1 \Delta t}{1!} \frac{\partial \omega^t}{\partial t} + \frac{(-2)^2 \Delta t^2}{2!} \frac{\partial^2 \omega^t}{\partial t^2} + O(\Delta t^3) \quad (4.14)$$

where the first expansion is simply ω^t , thus making the scheme implicit. We again arrange ω^t , $\omega^{t-\Delta t}$ and $\omega^{t-2\Delta t}$ as a linear combination of the above expansions:

$$\begin{aligned} \alpha_0 \omega^t + \alpha_1 \omega^{t-\Delta t} + \alpha_2 \omega^{t-2\Delta t} &= (\alpha_0 + \alpha_1 + \alpha_2) \omega^t - \frac{(\alpha_1 + 2\alpha_2) \Delta t}{1!} \frac{\partial \omega^t}{\partial t} \\ &\quad + \frac{(\alpha_1 + 4\alpha_2) \Delta t^2}{2!} \frac{\partial^2 \omega^t}{\partial t^2} + O(\Delta t^3). \end{aligned} \quad (4.15)$$

We wish to preserve the time derivative term, whilst the ω^t and second time derivative terms vanish, thus we choose $\alpha_0 + \alpha_1 + \alpha_2 = 0 = \alpha_1 + 4\alpha_2$ and $\alpha_1 + 2\alpha_2 = -1$ [3]. Solving yields $\alpha_0 = \frac{3}{2}$, $\alpha_1 = -2$, $\alpha_2 = \frac{1}{2}$, substitution of these coefficients into (4.15) and division by Δt gives the second order Adams-Moulton scheme:

$$\frac{1}{\Delta t} \left(\frac{3}{2} \omega^t - 2 \omega^{t-\Delta t} + \frac{1}{2} \omega^{t-2\Delta t} \right) = \frac{\partial \omega^t}{\partial t} + O(\Delta t^2) \quad (4.16)$$

which approximates $\partial \omega^t / \partial t$ with error $O(\Delta t^2)$.

Chapter 5

Simulation Methods for the Kolmogorov System

It is now time to integrate spectral methods, and time stepping techniques, to create an iterative solver for evolving the Kolmogorov system. The implementation of our solver took place in Python 3.14.6, and made use of the CuPY package for GPU accelerated computing.

5.1 Kolmogorov Spectral Scheme

We must now derive a suitable scheme that evolves the Kolmogorov system with time. We start by considering the Kolmogorov system on the rectangular domain of extent Γ_x and Γ_y in the x and y directions. The equations describing the evolution of the vorticity field on this domain read:

$$\frac{\partial \omega}{\partial t} + J(\psi, \omega) = \frac{1}{Re} \nabla^2 \omega + F(y) \quad (5.1)$$

$$\omega = -\nabla^2 \psi \quad (5.2)$$

$$J(\psi, \omega) = \frac{\partial \psi}{\partial y} \frac{\partial \omega}{\partial x} - \frac{\partial \psi}{\partial x} \frac{\partial \omega}{\partial y} \quad (5.3)$$

$$F(y) = -\frac{1}{Re} \cos(y) \quad (5.4)$$

where $J(\psi, \omega)$ and $F(y)$ have been defined separately to simplify the form of (5.1) and (5.2). We discretise these equations spatially on the 2D periodic grid:

$$\begin{aligned} x_j &= \frac{j\Gamma_x}{N_x}, \quad j = 0, \dots, N_x - 1 \\ y_\ell &= \frac{\ell\Gamma_y}{N_y}, \quad \ell = 0, \dots, N_y - 1. \end{aligned} \quad (5.5)$$

The discretised vorticity on the grid (5.5) reads:

$$\begin{aligned} \omega(x_j, y_\ell) &= \sum_{n=-N_y/2+1}^{N_y/2} \exp\left(in\ell \frac{2\pi}{N_y}\right) \left(\hat{\omega}_{0,n} \dots \right. \\ &\quad \left. + \sum_{m=0}^{N_x/2} \left[\hat{\omega}_{m,n} \exp\left(imj \frac{2\pi}{N_x}\right) + \hat{\omega}_{m,n}^* \exp\left(-imj \frac{2\pi}{N_x}\right) \right] + \hat{\omega}_{N_x/2,n} (-1)^j \right) \end{aligned} \quad (5.6)$$

with $m = 0, \dots, N_x - 1$ and $n = -N_y/2 + 1, \dots, N_y/2$. We will ignore (5.2) for the remainder of this section, as it is not yet required, thus applying the DFT to equation (5.1) yields:

$$\frac{\partial \hat{\omega}_{m,n}}{\partial t} + \hat{J}_{m,n} = -\frac{K_{m,n}}{Re} \hat{\omega}_{m,n} + \hat{F}_n \quad (5.7)$$

where $K_{m,n} = 4\pi^2(m^2/\Gamma_x^2 + n^2/\Gamma_y^2)$. We now discretise the system temporally, on the grid:

$$t_c = c\Delta t, \quad c = 0, \dots, T \quad (5.8)$$

where T is the maximum time. By evaluating the system at running time t , and applying the Adams-Moulton approximation for the time derivative (4.16) we obtain:

$$\frac{1}{\Delta t} \left(\frac{3}{2} \hat{\omega}_{m,n}^t \hat{H}_{m,n}^t \right) + \hat{J}_{m,n}^t = -\frac{K_{m,n}}{Re} \hat{\omega}_{m,n}^t + \hat{F}_n \quad (5.9)$$

where $\hat{H}_{m,n}^t = -2\hat{\omega}_{m,n}^{t-\Delta t} + (1/2)\hat{\omega}_{m,n}^{t-2\Delta t}$, and $\hat{J}_{m,n}^t = 2\hat{J}_{m,n}^{t-\Delta t} - \hat{J}_{m,n}^{t-2\Delta t}$. Finally, rearranging (5.9) to make $\hat{\omega}_{m,n}^t$ the subject finally yields the spectral time stepping scheme:

$$\hat{\omega}_{m,n}^t = \frac{2}{3} \left(1 + \frac{2K_{m,n}}{3Re} \right)^{-1} \left([\hat{F}_n - \hat{J}_{m,n}^t] \Delta t - \hat{H}_{m,n}^t \right) \quad (5.10)$$

which evolves the vorticity field of the Kolmogorov system in spectral space.

5.2 The Non-Linear Advection Term

Using (5.10) we can evolve the system from time $t - \Delta t$ to t , we recall how this requires the extrapolation of the advection term $\hat{J}_{m,n}^t$. If we were to reuse this extrapolation to obtain $\hat{J}_{m,n}^{t+\Delta t}$ the error of this approximation would compound, and the dynamics of the system would be obscured. We instead use $\hat{\omega}_{m,n}^t$, obtained using (5.10), to calculate $\hat{J}_{m,n}^t$ which can be used to extrapolate the next time step. We start by discretising (5.3) spatially on the grid (5.5) and temporally on (5.8):

$$J_{j,\ell}^t = \frac{\partial \psi_{j,\ell}^t}{\partial y} \frac{\partial \omega_{j,\ell}^t}{\partial x} - \frac{\partial \psi_{j,\ell}^t}{\partial x} \frac{\partial \omega_{j,\ell}^t}{\partial y} \quad (5.11)$$

We can re-write the partial derivatives as the IDFT of the spectral derivatives:

$$J_{j,\ell}^t = \mathcal{F}^{-1}(i\beta_n \hat{\psi}_{m,n}^t) \mathcal{F}^{-1}(i\alpha_m \hat{\omega}_{m,n}^t) - \mathcal{F}^{-1}(i\alpha_m \hat{\psi}_{m,n}^t) \mathcal{F}^{-1}(i\beta_n \hat{\omega}_{m,n}^t) \quad (5.12)$$

where $(k_x)_m = 2\pi m/\Gamma_x$ and $(k_y)_n = 2\pi n/\Gamma_y$ represent physical wave numbers in x and y respectively. has been introduced to separate the positional subscript from . Discretising (5.2) and applying the DFT yields $\hat{\omega}_{m,n} = K_{m,n} \hat{\psi}_{m,n}$, which can be rearranged to obtain:

$$\hat{\psi}_{m,n}^t = \frac{\hat{\omega}_{m,n}^t}{K_{m,n}}. \quad (5.13)$$

Here division by $K_{m,n}$ in spectral space is equivalent to integration in physical space, where $k_{0,0} = 0$ is representative of the constant mode. Since $\omega_{m,n}^t$ is periodic, and only defined up to a constant in physical space, we can set $\hat{\psi}_{0,0}^t = 0$ without loss of information. Substituting (5.13) into (5.12) yields:

$$J_{j,\ell}^t = \mathcal{F}^{-1}\left(i\frac{\beta_n}{K_{m,n}}\hat{\omega}_{m,n}^t\right) \mathcal{F}^{-1}(i\alpha_m \hat{\omega}_{m,n}^t) - \mathcal{F}^{-1}\left(i\frac{\alpha_m}{K_{m,n}}\hat{\omega}_{m,n}^t\right) \mathcal{F}^{-1}(i\beta_n \hat{\omega}_{m,n}^t) \quad (5.14)$$

where $J_{j,\ell}^t$ becomes dependent on only $\hat{\omega}_{m,n}^t$. Taking the DFT of (5.14) yields the spectral approximation for advection:

$$\hat{J}_{m,n}^t = \mathcal{F}\left[\mathcal{F}^{-1}\left(i\frac{\beta_n}{K_{m,n}}\hat{\omega}_{m,n}^t\right) \mathcal{F}^{-1}(i\alpha_m \hat{\omega}_{m,n}^t) - \mathcal{F}^{-1}\left(i\frac{\alpha_m}{K_{m,n}}\hat{\omega}_{m,n}^t\right) \mathcal{F}^{-1}(i\beta_n \hat{\omega}_{m,n}^t)\right]. \quad (5.15)$$

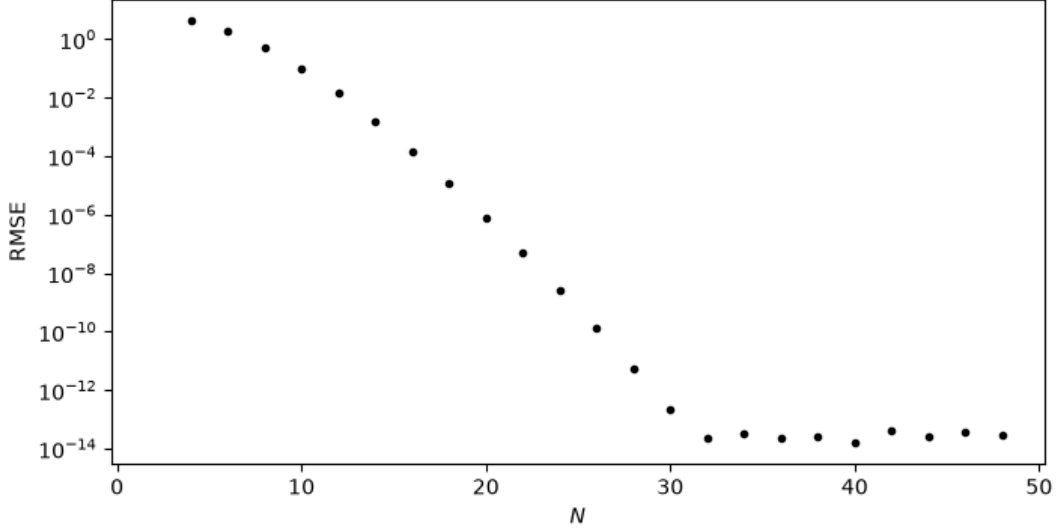


Figure 5.1: Scatter plot showing the Root Mean Squared Error (RMSE) of approximation (5.15), for the test function $\psi = \exp(\cos x + \cos y)$, at different grid resolutions $N = N_x = N_y$. Results computed and plotted on a logarithmic RMSE axis in Python (Appendix A.4.3).

This spectral scheme was implemented as the function “`advection()`” in Python (Appendix A.4.1). Figure 5.1 shows the RMSE of the approximation (5.15) for the test function $\psi = \exp(\cos x + \cos y)$. It is clear that the scheme exhibits spectral accuracy, thus “`advection()`” is validated.

Since the advection term has a non-linearity of order 2 we must apply Fourier filtering to prevent aliasing, this entails removal of modes with wave number $m > N_x/3$ and $|n| > N_y/3$. This was achieved numerically through the use of “`aliasing_mask()`” which creates a mask that can be applied to the vorticity field in spectral space, to remove unwanted modes.

5.3 Simulation Approach

We will now discuss the simulation of the Kolmogorov system. We first consider our simulated domain $x, y \in [0, \Gamma_x] \times [0, \Gamma_y]$. The physical wave numbers in the x -direction take the form $(k_x)_m = 2\pi m/\Gamma_x$, $m = 0, \dots, N_x - 1$, on the grid (5.5), thus if $\Gamma_x < 2\pi$ then $(k_x)_m > 1$ for all m . We recall from linear stability that only modes with a physical wave number $(k_x)_m < 1$ thus we need a grid with $\Gamma_x > 2\pi$ to properly represent unstable modes. We also consider that the periodicity of the Kolmogorov system is 2π in the y direction, thus we can choose to decompose $\Gamma_x = 2\pi a_x$, $\Gamma_y = 2\pi a_y$, where $a_x, a_y \in \mathbb{Z}^+$ scale the domain by an integer multiple

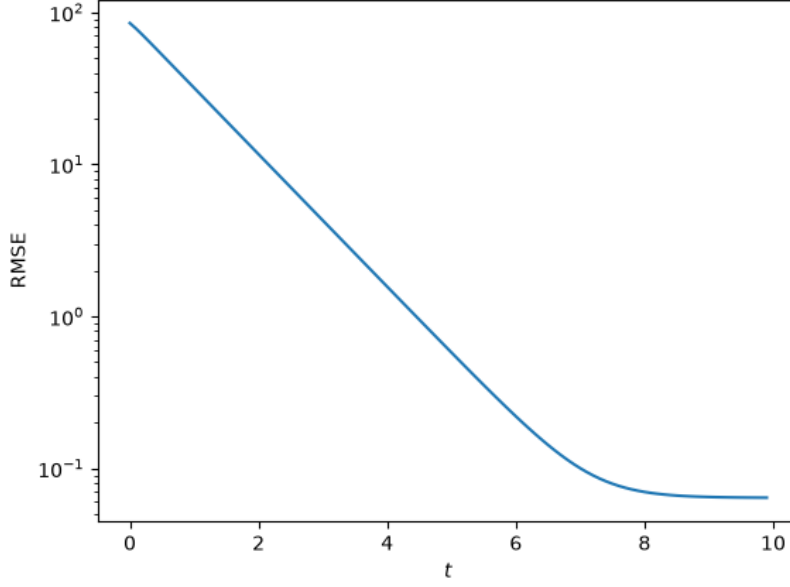


Figure 5.2: Root Mean Squared Error (RMSE) of Kolmogorov system velocity field with respect to the steady state field $\omega_s = -\cos(y)$, as the system evolves with time, plotted on a logged RMSE axis. Vorticity field evolved using “`Kolmogorov_simulation()`” with $N = 128, a_x = 1, a_y = 1, Re = 1, \Delta t = 0.1, t_{max} = 10$ with an initial `state=False`. Computed and plotted on a logged RMSE axis in python (Appendix A.4.5).

in x and y . The results section will investigate different domain shapes and sizes, characterised by a_x and a_y , in order to observe the unstable growth of different modes. We will additionally introduce N as the grid resolution of a $[0, 2\pi]$ region on our main grid. By choosing $N_x = a_x N$ and $N_y = a_y N$ we can keep the spacing of grid points in the x and y directions consistent. We also introduce the notation (a_x, a_y) to denote the domain shape, so for instance a domain with $a_x = 2, a_y = 1$ would become $(2, 1)$.

We now introduce “`Kolmogorov_simulation()`”, created in Python (Appendix A.4.4), which iteratively solves the vorticity field of the Kolmogorov system. “`Kolmogorov_simulation()`” takes inputs $N, Re, t_{max}, \Delta t, a_x, a_y, \text{state}$ as simulation parameters, where t_{max} is the length of the simulation, and `state` initialises the system from steady state if **True** or a grid with no initial vorticity if **False**. The function takes the additional parameters `frame` and `save_file`, where `frame` is the number of time steps between saving the vorticity field as a `.npy` file, and `save_file` tells the program where to save the vorticity field. Figure 5.2 shows the RMSE of the vorticity field with respect to the steady state $\omega_s = -\cos y$, evolved from a state with no initial vorticity. The RMSE decreases linearly on the logged axis, which is indicative of

exponential decay .The vorticity field must converge towards the steady state as the RMSE decreases, thus the solver “`Kolmogorov_simulation()`” corroborates the standard steady state result for Kolmogorov flow. The choice of $Re = 1$, in addition to no perturbations, guarantees the Kolmogorov flow steady state to remain stable.

Chapter 6

Results

This chapter will make use of `Kolmogorov_simulation()` (Appendix A.4.4) to simulate the Kolmogorov system for Reynolds numbers where unstable perturbations are allowed to grow. We are interested in the growth of unstable perturbations from the steady state of the Kolmogorov system, thus for this section all simulations will start from $\omega_s = -\cos y$, hence `state` will be set to `True` for `Kolmogorov_simulation()` throughout this chapter. The results in this chapter will focus heavily on the unstable growth exhibited by the Kolmogorov system on a range of domain shapes, with particular interest in the transition of a system to turbulence.

6.1 Diagnostic Quantities

Before discussing the results of simulating Kolmogorov flow, we must first define the following quantities that will be useful for understanding our simulation output, where $\tilde{\omega}_{j,\ell}^t$ is the perturbation of the vorticity field from the steady state, and $\tilde{u}_{j,\ell}^t$, $\tilde{v}_{j,\ell}^t$ represent the velocity field perturbations:

$$\tilde{\zeta}^t = \frac{1}{2N_x N_y} \sum_{\ell=0}^{N_y-1} \sum_{j=0}^{N_x-1} (\tilde{\omega}_{j,\ell}^t)^2 \quad (6.1)$$

$$\tilde{\Delta}^t = \frac{1}{N_x N_y} \sum_{\ell=0}^{N_y-1} \sum_{j=0}^{N_x-1} \left| \frac{\tilde{\omega}_{j,\ell}^t - \tilde{\omega}_{j,\ell}^{t-\Delta t}}{\Delta t} \right|. \quad (6.2)$$

The mean perturbation enstrophy, denoted by $\tilde{\zeta}^t$, is the enstrophy of the vorticity field perturbation averaged over every point on the spatial grid. Recall from stability analysis that $\tilde{\omega}_{j,\ell}^t \propto e^{\lambda t}$, thus we can make the additional observation that $\tilde{\zeta}^t \propto e^{2\lambda t}$, which can be used to

validate the unstable mode selected by the system.

The final quantity is the mean absolute time derivative, denoted as $\tilde{\Delta}^t$, which is the absolute value of the time derivative averaged over every grid point, this is essential illustrating the convergence of the vorticity field towards a steady state.

A python script “`perturbation_energy.py`” was then developed (Appendix A.5.2) which returns $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ at every time-step for given simulation data generated by “`Kolmogorov_simulation()`”. Additionally the growth rates for unstable modes have been calculated in this chapter using `point_growth()` (Appendix A.1.5).

6.2 The Trivial Domain (1, 1)

As we begin to explore the unstable behaviour of the Kolmogorov system on different domain shapes, we must first consider the most basic shape (1, 1), which is a square $[0, 2\pi]^2$ domain. We recall how the discretised system is represented as a set of modes with physical wave number $k_x = 2\pi m/\Gamma_x$ and $k_y = 2\pi n/\Gamma_y$, additionally $\Gamma_x = \Gamma_y = 2\pi$ for (1, 1), thus the physical wave numbers k_x, k_y must be whole numbers. Finally we recall that the system must have $k_x \in (0, 1)$ to become unstable, and since physical wave numbers in this region are not represented on (1, 1) we can conclude that instability cannot be simulated on this grid. This result is corroborated by [11] who states that a square domain on $[0, 2\pi]^2$ experiences the flow experiences global stability (or anti-turbulence) for all forcing amplitudes.

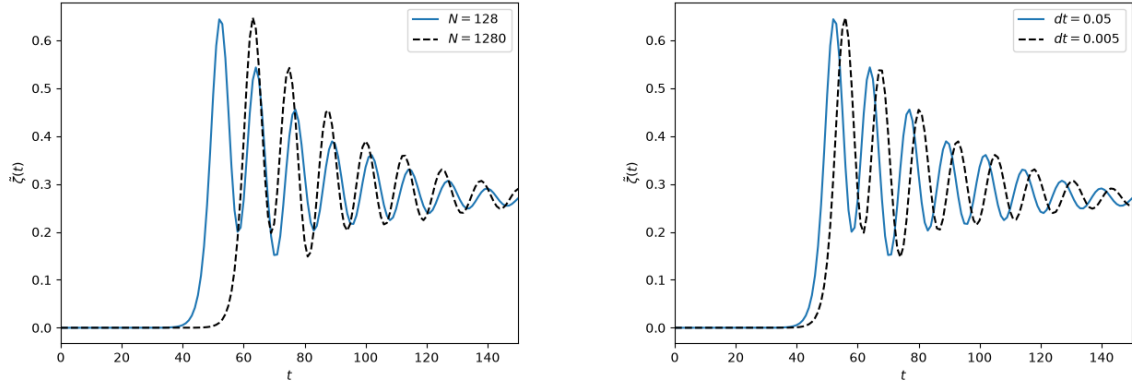
6.3 Simulation on (2, 1)

We now begin discussing the results of the Kolmogorov system on a range of domain shapes. We begin by considering the simplest domain that can produce instability, (2, 1). The simulations in this section utilise a time step $\Delta t = 0.5/Re$, and a resolution $N = 128$, making the grid a total 256×128 points. We now consider the discretisation of the physical wave number k_x in spectral space, $(k_x)_m = 2\pi m/\Gamma_x$ where $m = 0, \dots, N_x - 1$. Since only modes with $k_x \in (0, 1)$ can become unstable, and the only mode represented in discrete spectral space with a k_x in the unstable region occurs for $m = 1$, we surmise that the only unstable mode has a physical wave number $k_x = 1/2$.

We can check the suitability of our spatio-temporal mesh by conducting a mesh convergence

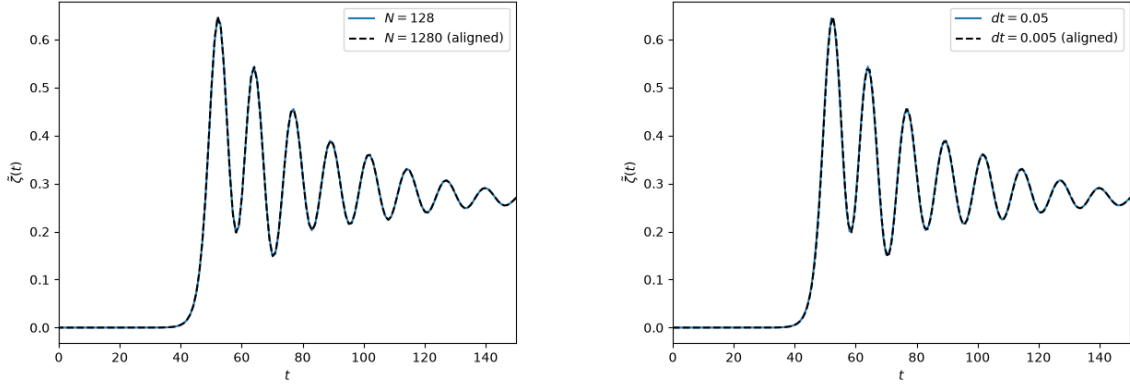
study. The largest Reynolds number we will work with in this section is $Re = 100$, thus we must validate the mesh for $Re = 100$. Figure 6.1 compares the enstrophy of the Kolmogorov system for distinct spatial and temporal spacings.

The sub-figure 6.1a shows the enstrophy for $N = 128$ and $N = 1280$, the shape of the plots appears identical, however $N = 1280$ appears offset from $N = 128$ on the t axis. This stems from unstable modes taking longer to initially propagate through higher resolution grids. Sub-figure 6.1b similarly shows the enstrophy for $\Delta t = 0.005$ and $\Delta t = 0.0005$, where the enstrophy values are offset again on the t axis. The choice of $\Delta t = 0.05$ is the result of choosing $\Delta t = 0.5/Re$, where $Re = 100$.



(a) Plot showing $\tilde{\zeta}^t$ for 2 distinct resolutions, $N = 128$ and $N = 1280$, with $\Delta t = 1/Re$ and $Re = 20$. (b) Plot showing $\tilde{\zeta}^t$ for 2 distinct time step sizes, $\Delta t = 0.05$ and $\Delta t = 0.005$, with $N = 128$ and $Re = 20$.

Figure 6.1: Plots showing $\tilde{\zeta}^t$ over time for different spatial and temporal meshes. Calculated and plotted using Python (Appendix A.5.1).



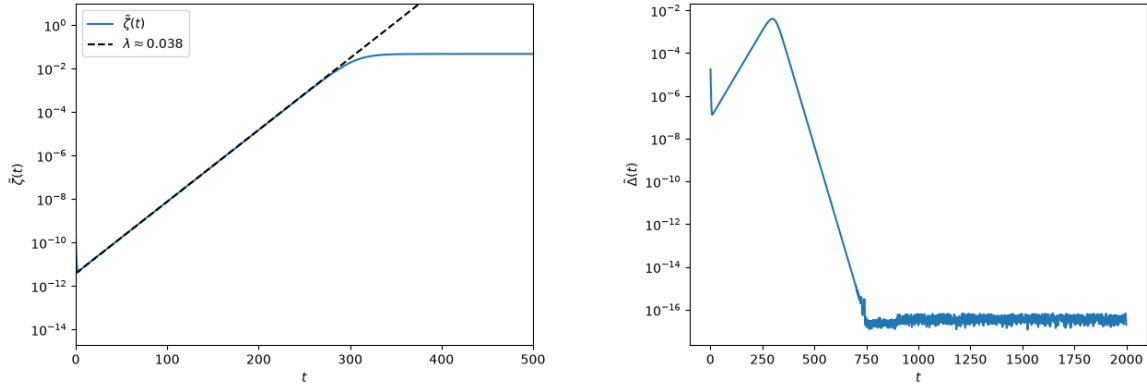
(a) Plot showing $\tilde{\zeta}^t$ for 2 distinct resolutions, $N = 128$ and $N = 1280$, with $\Delta t = 1/Re$ and $Re = 20$, where a temporal offset of 10.81 has been applied. (b) Plot showing $\tilde{\zeta}^t$ for 2 distinct time step sizes, $\Delta t = 0.05$ and $\Delta t = 0.005$, with $N = 128$ and $Re = 20$, where a temporal offset of 3.4 has been applied.

Figure 6.2: Plots showing $\tilde{\zeta}^t$ over time for different spatial and temporal meshes. Calculated and plotted using Python (Appendix A.5.1).

We can show that the plots are qualitatively identical, whilst experiencing a temporal offset in the onset of instability. Figure 6.2 shows plots of the same $\tilde{\zeta}^t$ with a temporal offset applied. The plot of $\tilde{\zeta}^t$ in sub-figure 6.2a for $N = 1280$ has been temporally offset by 10.81, and the plot of $\tilde{\zeta}^t$ in sub-figure 6.2b has been temporally offset by 3.4. The plots now appear completely aligned, which suggests that the qualitative results are relatively unaffected by mesh resolution. A more thorough mesh convergence study would help to obtain more appropriate mesh resolution, however for the purposes of understanding unstable growth from Kolmogorov flow the mesh $N = 128, \Delta t = 0.5/Re$ is satisfactory.

Instability on $(2, 1)$ with $Re = 2.5$

We will now discuss the results of running “Kolmogorov_simulation()” on the grid $(2, 1)$, with $N = 128$ and $\Delta t = 0.5/Re$. We start with a low Reynolds number $Re = 2.5$, as to observe instability first on a smaller scale. The sub-figure 6.3a shows $\tilde{\zeta}^t$ for the Kolmogorov system on $(2, 1)$ with $Re = 2.5$. The system initially becomes unstable at $t = 0$ when the system is perturbed, where the slope of $\tilde{\zeta}^t$ is visually identical to the tangent representing the expected growth for $\lambda = 0.038\dots$, thus the perturbations grow predictably. After $t = 300$ the growth of the perturbation stops, and the overall $\tilde{\zeta}^t$ of the system appears steady.



(a) Plot showing $\tilde{\zeta}(t)$ for the Kolmogorov system simulated on $(2, 1)$ with $Re = 2.5$ and $N = 128$, evolved with time. The black dashed line is a tangent with gradient 2λ , representing the expected growth rate for $\lambda = 0.038$. Results plotted for $t \in [0, 500]$ on a logged vertical axis using Python (Appendix A.5.4).

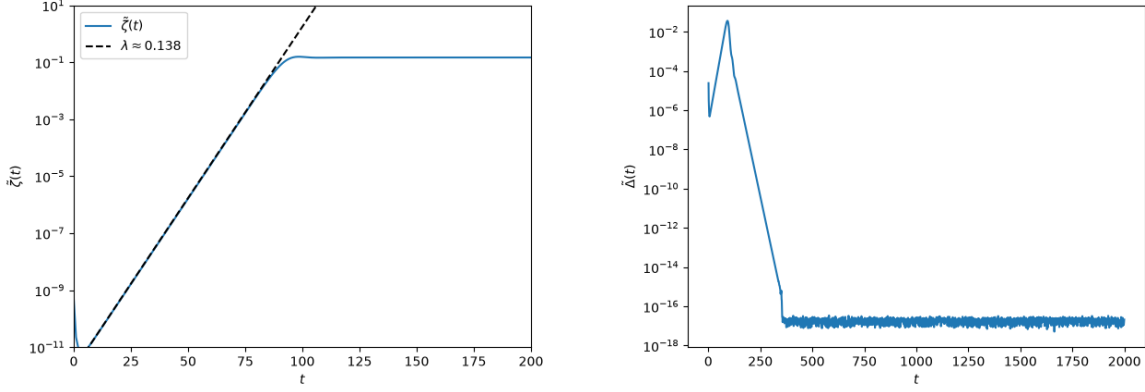
(b) Plot showing $\tilde{\Delta}(t)$ for the Kolmogorov system simulated on $(2, 1)$, with $N = 128$ and $Re = 2.5$, evolved with time. Results plotted for $t \in [0, 500]$ on a logged vertical axis in Python.

Figure 6.3: Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on $(2, 1)$, with $N = 128$ and $Re = 2.5$. Calculated in Python using `perturbation_energy.py` and plotted with code in Appendix A.5.4.

Through examination of sub-figure 6.3b we see that $\tilde{\Delta}^t$ initially increases linearly on the logged axis, which is indicative of the exponential growth. We stipulated that $\tilde{\zeta}^t$ becomes steady after $t = 300$, we can confirm that the vorticity field also nears a steady state at this time, since $\tilde{\Delta}^t$ decreases linearly on the logged vertical axis. The linear decrease of $\tilde{\Delta}^t$ on the logged vertical axis is indicative of exponential decay, thus the change in the system between time steps slows exponentially. We can surmise that the unstable flow approaches a stable steady state.

Instability on $(2, 1)$ with $Re = 5$

We can discuss solutions on $(2, 1)$ for a slightly larger Reynolds number, $Re = 5$. Sub-figure 6.4a shows $\tilde{\zeta}^t$ for the Kolmogorov system on $(2, 1)$ for $Re = 5$. The instability is initialised at $t = 0$, The system initially becomes unstable at approximately $t = 0$ when the system is perturbed, and the slope of $\tilde{\zeta}^t$ is visually identical to the tangent representing the expected gradient for $\lambda = 0.138\dots$, thus the system grows predictably. Sub-figure 6.4b shows $\tilde{\Delta}^t$ decreasing linearly on the logged vertical axis after $t = 75$, thus we conclude that the system has again reached a stable steady state.



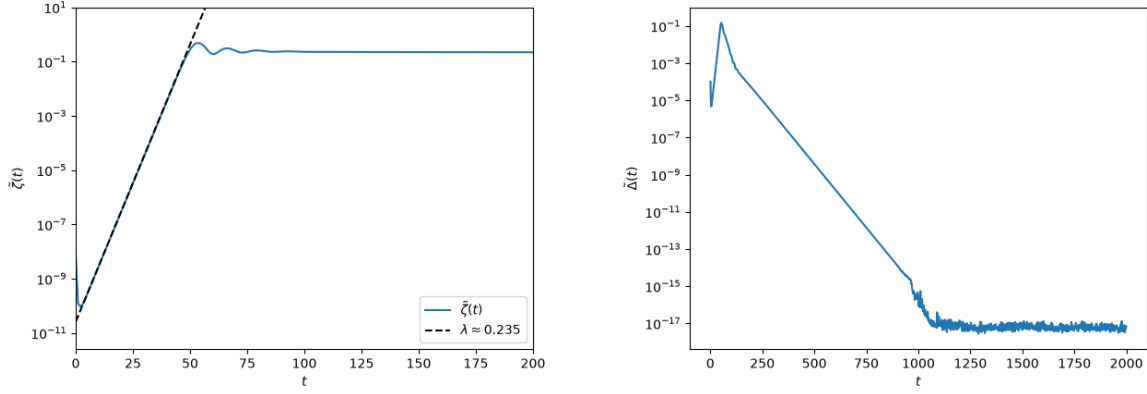
(a) Plot showing $\tilde{\zeta}(t)$ for the Kolmogorov system simulated on $(2, 1)$ with $Re = 5$ and $N = 128$, evolved with time. The black dashed line is a tangent with gradient 2λ , representing the expected growth rate for $\lambda = 0.138$ predicted by linear stability analysis. Results plotted for $t \in [0, 200]$ on a logged vertical axis using Python.

(b) Plot showing $\tilde{\Delta}(t)$ for the Kolmogorov system simulated on $(2, 1)$, with $N = 128$ and $Re = 5$, evolved with time. Results plotted for $t \in [0, 500]$ on a logged vertical axis in Python.

Figure 6.4: Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on $(2, 1)$, with $N = 128$ and $Re = 5$. Calculated in Python using `perturbation_energy.py`, and plotted with code in Appendix A.5.4.

6.3.1 Instability on $(2, 1)$ with $Re = 25$

We now consider the evolution of the Kolmogorov system on $(2, 1)$ for a higher Reynolds number, $Re = 25$, where we expect the growth rate of instability to be much greater. Sub-figure 6.5a shows $\tilde{\zeta}^t$ for the Kolmogorov system on $(2, 1)$ for $Re = 25$. The instability is initialised at $t = 0$, here the slope of $\tilde{\zeta}^t$ visually matches the tangent representing expectation from linear stability analysis, $\lambda = 0.235 \dots$. At approximately $t = 50$ the growth of the unstable perturbation peaks, however unlike for the $Re = 2.5$ flow, $\tilde{\zeta}^t$ begins to exhibit a dampened oscillatory behaviour. These dampened oscillations represent circulation around a stable steady state, which suggests that there might be Reynolds numbers for which $\tilde{\zeta}^t$ orbits the steady state indefinitely, or even has an orbit which grows. Examining sub-figure 6.5b we again see that $\tilde{\Delta}^t$ decreases linearly on the logged vertical axis, thus we surmise that the vorticity field for $Re = 25$ approaches a steady state.



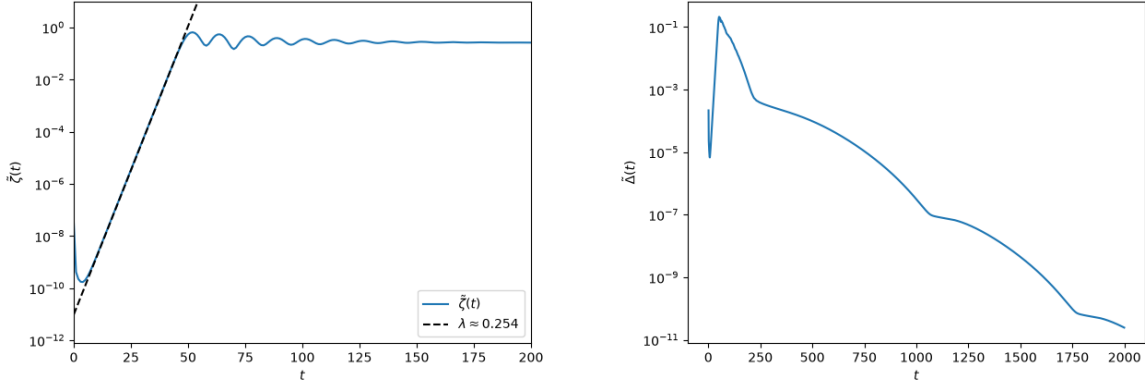
(a) Plot showing $\tilde{\zeta}(t)$ for $Re = 25$ on $(2, 1)$ with $N = 128$, evolved with time. The black dashed line is a tangent with gradient 2λ , representing the expected growth rate for $\lambda = 0.235$ predicted by linear stability analysis. Plotted on a logged vertical axis using Python.

(b) Plot showing $\tilde{\Delta}(t)$ for $Re = 25$ on $(2, 1)$ with $N = 128$, evolved with time. Plotted on a logged vertical axis in Python.

Figure 6.5: Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on $(2, 1)$, with $N = 128$ and $Re = 25$. Calculated in Python using `perturbation_energy.py`, and plotted with code in Appendix A.5.4.

6.3.2 Instability on $(2, 1)$ with $Re = 100$

We finally discuss solutions on $(2, 1)$ for a much higher Reynolds number, $Re = 100$. Sub-figure 6.6a shows $\tilde{\zeta}^t$ for Kolmogorov system on $(2, 1)$ with $Re = 100$. The system initially becomes unstable at $t = 0$, where the slope of $\tilde{\zeta}^t$ visually matches the tangent representing expectation from linear stability analysis, $\lambda = 0.255 \dots$. At $t = 50$ the growth of the unstable perturbation peaks, and similarly to the $Re = 25$ case $\tilde{\zeta}^t$ begins to oscillate. The oscillations around the steady state appear to have a higher amplitude, and take significantly longer to become dampened. We can see from sub-figure 6.6b that $\tilde{\Delta}^t$ begins to decrease linearly on the logged vertical axis from $t = 100$ to $t = 200$. Once the system evolves to $t = 200$ we find that $\tilde{\Delta}^t$ continues to decrease, however it does not do so linearly, as a result this system still approaches a steady state, but it does not do so in the linear regime.



(a) Plot showing $\tilde{\zeta}(t)$ for $Re = 100$ on $(2, 1)$ with $N = 128$, evolved with time. The black dashed line is a tangent with gradient 2λ , representing the expected growth rate for $\lambda = 0.255$ predicted by linear stability analysis. Plotted on a logged vertical axis using Python.

(b) Plot showing $\tilde{\Delta}(t)$ for $Re = 100$ on $(2, 1)$ with $N = 128$, evolved with time. Plotted on a logged vertical axis in Python.

Figure 6.6: Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on $(2, 1)$, with $N = 128$ and $Re = 100$. Calculated in Python using `perturbation.energy.py`, and plotted with code in Appendix A.5.4.

6.3.3 Flow Reversal on (2, 1)

We have discussed the evolution of unstable perturbations to the Kolmogorov system for $Re = 2.5, 5, 25, 100$, where it was found that for each of these Reynolds numbers the solutions on $(2, 1)$ converge to individual steady states as time evolved. We now consider the vorticity field as a whole, rather than just its perturbations. Figure 6.7 shows the vorticity fields of the Kolmogorov system for $Re = 2.5, 5, 25, 100$ at $t = 1999$. The vorticity fields here are nearing the steady state, as predicted by our analysis of $\tilde{\Delta}^t$. We see that as the Reynolds number increases the vorticity field separates into a positive and a negative vortex, where further increasing the Reynolds number begins to rotate the vortices into the orthogonal direction. Examining the plot for $Re = 100$ we see that the vorticity field has almost fully rotated, such that it forms bands of vorticity that span the y axis. The vorticity of Kolmogorov flow is defined as $\omega_s = -\cos y$, which appears as stripes of vorticity stretching across the x axis. The resulting rotation of the vorticity field for higher Reynolds numbers suggests that the direction of the flow may also be changing. We define the following quantity:

$$\Theta = \frac{\sum_{\ell=0}^{N_y-1} \sum_{j=0}^{N_x-1} (v_{j,\ell}^t)^2}{\sum_{\ell=0}^{N_y-1} \sum_{j=0}^{N_x-1} (u_{j,\ell}^t)^2} \quad (6.3)$$

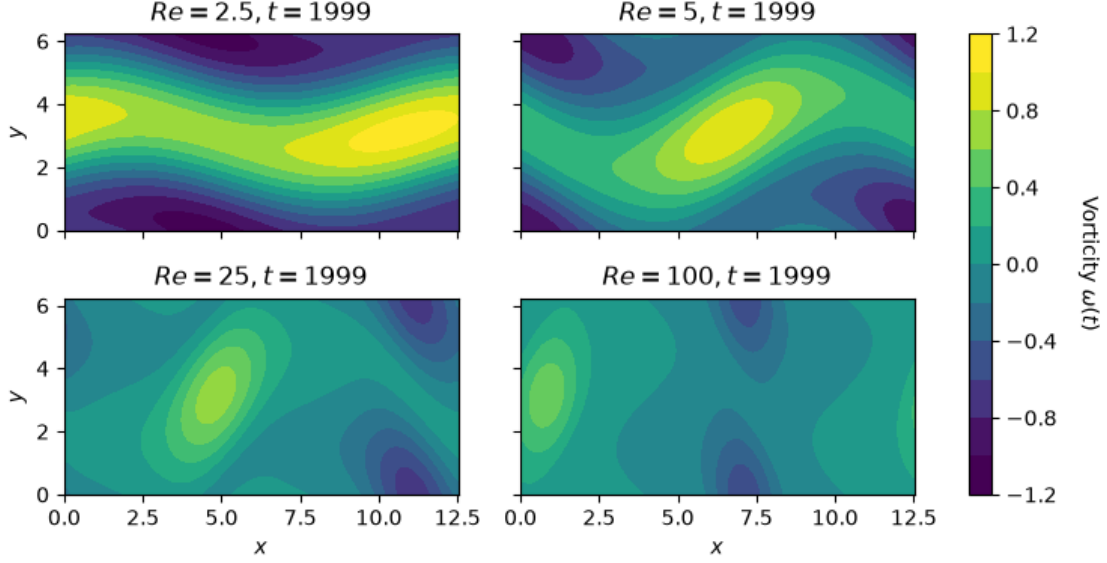


Figure 6.7: Contourf plots of the vorticity field of $(2, 1)$, for $Re = 2.5, 5, 25, 100$ at $t = 1999$. Plotted in Python (Appendix A.5.5).

where Θ is a ratio of the kinetic energies in the y and x directions, which we will refer to as the directional energy. Table 6.1 shows Θ computed for each of the Reynolds numbers in figure 6.7 at $t = 1999$.

Re	2.5	5	25	100
Θ	0.251...	1.518...	9.840...	27.763...

Table 6.1: Directional energy Θ for different Reynolds numbers at $t = 1999$, output from Python (Appendix A.5.3).

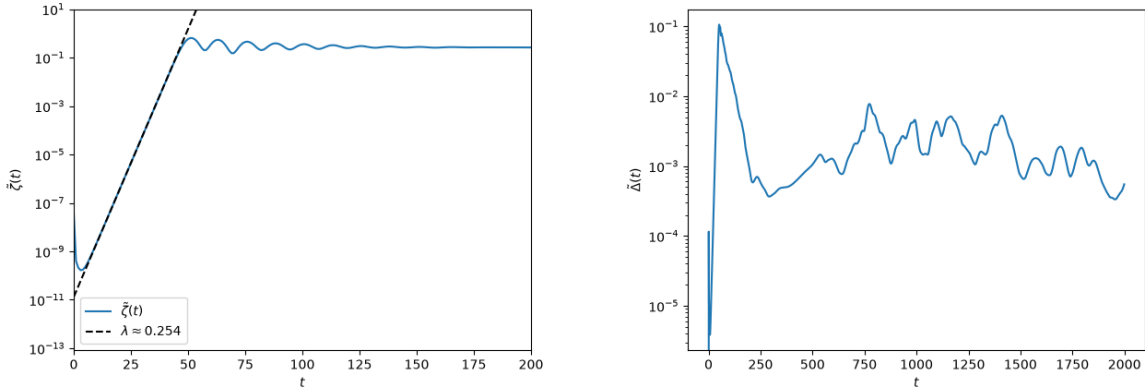
It is clear that Θ increases with the Reynolds number. $Re = 100$ the directional energy $\Theta = 27.763\dots$, which indicates that the flow of the fluid for $Re = 100$ is primarily in the y direction. We see that the flow reverses more severely with higher Reynolds number then, whilst outside the scope of this project, further investigation could be conducted to investigate flow reversal for increasingly higher Reynold numbers.

6.4 Simulation on $(2, 2)$

The logical progression from $(2, 1)$ is to the square domain $(2, 2)$. The unstable modes for this domain size are exactly the same as for $(2, 1)$, thus we will choose not to repeat the study of lower Reynolds number flows, and will instead investigate $Re = 100$ on the domain $(2, 2)$ in pursuit of compelling results.

6.4.1 Instability and Turbulence on (2, 2) for $Re = 100$

Sub-figure 6.6a shows $\tilde{\zeta}^t$ for Kolmogorov system on (2, 2) with $Re = 100$. The system initially becomes unstable at $t = 0$, where the slope of $\tilde{\zeta}^t$ visually matches the tangent representing expectation from linear stability analysis, $\lambda = 0.255 \dots$. In much the same way as for the (2, 1) case $\tilde{\zeta}^t$ begins to oscillate after $t = 50$. If we now turn our attention to sub-figure 6.8b we see that instead of $\tilde{\Delta}^t$ decaying towards 0, like the (2, 1) case, the plot instead appears complex and unpredictable.



(a) Plot showing $\tilde{\zeta}(t)$ for $Re = 100$ on (2, 2) with $N = 128$, evolved with time. The black dashed line is a tangent with gradient 2λ , representing the expected growth rate for $\lambda = 0.255$ predicted by linear stability analysis. Plotted on a logged vertical axis using Python.

(b) Plot showing $\tilde{\Delta}(t)$ for $Re = 100$ on (2, 2) with $N = 128$, evolved with time. Plotted on a logged vertical axis in Python.

Figure 6.8: Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on (2, 2), with $N = 128$ and $Re = 100$. Calculated in Python using `perturbation.energy.py`, and plotted with code in Appendix A.5.6.

In order to better understand the fluctuations in $\tilde{\Delta}^t$ we can visualise the vorticity field at the time where the complexity begins. Figure 6.9 shows the vorticity field for the Kolmogorov system on (2, 2) for $t = 50, 75, 100, 1000$. We see that at $t = 50$ the vorticity appears in bands of vorticity, where it appears to be 4π periodic along x , which is indicative of growth resulting from the unstable mode with $k_x = 1/2$. As the system evolves to $t = 75$ the bands of vorticity appear to break up into smaller, lower magnitude vortices. At $t = 100$ the pattern remains similar to $t = 75$, except the magnitude of the vorticity appears even more subdued. Finally at $t = 1000$ the vorticity field appears spatially complex. Taking into account the temporal complexity of $\tilde{\Delta}^t$, and the spatial complexity of the vorticity field, we conclude that the vorticity field has transitioned into a turbulent state.

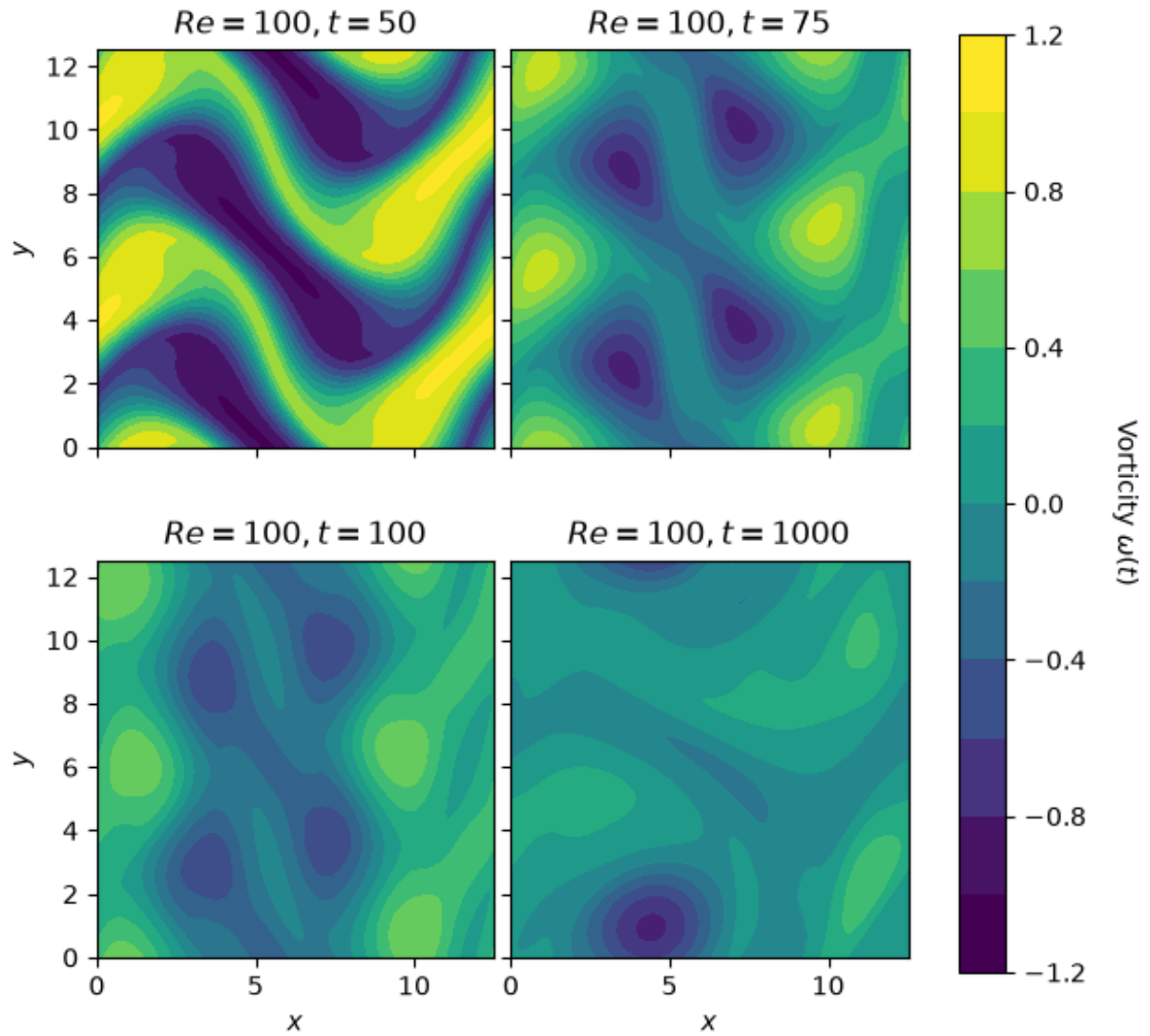


Figure 6.9: Contour plots of the vorticity field of the Kolmogorov system at $t = 50, 75, 100, 1000$ for $Re = 100$ on $(2, 2)$, plotted using Python (Appendix A.5.6)

6.5 Simulation on (4, 4)

Having discussed the instability for the Kolmogorov system on the simplest domains, we must now explore a larger domain. Here the domain size (4, 4) was chosen for its scale, and relative ease of computation. The resolution of the grid (4, 4) when $N = 128$ becomes 512×512 , with a domain size $[0, 8\pi]^2$. We can again determine the unstable modes selected by this discretisation, where the physical wave number $(k_x)_m = 2\pi m/\Gamma_x$, $m = 0, \dots, N_x/2$, with $\Gamma_x = 8\pi$, $N_x = 512$. Here the only physical wave numbers represented in discrete spectral space, within the region of instability on $k_x \in (0, 1)$, are $k_x = 1/4, 1/2, 3/4$.

In this section we will study the cases where $Re = 2.5, 5, 25$ in order to grasp some of the changing dynamics of the system as the Reynolds number increases. The time step is again $\Delta t = 0.5/Re$, and the problems here were computed up to $t = 2000$.

6.5.1 Instability on (4, 4) with $Re = 2.5$

We now discuss the results of solving the Kolmogorov system on (4, 4) with $Re = 2.5$. Table 6.2 shows the largest eigenvalue for the modes with $k_x = 1/4, 1/2, 3/4$ that are represented by (4, 4). From table 6.2 we see that the growth rates for $k_x = 1/4$ and $k_x = 1/2$ are both

k_x	1/4	1/2	3/4
λ	0.035...	0.038...	-0.084...

Table 6.2: Largest eigenvalue λ for a given k_x , when $Re = 2.5$, results produced by “point_growth()” (Appendix A.1.5)

close values, thus we might expect the modes with either k_x to be selected by unstable growth. Additionally, the growth rate for $k_x = 3/4$ is negative, thus we expect the mode with $k_x = 3/4$ to be stable.

Figure 6.10 shows that $\tilde{\zeta}^t$ has a gradient closer to the dashed tangent line representing the growth rate $\lambda = 0.035\dots$, than the dashed tangent line representing $\lambda = 0.038\dots$. This suggests that the enstrophy of the vorticity perturbation selects the mode with $k_x = 1/4$ when it undergoes unstable growth.

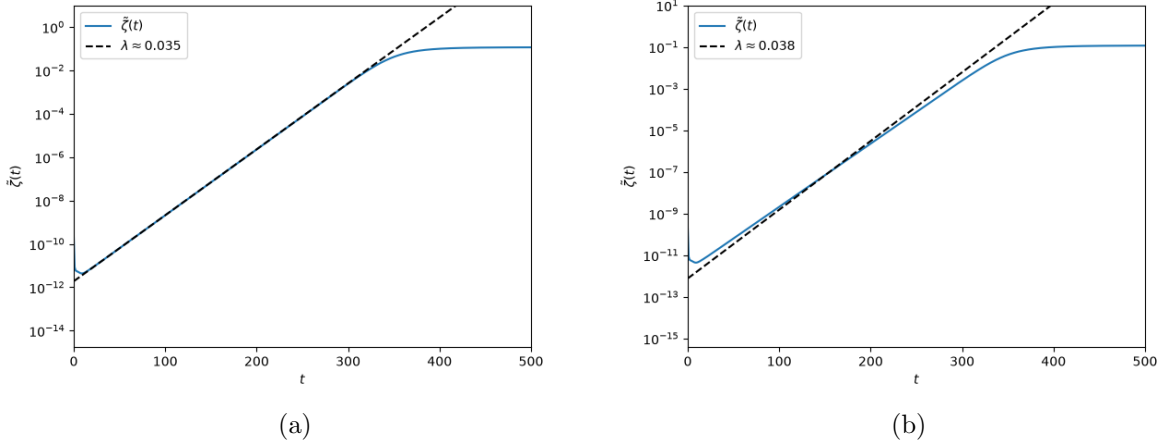


Figure 6.10: Plot showing $\tilde{\zeta}(t)$ on (4, 4) with $N = 128$ and $Re = 2.5$ on a logged vertical axis. The black dashed line is a tangent with gradient 2λ , representing the expected growth rate (a) $\lambda = 0.035\dots$ indicative of the $k_x = 0.25$ mode, and (b) $\lambda = 0.038\dots$ indicative of the $k_x = 0.5$ mode. Calculated in Python using `perturbation_energy.py`, and plotted by code in Appendix A.5.7.

Figure 6.11 shows that $\tilde{\Delta}^t$ initially increases as the system is perturbed at $t = 0$. At approximately $t = 500$ we observe that $\tilde{\Delta}^t$ then begins decreasing linearly on the logged vertical axis, this is indicative of exponential decay, thus the vorticity field reaches a steady state for $Re = 2.5$ on (4, 4).

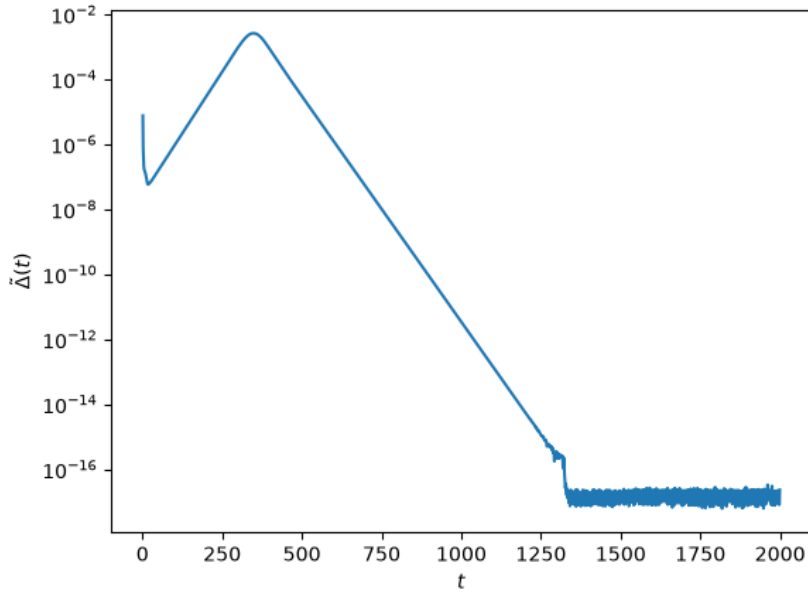


Figure 6.11: Mean absolute time derivative $\tilde{\Delta}(t)$ for $Re = 2.5$ on (4, 4) with $N = 128$, evolved with time. Calculated in Python using `perturbation_energy.py`, and plotted by code in Appendix A.5.7.

The growth of $\tilde{\zeta}^t$ more closely resembles the growth expected from the unstable mode with $k_x = 1/4$ than the mode with $k_x = 1/2$, which has the largest growth rate. It is therefore worthwhile understanding systems steady state, in order to gain further understanding. Figure 6.12 depicts the vorticity field of the Kolmogorov system on $(4, 4)$ for a $Re = 2.5$ at $t = 2000$, as the system nears its steady state. The vorticity field appears to be separated into 4 distinct bands of positive vorticity, which span the x axis. As we move along the x axis each band has a positive vortex, around which the band “kinks” upwards, and a second positive vortex around which the band kinks back downwards. This pattern is 8π periodic across the x axis, which is indicative of a steady state resulting from growth of the mode with $k_x = 1/4$, which corroborates expectations drawn from figure 6.10.

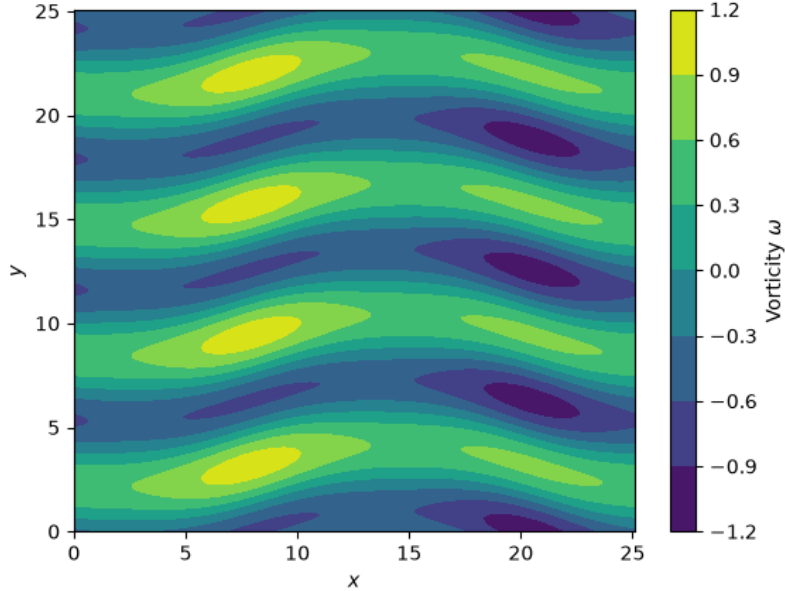


Figure 6.12: Contourf plot of the vorticity field on $(4, 4)$ with $Re = 2.5$ at $t = 2000$. Plotted in Python (Appendix A.5.8).

6.5.2 Instability on $(4, 4)$ with $Re = 5$

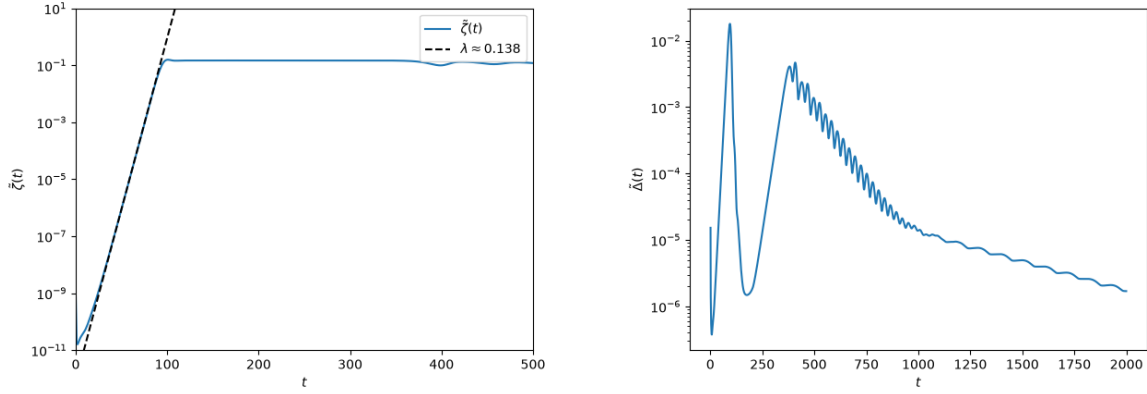
Similarly to the previous case we can assess the growth rate of unstable perturbations for the physical wave numbers $k_x = 1/4, 1/2, 3/4$ that are represented on $(4, 4)$. Table 6.3 shows the largest growth rate λ for a given k_x . We expect the mode with $k_x = 1/2$ to likely dominate the system, since the growth rates of the other modes are much smaller. Sub-figure 6.13a shows $\tilde{\zeta}^t$ for the grid $(4, 4)$ with $Re = 5$. The system initially becomes unstable at $t = 0$, where the slope of $\tilde{\zeta}^t$ visually matches the tangent representing the expected growth, with $\lambda = 0.138\dots$. The

k_x	1/4	1/2	3/4
λ	0.081...	0.138...	0.073...

Table 6.3: Largest eigenvalue λ for a given k_x , when $Re = 2.5$, results produced by “point_growth()” (Appendix A.1.5)

plot of $\tilde{\zeta}^t$ levels out after $t = 100$, where it appears to stay smooth until it begins to exhibit oscillatory behaviour at $t = 400$.

If we turn our attention to sub-figure 6.13b we see that $\tilde{\Delta}^t$ initially peaks, before declining again after $t = 100$, where $\tilde{\Delta}^t$ behaves as if the system is approaching a steady state. At approximately $t = 200$ we find $\tilde{\Delta}^t$ turns sharply, and begins increasing again. After $\tilde{\Delta}^t$ peaks again at approximately $t = 500$, we start to observe oscillations forming.



(a) Plot showing $\tilde{\zeta}(t)$ on (4, 4) with $N = 128$ and $Re = 5$ on a logged vertical axis. The black dashed line is a tangent with gradient 2λ , representing the expected growth rate $\lambda = 0.138\dots$ indicative of the $k_x = 1/2$ mode. Calculated in Python using `perturbation_energy.py`, and plotted by code in Appendix A.5.7.

(b) plot showing $\tilde{\Delta}(t)$ for the Kolmogorov system simulated on (4, 4), with $N = 128$ and $Re = 5$, evolved with time. Results plotted on a logged vertical axis in Python.

Figure 6.13: Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on (4, 4), with $N = 128$ and $Re = 5$. Calculated in Python using `perturbation_energy.py`.

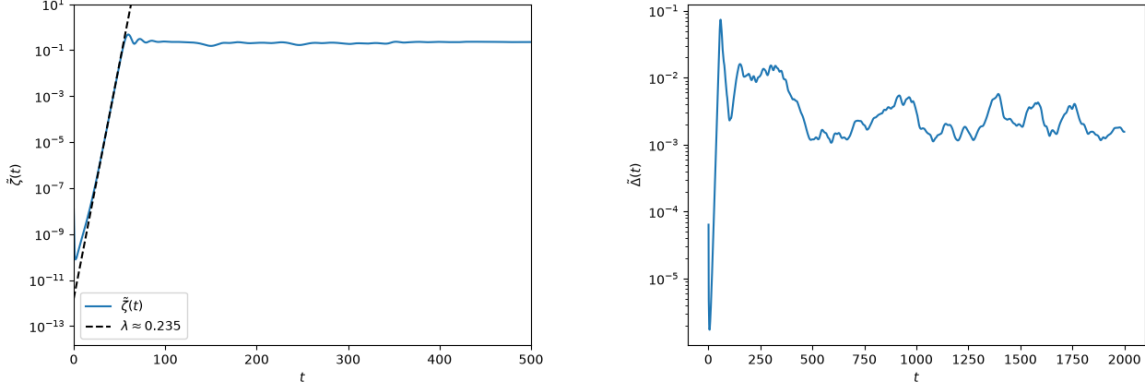
6.5.3 Instability and Turbulence on (4, 4) with $Re = 25$

We now consider the case $Re = 25$, first we assess the growth rate of unstable perturbations for the physical wave numbers $k_x = 1/4, 1/2, 3/4$ that are represented on (4, 4). Table 6.4 shows the largest growth rate λ for a given k_x . We expect the mode with $k_x = 1/2$ to likely dominate the system, since the growth rates of the other modes are much smaller. Examining sub-figure 6.14a we see that the growth of $\tilde{\zeta}^t$ matches the expected rate $\lambda = 0.235\dots$, as predicted by stability analysis. By $t = 50$ the growth of instability has peaked. Now taking a look at sub-figure 6.14b

k_x	1/4	1/2	3/4
λ	0.143...	0.235...	0.206...

Table 6.4: Largest eigenvalue λ for a given k_x , when $Re = 2.5$, results produced by “point_growth()” (Appendix A.1.5)

we observe an interesting phenomenon, the plot showing $\tilde{\Delta}^t$ has become disordered and chaotic, transitioning the system into a state where the time derivative has become unpredictable.



(a) Plot showing $\tilde{\zeta}(t)$ on (4, 4) with $N = 128$ and $Re = 25$ on a logged vertical axis. The black dashed line is a tangent with gradient 2λ , representing the expected growth rate $\lambda = 0.235\dots$ indicative of the $k_x = 1/2$ mode. Calculated in Python using `perturbation_energy.py`, and plotted by code in Appendix A.5.7.

(b) Plot showing $\tilde{\Delta}(t)$ for the Kolmogorov system simulated on (4, 4), with $N = 128$ and $Re = 25$, evolved with time. Results plotted on a logged vertical axis in Python.

Figure 6.14: Plots showing $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$ for the Kolmogorov system on (4, 4), with $N = 128$ and $Re = 25$. Calculated in Python using `perturbation_energy.py`.

Figure 6.15 shows the resulting vorticity field at $t = 50, 60, 75, 200$. If we first examine the field at $t = 50$, we see that it has separated into positive and negative bands of vorticity, which span across the x axis. This banded pattern appears to repeat twice over the domain on the x axis, which is indicative of the unstable perturbed mode with $k_x = 1/2$. As the system evolves to $t = 60$ the banded pattern begins to kink up, forming independent positive and negative vortices. Evolving the system to $t = 75$, we see that the field again forms a banded pattern, this time spanning the y axis, and with a much lower magnitude of vorticity. By the time the system evolves to $t = 200$ the banded pattern has fully broken down, and the system has become spatially complex. Between the unpredictable motions of $\tilde{\Delta}^t$, and spatial complexity of the flow, we conclude that the system has become turbulent.

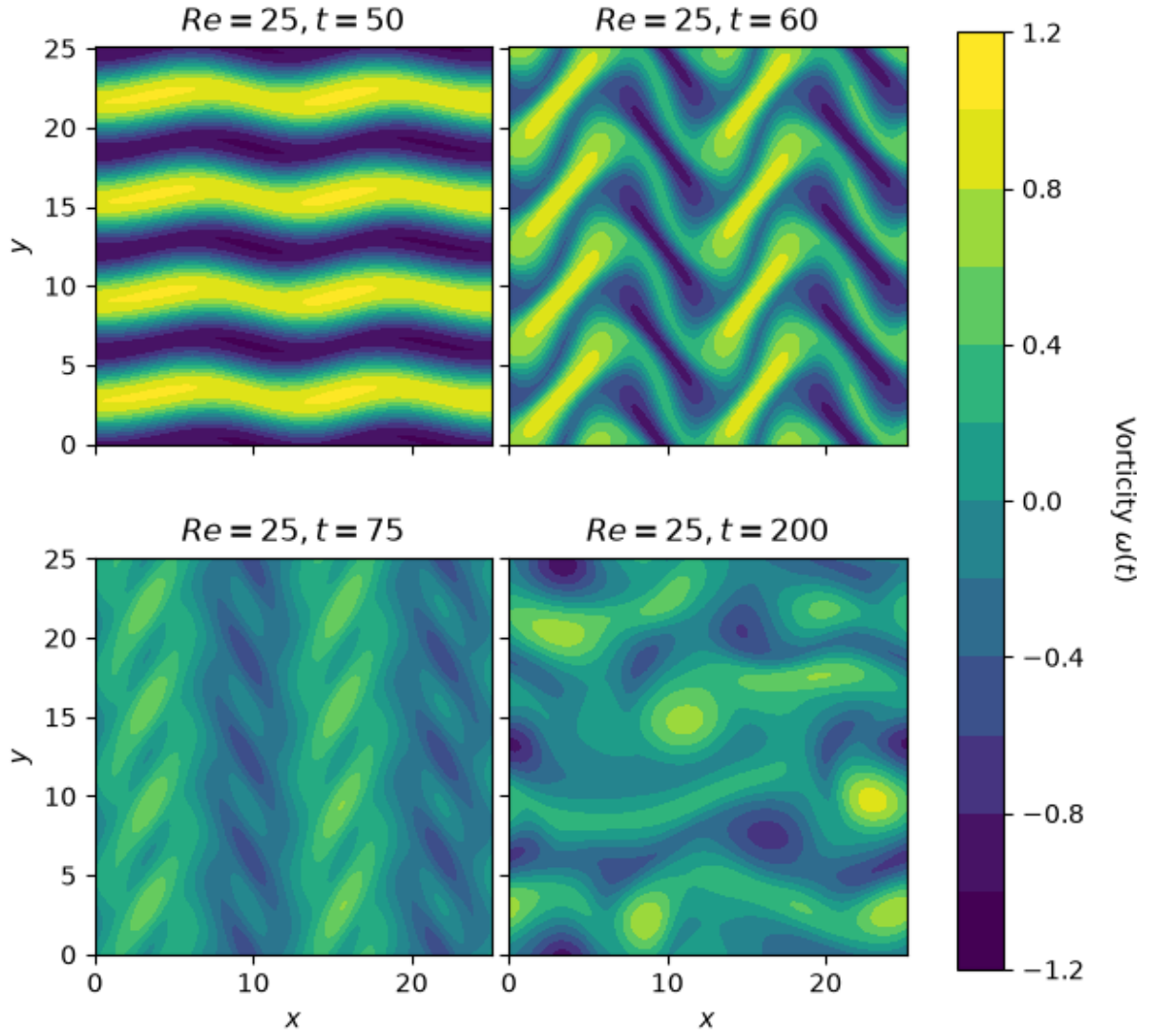


Figure 6.15: Contour plots of the vorticity field of the Kolmogorov system at $t = 50, 60, 75, 200$ for $Re = 25$ on $(4, 4)$, plotted using Python (Appendix A.5.8).

Comparing the turbulence here with that of the $(2, 2)$ case in figure 6.9, we find that this system appears to exhibit far more turbulence, even for a significantly lower Reynolds number. We now recall the results for $(2, 1), (2, 2)$ and $(4, 4)$, it is apparent that the additional domain space elicits more interesting behaviour from the Kolmogorov system. We could go on to explore $(4, 4)$ for larger Reynolds numbers, however we instead choose to investigate an even larger domain.

6.6 Simulation on (16, 16)

The final set of results for this paper will take place on (16, 16). The stability analysis for a domain of this size is too complex for the scope of this paper, thus we will simply appreciate the turbulent systems which unfold.

6.6.1 Instability on (16, 16) with $Re = 2.5$

For simulating on a grid that is significantly larger than (4, 4) it seems appropriate to begin with the Reynolds number $Re = 2.5$. Figure 6.16 shows $\tilde{\Delta}^t$ for the (16, 16) grid, with $Re = 2.5$. When the system is perturbed at $t = 0$ we see that $\tilde{\Delta}^t$ begins to increase linearly on the logged vertical axis. At approximately $t = 250$ we see $\tilde{\Delta}^t$ begin to decrease linearly on the logged vertical axis. This looks like the typical case of $\tilde{\Delta}^t$ we have seen for $Re = 2.5$, until $\tilde{\Delta}^t$ begins to increase again. After $t = 750$ we observe sporadic increases in $\tilde{\Delta}^t$, which appears complex over time. In order to determine if the system has become turbulent we must examine the vorticity field.

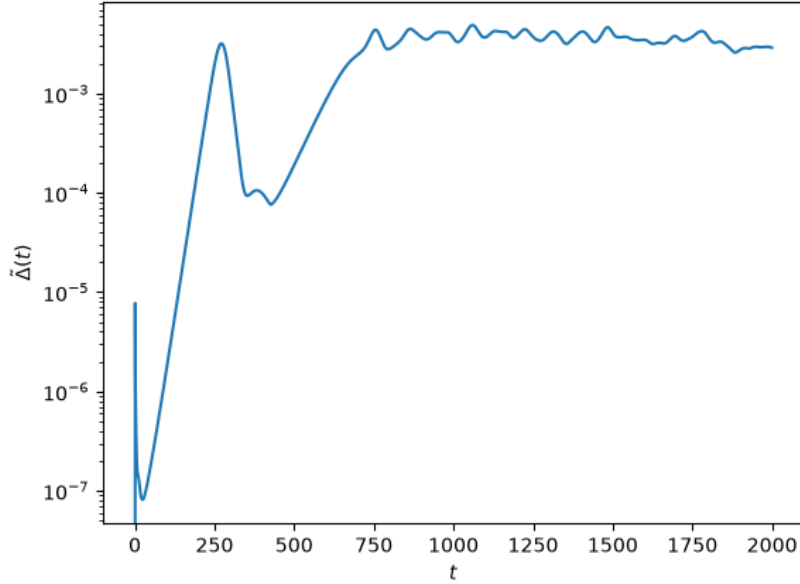


Figure 6.16: Plot showing $\tilde{\Delta}(t)$ for the Kolmogorov system simulated on (16, 16), with $N = 128$ and $Re = 2.5$, evolved with time. Results plotted on a logged vertical axis in Python (Appendix A.5.9).

Figure 6.17 shows the vorticity field of the system on (16, 16), with $Re = 2.5$, and at $t = 300, 1000, 1250, 1500$. At $t = 300$ the vorticity field is arranged into the typical bands of positive

and negative vorticity. As the system evolves to $t = 1000$ some of the bands of vorticity appear to have shifted. As the system evolves to $t = 1250$ and then $t = 1500$ the vorticity does not appear to change much again, apart from some small disturbances. From this we conclude that whilst the system does not become stable, and $\tilde{\Delta}^t$ appears temporally complex, the vorticity is at no point spatially complex. Thus, the Kolmogorov system does not exhibit turbulence for $Re = 2.5$ on $(16, 16)$.

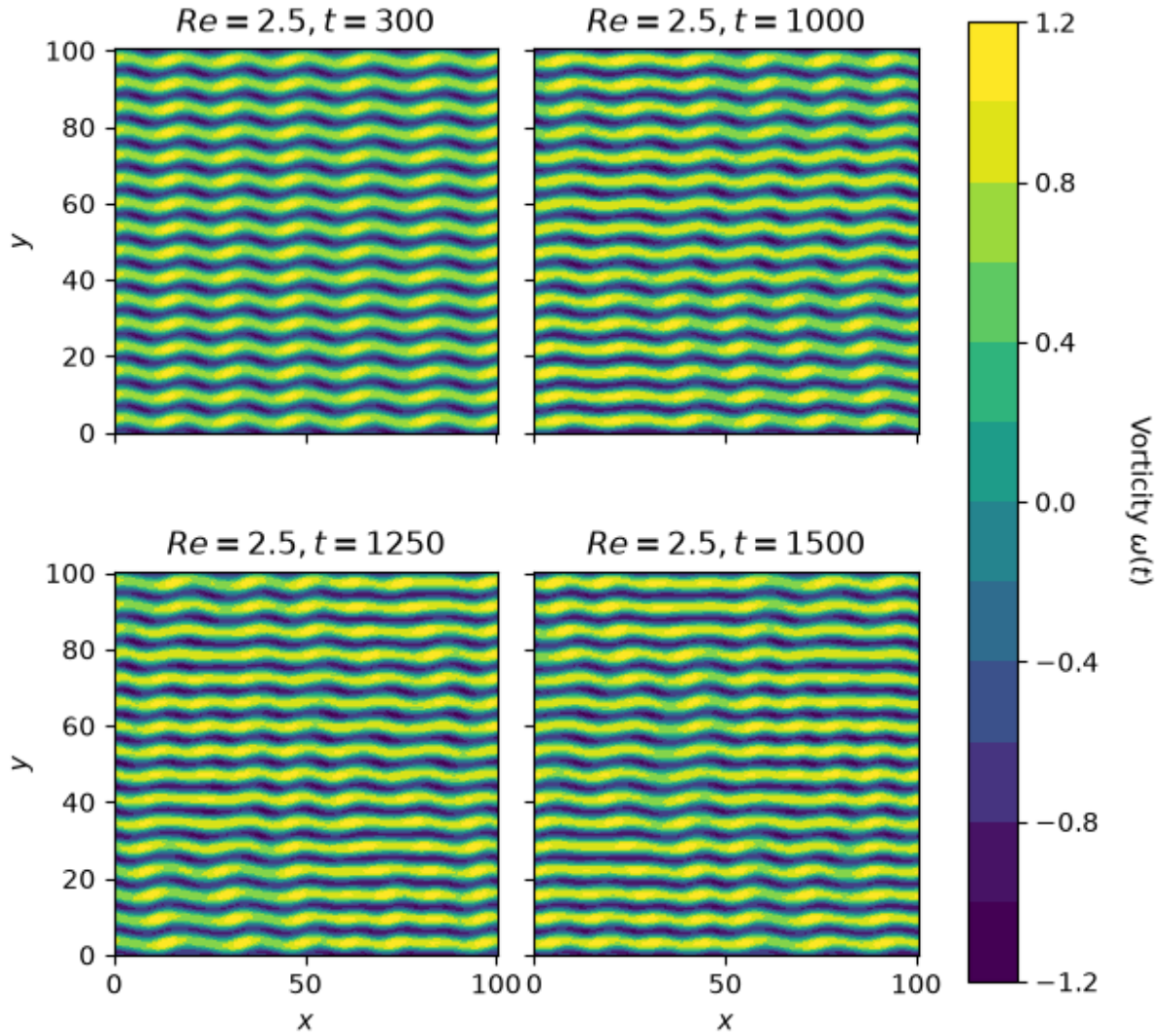


Figure 6.17: Contour plots of the vorticity field of the Kolmogorov system at $t = 300, 1000, 1250, 1500$ for $Re = 2.5$ on $(16, 16)$, plotted using Python (Appendix A.5.9)

6.6.2 Instability and Turbulence on (16, 16) with $Re = 100$

In the interest of observing larger scale turbulent behaviour the Kolmogorov system was simulated on (16, 16) with $Re = 100$. Figure 6.18 shows $\tilde{\Delta}^t$ for the (16, 16) grid, with $Re = 100$. When the system is perturbed at $t = 0$ we see that $\tilde{\Delta}^t$ begins to increase linearly on the logged vertical axis. At approximately $t = 100$ we see $\tilde{\Delta}^t$ changes direction, and begins decreasing on the logged vertical axis, however it does not do so linearly. After this point $\tilde{\Delta}^t$ exhibits temporal complexity. In order to determine if the system has become turbulent we must again examine the vorticity field.

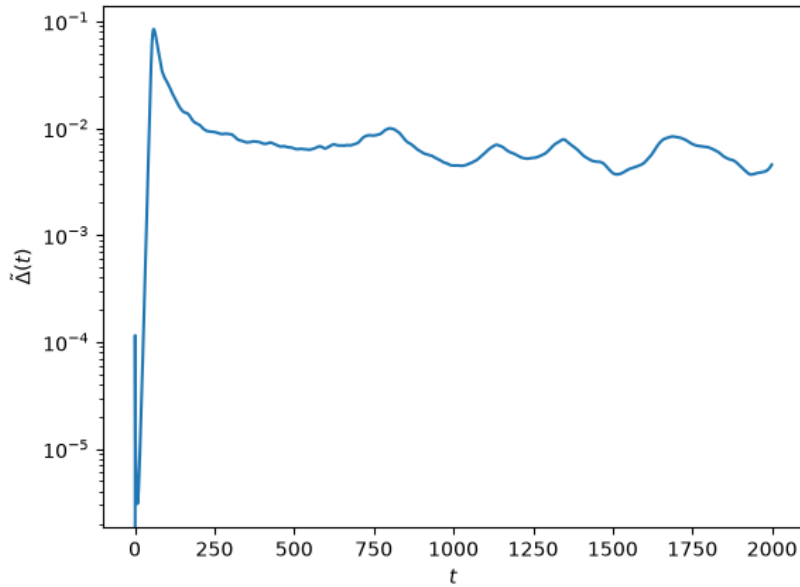


Figure 6.18: Plot showing $\tilde{\Delta}(t)$ for the Kolmogorov system simulated on (16, 16), with $N = 128$ and $Re = 100$, evolved with time. Results plotted on a logged vertical axis in Python (Appendix A.5.9).

Figure 6.19 shows the vorticity field for $t = 40, 55, 75, 200$. At $t = 40$ the vorticity field still looks like Kolmogorov flow, with long strips of positive then negative vorticity, that span across the x axis. As the system evolves to $t = 55$ the vorticity field becomes unstable, however due to the scale of the domain this instability occurs locally and then propagates outwards. The strips of vorticity have begun twisting and folding over each other. By the time the system has evolved to $t = 75$ the vorticity field is already starting to appear spatially complex, the positive vorticity is arranged in long ribbon like structures, which have begun twisting and folding into each other. Finally by the time the system has evolved to $t = 200$ there is no sense

of structure, the vorticity field has become spatially complex. The system is both spatially, and temporally complicated, thus it has transitioned into a state of turbulence. From this process we see how the Kolmogorov steady state transitions into an unstable flow, which then transitions into turbulence.

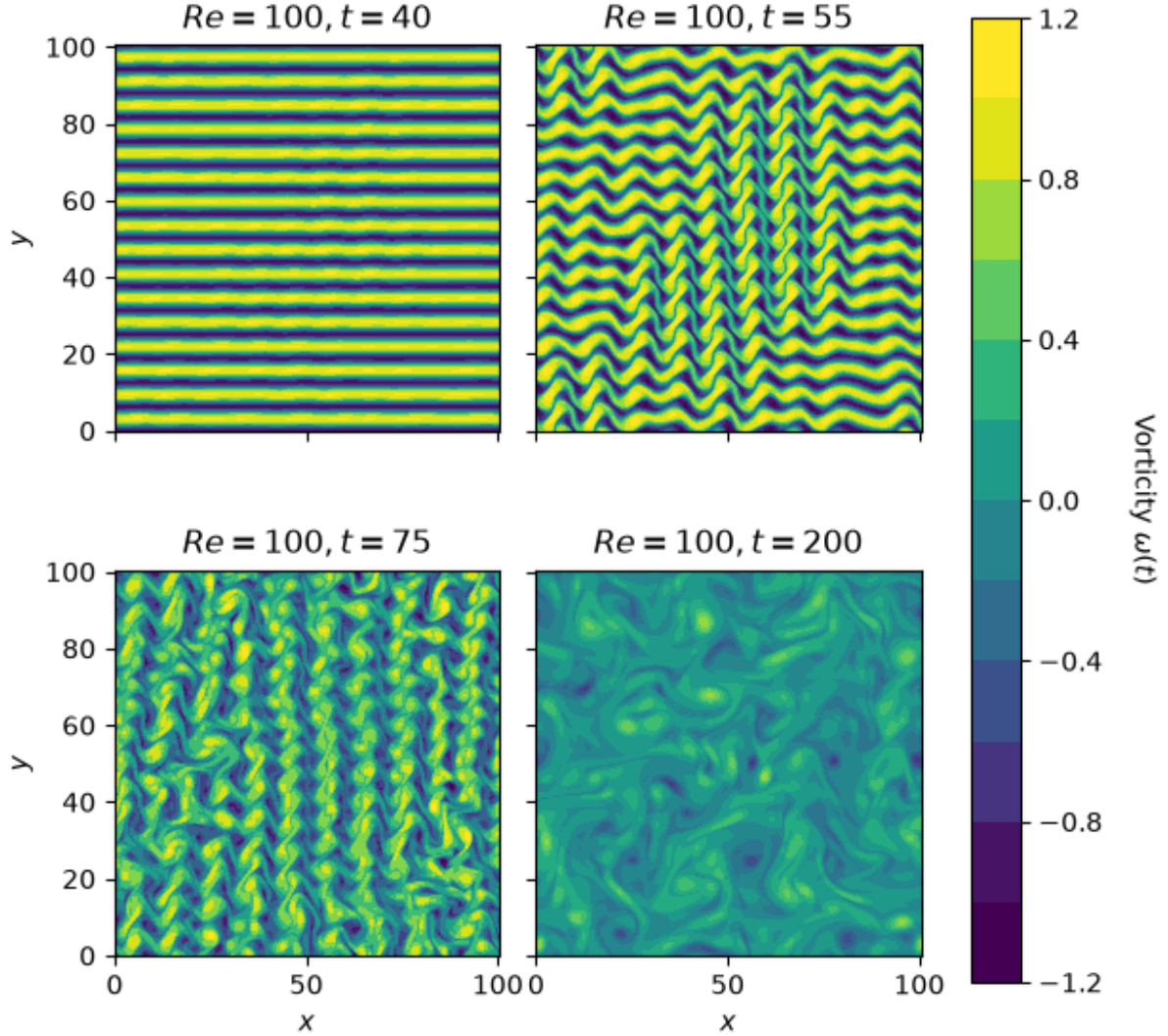


Figure 6.19: Contour plots of the vorticity field of the Kolmogorov system at $t = 40, 55, 75, 200$ for $Re = 100$ on $(16, 16)$, plotted using Python (Appendix A.5.9)

6.7 Conclusions on Transition to Turbulence

Whilst the stability of the Kolmogorov system for any domain is relatively straightforward to assess, the likelihood that a system becomes turbulent is much harder to predict. The findings in this chapter suggest that larger domains have a much easier time transitioning into turbulence,

with turbulence on $(4, 4)$ exhibiting far more complex spatial structure than that of $(2, 2)$ for example. The transition to turbulence was characterised by an initial growth of an unstable perturbation, followed by the subsequent disruption of the perturbed vorticity field, as the system becomes more spatially complex.

Chapter 7

Discussion

This paper set out to investigate the instability and turbulence of the Kolmogorov system, through the use of spectral methods and time stepping techniques. Overall this paper managed to successfully simulate the Kolmogorov system using spectral methods, and in doing so explored the transition to turbulence the system undergoes at unstable Reynolds numbers. The transition to turbulence manifested as the growth of an initial unstable perturbation, followed by a subsequent rise in spatial complexity as the perturbed field became disordered. This paper also found the critical Reynolds number for instability in the Kolmogorov system to be $Re_c = \sqrt{2}$, which is consistent with the findings of [10, 8].

A limitation of the present study was the choice of mesh resolution used to simulate the Kolmogorov system. The results were found to vary in time for different spatial and temporal resolutions, as discussed. Whilst results only varied in time, and remained qualitatively consistent, it would have still been preferable to choose a more consistent grid. A future study should make use of a more rigorous mesh convergence study, so that the correct spatio-temporal grid can be chosen. Another limitation was the rigid time stepping approach, as the numerical scheme could become unstable if the time step was not sufficiently small. A future study could make use of the CFL condition to implement adaptive time stepping, thus improving scheme stability and efficiency.

Overall the paper achieved its aims, successfully simulating the Kolmogorov system, and its transition into instability.

References

- [1] Bailly, C. and Comte-Bellot, G. *Turbulence*. Apr. 2015. DOI: 10.1007/978-3-319-16160-0.
- [2] Batchaev, A. and Ponomarev, V. M. “Experimental and theoretical investigation of Kolmogorov flow on a cylindrical surface”. In: *Fluid Dynamics* 24 (1989), pp. 675–680. URL: <https://api.semanticscholar.org/CorpusID:123237266>.
- [3] Beaume, C. *Spectral element solver for the sub-surface fluid dynamics of dry salt lakes*. 2024. URL: https://cbeaume.com/download/DryLa_manual.pdf.
- [4] Cooley, J. W. and Tukey, J. W. “An Algorithm for the Machine Calculation of Complex Fourier Series”. In: *Mathematics of computation* 19.90 (1965), pp. 297–301.
- [5] Döös, K., Lundberg, P., and Aldama-Campino, A. *Basic Numerical Methods in Meteorology and Oceanography*. Stockholm University Press, 2022.
- [6] Fylladitakis, E. D. “Kolmogorov flow: seven decades of history”. In: *Journal of Applied Mathematics and Physics* 6.11 (2018), pp. 2227–2263.
- [7] James Stewart Daniel K. Clegg, S. W. *Calculus, Metric Edition*. eng. Cengage Learning EMEA, 2020. URL: <https://www.perlego.com/book/4208725/calculus-metric-edition-pdf>.
- [8] L.D., M. and Ia.G., S. “Investigation of the stability of a stationary solution of a system of equations for the plane movement of an incompressible viscous liquid”. In: *Journal of Applied Mathematics and Mechanics* 25.6 (1961), pp. 1700–1705. DOI: [https://doi.org/10.1016/0021-8928\(62\)90149-1](https://doi.org/10.1016/0021-8928(62)90149-1).

-
- [9] Lloyd N., T. *Spectral methods in MATLAB*. eng. Software, environments, tools. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2000.
- [10] Love, P. “Bifurcations in Kolmogorov and Taylor-Vortex Flows”. PhD thesis. California Institute of Technology, 1999.
- [11] Lucas, D. and Kerswell, R. “Spatiotemporal dynamics in two-dimensional Kolmogorov flow over large domains”. In: *Journal of Fluid Mechanics* 750 (2014), pp. 518–554. DOI: 10.1017/jfm.2014.270.
- [12] Paterson, A. R. “Vorticity”. In: *A First Course in Fluid Dynamics*. Cambridge University Press, 1983, pp. 71–94.
- [13] Scharinger, J. “Fast encryption of image data using chaotic Kolmogorov flows”. In: *Journal of Electronic imaging* 7.2 (1998), pp. 318–325.
- [14] Sreenivasan, K. R. “Fluid turbulence”. In: *Rev. Mod. Phys.* 71 (2 Mar. 1999), S383–S395. DOI: 10.1103/RevModPhys.71.S383.
- [15] Stade, E. “Fourier Coefficients and Fourier Series”. In: *Fourier Analysis*. John Wiley & Sons, Ltd, 2005. Chap. 1, pp. 1–77. DOI: <https://doi.org/10.1002/9781118165508.ch1>.
- [16] Stade, E. “Fourier Transforms and Fourier Integrals”. In: *Fourier Analysis*. John Wiley & Sons, Ltd, 2005. Chap. 6, pp. 297–352. DOI: <https://doi.org/10.1002/9781118165508.ch6>.
- [17] Wyngaard, J. C. “Atmospheric turbulence”. In: *Annual Review of Fluid Mechanics* 24.1 (1992), pp. 205–234.

Appendix A

Appendix

A.1 Stability Analysis

A.1.1 Function: growth()

```
from numpy import arange, eye, linalg, max, reshape, linspace, meshgrid
    , append, array
import matplotlib.pyplot as plt

'''
MATRIX ENTRY FUNCTIONS
'''
def alpha(kx,ky):
    # calculate alpha for kx,ky
    try:
        alpha = kx*(ky/(kx**2 + ky**2) - 1/2)
    except:
        alpha = 0
    return alpha

def beta(kx,ky,Re):
    # calculate beta for kx,ky
    try:
        beta = -(kx**2 + ky**2)/Re
    except:
        beta = 0
    return beta

def gamma(kx,ky):
    # calculate gamma for kx,ky
    try:
        gamma = kx*(ky/(kx**2 + ky**2) + 1/2)
    except:
        gamma = 0
    return gamma

'''
```

```

KY CALCULATOR
'''
def KY(m): # ky calculation function
    ky = arange(-m,m+1).reshape(-1,1) # calculate ky for a given matrix
    number
    return ky

'''
MAXIMUM EIGENVALUE
'''
def growth(ky,kx,Re):
    dim = len(ky) # dimension of matrix
    # build tri diagonal matrix from constituents
    M = eye(dim,k=-1)*alpha(kx,ky) + eye(dim)*beta(kx,ky,Re) + eye(dim,
    k=1)*gamma(kx,ky)
    return max(linalg.eigvals(M)).real # return its largest eigenvalue

```

A.1.2 Function: growth_mesh()

```

'''
INSTABILITY CALCULATOR
'''
def growth_mesh(Dkx,DRe,h,m):
    Nkx = round((Dkx[1]-Dkx[0])/h) # number of mesh points in kx
    NRe = round((DRe[1]-DRe[0])/h) # number of mesh points in Re
    kx = linspace(Dkx[0],Dkx[1],Nkx) # kx linspace
    Re = linspace(DRe[0],DRe[1],NRe) # Re linspace
    (KX,RE) = meshgrid(kx,Re) # meshgrid
    ky = KY(m) # ky for given matrix

    growth_mesh = [] # store largest eigenvalue across the grid
    for r in range(NRe): # loop over rows
        growth_row = []
        for c in range(Nkx): # loop over column
            growth_row.append(growth(ky,KX[r][c],RE[r][c])) # growth
            rate determined from matrix determinant
        growth_mesh.append(growth_row) # append growth rate to the mesh

    return (KX,RE,array(growth_mesh)) # return mesh of growth rates

```

A.1.3 Re_c Convergence as $n \rightarrow \infty$

```

from functions_package.instability import growth_mesh
import matplotlib.pyplot as plt
from numpy import arange, append, min, sqrt

Re_c_diff = [] # initialise storage for critical reynolds number data
nmax = 40 # number of grid matrix Mn used
n = arange(1,nmax+1) # grid in n, for the resolutions of the grid in kx
,Re
grid_size = 10 # number of grid points

h = 1e-2 # grid spacing

```

```

eRe = sqrt(2) # test reynolds number
DRe = [eRe - grid_size*h/2, eRe + grid_size*h/2] # set mesh around Re =
      sqrt(2)
Dkx = [h, h + grid_size*h] # set mesh to be h away from kx = 0

(KX,RE,growth) = growth_mesh(Dkx,DRe,h,40) # get the growth for the
      mesh near kx = h, for n = 40
cont = plt.contour(KX,RE,growth,levels = [0]).allsegs # save segments
Re_c = min(cont[0][0][:,1]) # save minimum

for i in range(len(n)): # loop over n to show convergence for larger
      matrix
    (KX,RE,growth) = growth_mesh(Dkx,DRe,h,n[i]) # get the growth for
      the mesh near kx = h
    cont = plt.contour(KX,RE,growth,levels = [0]).allsegs # save
      segments
    Re_c_diff.append(abs(min(cont[0][0][:,1]) - Re_c)) # append
      absolute difference between Re_c at current n and n=40

plt.close()
plt.semilogy(n,Re_c_diff) # plot on logged axis
plt.xlabel(r'$n$')
plt.ylabel(r'$C^n$')
plt.show() # show plot

```

A.1.4 Re_c Convergence as $\Delta S \rightarrow 0$

```

from functions_package.instability import growth_mesh
import matplotlib.pyplot as plt
from numpy import arange, append, min, sqrt

Re_c_diff = [] # initialise storage for critical reynolds number data
n = 20 # number of grid matrix Mn used
grid_size = 10 # number of grid points

a = arange(1,40) # number of grid spacings
h = 1e-2/a # grid spacing
hmin = 1e-2/40
eRe = sqrt(2) # test reynolds number
DRe = [eRe - grid_size*hmin/2, eRe + grid_size*hmin/2] # set mesh
      around Re = sqrt(2)
Dkx = [hmin, hmin + grid_size*hmin] # set mesh to be h away from kx = 0

(KX,RE,growth) = growth_mesh(Dkx,DRe,hmin,20) # get the growth for the
      mesh for h/40
cont = plt.contour(KX,RE,growth,levels = [0]).allsegs # save segments
Re_c = min(cont[0][0][:,1]) # save minimum
print(Re_c)

for i in range(len(a)): # loop over n to show convergence for larger
      matrix
    DRe = [eRe - grid_size*h[i]/2, eRe + grid_size*h[i]/2] # set mesh
      around Re = sqrt(2)

```

```

Dkx = [h[i], h[i] + grid_size*h[i]] # set mesh to be h away from kx
= 0

(KX,RE,growth) = growth_mesh(Dkx,DRe,h[i],n) # get the growth for
the mesh near kx = h
cont = plt.contour(KX,RE,growth,levels = [0]).allsegs # save
segments
Re_c_diff.append(abs(min(cont[0][0][:,1]) - Re_c)) # append
absolute difference between Re_c at current n and n=40

plt.close()
plt.plot(h,Re_c_diff) # plot on logged axis
plt.xlabel(r'$\Delta S$')
plt.ylabel(r'',)
plt.show() # show plot

```

A.1.5 Function: point_growth()

```

def point_growth(kx,Re):
n = 1 # matrix number n
growth0 = (growth(KY(1),kx,Re)) # growth for n = 1

run = True # while loop
while run == True:
    n += 1 # increment n by 1
    growth1 = growth0 # update old variable
    growth0 = growth(KY(n),kx,Re) # calculate growth for new n
    diff = abs(growth1-growth0) # take the absolute difference
    if diff < 1e-14: # if result remains unchanged between n end loop
        run = False
    return growth0 # retrain the most recent growth rate

```

A.2 Spectral Methods

A.2.1 Spectral Accuracy in 1D

```

from functions_package.error import rmse # RMSE function
from numpy import pi,cos,sin,real,arange,zeros,exp,log,append,linspace,
    sqrt
import matplotlib.pyplot as plt
from scipy.fft import rfft, irfft, rfftfreq

'''
N GRID
'''
Gamma = 2*pi # periodicity of the system
mN = 25 # number of paired points in N
N = 2*arange(1,mN) # N grid
Dx = [0,Gamma] # range of x domain
RMSE = [] # initialise root mean squared error array

'''
MAIN LOOP
'''
for i in range(len(N)):
    '''
    MESHGRID SETUP
    '''
    Nx = N[i] # number of grid points in x set by N
    hx = Gamma/Nx # grid spacing
    kx = 2*pi*rfftfreq(Nx,d=hx) # kx,ky mesh in spectral space
    x = linspace(Dx[0],Dx[1]-hx,Nx) # mesh of points in x

    '''
    OPERATOR TESTING
    '''
    u = exp(sin(pi*x/Gamma)*cos(pi*x/Gamma)) # test function
    dudx = pi/Gamma*(cos(pi*x/Gamma)**2-sin(pi*x/Gamma)**2)*u #
    expected

    uh = rfft(u) # test function in spectral space
    dudxAh = 1j*kx*uh # spectral derivative

    dudxA = irfft(dudxAh) # derivative approximated in physical space

    RMSE.append(rmse(dudx,dudxA,Nx)) # RMSE of the approximation

'''
PLOTTING
'''
fig = plt.figure(figsize=(8,4))
ax = plt.gca()
ax.set_yscale('log')
plt.scatter(N,RMSE,color='k',marker='.') # log y axis plot of N against
    RMSE
plt.xlabel(r'$N$') # x label
plt.ylabel('RMSE') # y label
plt.savefig('spectral_accuracy') # save figure

```

A.3 Time-Stepping in Python

A.3.1 Implicit and Explicit Solutions to the Heat Equation

```

from functions_package.mesh import mesh # creates a periodic mesh (
    custom)
from numpy import linspace, pi, cos, zeros, shape, meshgrid, exp, max
import matplotlib.pyplot as plt
from scipy.fft import rfftfreq, rfft, irfft

'''
GRID SETUP
'''
[x0,xmax,Nx] = [0,2*pi,30] # interval and number of mesh points
dx = (xmax-x0)/Nx # spacing in x
x = linspace(x0,xmax-dx,Nx) # x mesh
u0 = exp(-cos(pi*x/xmax)**2) # initial condition
uh0 = rfft(u0) # initial condition in spectral space

[dt,tmax] = [0.01065,5] # step size and time length
Nt = round(tmax/dt) # number of time steps, round to int
t = linspace(0,tmax,Nt) # t grid

D = 1 # diffusion constant
kx = 2*pi*rfftfreq(Nx,dx) # fourier grid
kx2 = kx**2 # spectral laplacian

imp_u = zeros((Nx,Nt)) # intialise implicit values
imp_u[:,0] = u0
exp_u = zeros((Nx,Nt)) # intialise explicit values
exp_u[:,0] = u0

(T,X) = meshgrid(t,x) # meshgrid for x and t

'''
STABILITY REGION FOR EXPLICIT
'''
print('Explicit stability ends at: dt = ', 2/(max(kx2)*D))

'''
IMPLICIT AND EXPLICIT SCHEMES
'''
def implicit(uh,kx2,D,dt): # implicit Euler scheme
    new_uh = uh/(1 + dt*D*kx2) # uh at new time
    return new_uh

def explicit(uh,kx2,D,dt): # explicit Euler scheme
    new_uh = (1 - dt*D*kx2)*uh # uh at new time
    return new_uh

'''
RUN SCHEMES
'''
uhi = uh0 # initialise implicit uh
uhe = uh0 # initialise explicit uh
for i in range(Nt - 1):
    uhi = implicit(uhi,kx2,D,dt) # iterate implicitly

```

```

    imp_u[:, i+1] = irfft(uhi).real # save result to array

    uhe = explicit(uhe,kx2,D,dt) # iterate explicitly
    exp_u[:, i+1] = irfft(uhe).real # save result to array

'''
PLOT IMPLICIT RESULTS
'''
fig = plt.figure() # create a figure
ax = plt.axes(projection='3d') # 3D plot
ax.plot_surface(T,X,imp_u, cmap='cool', edgecolor='blue') # plot
    implicit results
ax.set_title('Implicit Solver') # title
ax.view_init(elev=26, azimuth=-130) # plot orientation
ax.set_xlabel(r'$t$') # time axis label
ax.set_ylabel(r'$x$') # space axis label
ax.set_zlabel(r'$u$') # u axis label

# save figure to files
plt.savefig('figures/implicit_heat_equation.png',bbox_inches='tight')

'''
PLOT EXPLICIT RESULTS
'''

fig = plt.figure() # create figure
ax = plt.axes(projection='3d') # 3D plot
ax.plot_surface(T,X,exp_u, cmap='cool', edgecolor='blue') # plot
    explicit results
ax.set_title('Explicit Solver') # title
ax.view_init(elev=26, azimuth=-130) # plot orientation
ax.set_xlabel(r'$t$') # time axis label
ax.set_ylabel(r'$x$') # space axis label
ax.set_zlabel(r'$u$') # u axis label

# save figure to files
plt.savefig('figures/explicit_heat_equation.png',bbox_inches='tight')

```

A.4 Kolmogorov Simulation

A.4.1 Function: advection()

```
def advection(kx,ky,k2_inv,omegah): # Advection operator
    # calculates J by spectral methods
    J = irfft2(1j*ky*omegah*k2_inv)*irfft2(1j*kx*omegah)-irfft2(1j*kx*
    omegah*k2_inv)*irfft2(1j*ky*omegah)
    return J
```

A.4.2 Function: aliasing_mask()

```
def aliasing_mask(kx,ky,Nx,Ny,Dx,Dy): # creates a mask that applies
    anti-aliasing
    Gammax = Dx[1] # length of x axis
    Gammay = Dy[1] # length of y axis
    mkx = ones(len(kx)) # vector of ones in x
    mky = ones(len(ky)).reshape(-1,1) # vector of ones in y
    mkx[kx > 2*pi*Nx/(3*Gammax)] = 0 # Fourier filter in x
    mky[abs(ky) > 2*pi*Ny/(3*Gammay)] = 0 # Fourier filter in y
    return mkx*mky
```

A.4.3 Validating advection()

```
from functions_package.operators_gpu import advection # advection
    operator (custom)
from functions_package.mesh_gpu import mesh, fouriermesh, laplacian_inv
    # meshes (custom)
from functions_package.error import rmse # root mean squared error (
    custom)
from cupyx.scipy.fft import rfft2, irfft2
from cupy import pi,cos,sin, zeros, arange, exp, asnumpy
import matplotlib.pyplot as plt

'''
N GRID
'''
m = 25 # number of paired points in N
N = arange(1,m)*2 # N grid
NN = len(N) # number of points in n

'''
PARAMETERS
'''
Dx = [0,2*pi] # range of x domain
Dy = [0,2*pi] # range of y domain
RMSE = zeros(NN)

'''
MAIN LOOP
'''
for i in range(NN):
    '''
```

```

MESHGRID SETUP
'''
[Nx,Ny] = [int(N[i]),int(N[i])] # number of grid points in x,y set
by N
[X,Y] = mesh(Nx,Ny,Dx,Dy) # create x,y mesh in physical space
[kx,ky] = fouriermesh(Nx,Ny,Dx,Dy) # create kx,ky mesh in fourier
space
k2_inv = laplacian_inv(kx,ky) # mesh values for the inverse
laplacian in fourier space

'''
ADVECTION OPERATOR TESTING
'''
omega = (cos(X) + cos(Y) - sin(X)**2 - sin(Y)**2)*exp(cos(X) + cos(
Y)) # test function
omegah = rfft2(omega) # test function in spectral space
JA = advection(kx,ky,k2_inv,omegah) # run advection function

psi_x = -sin(X)*exp(cos(X) + cos(Y))
psi_y = -sin(Y)*exp(cos(X) + cos(Y))
omega_x = (-sin(X) - 2*cos(X)*sin(X))*exp(cos(X) + cos(Y)) - sin(X)
*omega
omega_y = (-sin(Y) - 2*cos(Y)*sin(Y))*exp(cos(X) + cos(Y)) - sin(Y)
*omega

J = psi_y*omega_x - psi_x*omega_y # analytical solution
RMSE[i] = rmse(J,JA,Nx*Ny) # root mean squared error of the
approximation

'''
PLOT RMSE
'''
fig = plt.figure(figsize=(8,4)) # create figure
ax = plt.gca() # axis object
ax.set_yscale('log') # log the y axis scale
plt.scatter(asnumpy(N),asnumpy(RMSE),color='k',marker='.') # semilogy
of N against RMSE
plt.xlabel(r'$N$') # x label
plt.ylabel('RMSE') # y label
plt.savefig('spectral_accuracy_advection') # save figure

```

A.4.4 Function: Kolmogorov_simulation()

```

from cupyx.scipy.fft import fftfreq, rfftfreq, rfft2, irfft2
from cupy import real, pi, shape, append, sqrt, cos, sin, zeros, ones,
array, reshape, random, save, asnumpy, sum
from functions_package.mesh_gpu import mesh, fouriermesh, laplacian,
laplacian_inv
import matplotlib.pyplot as plt

'''
SPECTRAL HELMHOLTZ OPERATOR
'''
def spectral_helmholtz(k2,dt,Re): # Kolmogorov flow specific Helmholtz
operator in spectral space
    Lh = 1 + 2*k2*dt/(3*Re) # calculate helmholtz operator

```

```

    return Lh # return operator
'''
COMPUTATION OF FUTURE KOLMOGOROV STEP
'''
def Kolmogorov(Lh,dt,fh,omegah1,omegah2,Jh1,Jh2):
    Jh0 = 2*Jh1 - Jh2 # approximates advection
    Hh0 = -2*omegah1 + 1/2*omegah2 # approximates H
    omegah0 = 1/Lh * (2/3) * (dt*(fh - Jh0) - Hh0) # iterates to future
    step

    return omegah0

'''
KOLMOGOROV LOOP
'''
def Kolmogorov_simulation(N,Re,dt,tmax,m,n,state, frame,save_file):
    '''
    MESHGRID SETUP
    '''
    # number of mesh points in x and y
    Nx = N*m
    Ny = N*n
    # domain size in x and y
    Dx = [0,2*m*pi]
    Dy = [0,2*n*pi]
    # create the meshgrid
    (X,Y) = mesh(Nx,Ny,Dx,Dy)
    [kx,ky] = fouriermesh(Nx,Ny,Dx,Dy) # fourier mesh
    k2 = laplacian(kx,ky) # laplacian in spectral space
    k2_inv = laplacian_inv(kx,ky) # inverse laplacian in spectral space
    mask = aliasing_mask(kx,ky,Nx,Ny,Dx,Dy)

    '''
    MODEL STATE
    '''
    if state == True:
        init_omega = -cos(Y) # steady state start
    else:
        init_omega = zeros((Nx,Ny)) # 0 matrix state start

    init_omega = init_omega + random.rand(Ny,Nx)/1000 # appGammay
    perturbation

    '''
    MODEL SETUP
    '''
    # spectral initial conditions
    init_omegah = rfft2(init_omega) # initial omega in spectral space
    init_Jh = rfft2(advection(kx,ky,k2_inv,init_omegah)) # initial
    advection in spectral space
    fh = rfft2(-1/Re*cos(Y)) # forcing term
    Lh = spectral_helmholtz(k2,dt,Re) # helmholtz operator in spectral
    space

    '''
    RUN SIMULATION
    '''

```

```

# intitalize shifting parameters
[omegah1,omegah2] = [init_omegah,init_omegah]
[Jh1,Jh2] = [init_Jh,init_Jh]

Nt = round(tmax/dt) # number of time steps
check = round(Nt/20) # check for percentage tracking

time = []
entrophy = []
for i in range(Nt): # main loop
    omegah0 = Kolmogorov(Lh,dt,fh,omegah1,omegah2,Jh1,Jh2) # time
stepping scheme

    # update old variables
    omegah2 = omegah1
    omegah1 = omegah0
    Jh2 = Jh1
    Jh1 = rfft2(advection(kx,ky,k2_inv,mask*omegah0))

    if i%check == 0: # completion message for given percentage
        print('Percent Complete: ', int(100*i/Nt)) #
percentage complete message
        plt.pcolormesh(asnumpy(X),asnumpy(Y),asnumpy(irfft2
(omegah0)), cmap = 'berlin',vmin = -1, vmax = 1)
        plt.colorbar()
        plt.xlabel('x')
        plt.ylabel('y')
        plt.savefig(f'{save_file}/{int(100*i/Nt)}.png')
        plt.close()

    if i%frame == 0: # save every given frame
        save(f'{save_file}/{int(i/frame)}', omegah0) # saves every
10 steps as a .npy file
        time.append(i*dt)

    save(f'{save_file}/time_series', time) # save the time axis of
simulation
    print('simulation complete:')

```

A.4.5 Validating Steady State for Kolmogorov_simulation()

```

from cupy import save, load, linspace, sum, asnumpy, meshgrid, pi, cos,
array
from numpy import exp
import matplotlib.pyplot as plt
from cupyx.scipy.fft import irfft2, rfft2
from functions_package.mesh_gpu import mesh, fouriermesh, laplacian_inv
from functions_package.error import rmse

N=128 # number of points x,y in (2*pi)**2
Re=1 # Reynolds number
tmax=10 # simulation length
dt = 0.1 # time step size
ax=1 # tiles in x
ay=1 # tiles in y
frame = 1 # saves the state every n frames

```

```

state=False # steady state initial conditions
Nt = round(tmax/(dt*frame)) # number of steps in time

'''
DIRECTORY SETUP
'''
# directory for simulation data
load_file = f'simulation_logs/{N}-{Re}-{tmax}-{round(dt,3)}-{ax}-{ay}-{
    frame}-{state}'
t = asnumpy(load(f'{load_file}/time_series.npy'))

Nx = N*ax
Ny = N*ay
resolution = Nx*Ny
# domain in x and y
Dx = [0,2*ax*pi]
Dy = [0,2*ay*pi]
# create the meshgrid
(X,Y) = mesh(Nx,Ny,Dx,Dy)
[kx,ky] = fouriermesh(Nx,Ny,Dx,Dy)
k2_inv = laplacian_inv(kx,ky)

omega_h_init = rfft2(-cos(Y)) # steady state field

RMSE = [] # initialise RMSE array

for i in range(Nt):
    RMSE.append(float(rmse(load(f'{load_file}/{i}.npy'),omega_h_init,Nx
        *Ny))) # RMSE between steady state and current vorticity field

plt.semilogy(t,RMSE) # plot on log y axis
plt.xlabel(r'$t$') # label x
plt.ylabel('RMSE') # label y
plt.savefig('steady_state_approach') # save figure

```

A.5 Results

A.5.1 Mesh Convergence Study

```

from numpy import load, ones, linspace, append
import matplotlib.pyplot as plt
from functions_package.error import rmse

baseline = load('pe-128-100-200-0.005-2-1-200-True.npy') # load baseline
data
spatial = load('pe-1280-100-200-0.005-2-1-200-True.npy') # load data
for higher spatial resolution
temporal = load('pe-128-100-200-0.0005-2-1-2000-True.npy') # load data
for higher temporal resolution

t = baseline[:,0] # time axis for baseline

'''
AXIS PLOTS
'''
plt.plot(t,baseline[:,1],label=r'$dt=0.005$') # plot baseline enstrophy
plt.plot(t,temporal[:,1], '--k',label=r'$dt=0.0005$') # plot enstrophy
for a finer temporal resolution
plt.xlim([0,150])
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\zeta}(t)$')
plt.legend()
plt.savefig('enstrophy_temporal_mesh_test')
plt.close()

plt.plot(t,baseline[:,1],label=r'$N=128$') # plot baseline enstrophy
plt.plot(t,spatial[:,1], '--k',label=r'$N=1280$') # plot enstrophy for a
finer spatial mesh
plt.xlim([0,150])
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\zeta}(t)$')
plt.legend()
plt.savefig('enstrophy_spatial_mesh_test')
plt.close()

'''
REALIGNED
'''
temporal_offset = 48.48 - 45.08 # offset between enstrophy on finer
temporal mesh and baseline
plt.plot(t,baseline[:,1],label=r'$dt=0.005$') # baseline enstrophy
plt.plot(t - temporal_offset,temporal[:,1], '--k',label=r'$dt=0.0005$ (
aligned)') # plot offset corrected plots
plt.xlim([0,150])
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\zeta}(t)$')
plt.legend()
plt.savefig('enstrophy_temporal_mesh_test_aligned')
plt.close()

```

```

spatial_offset = 56.02-45.21 # offset between enstrophy on finer
    temporal mesh and baseline
plt.plot(t,baseline[:,1],label=r'$N=128$') # baseline enstrophy
plt.plot(t - spatial_offset,spatial[:,1], '--k',label=r'$N=1280$ (
    aligned)') # plot offset corrected plots
plt.xlim([0,150])
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\zeta}(t)$')
plt.legend()
plt.savefig('enstrophy_spatial_mesh_test_aligned')
plt.close()

```

A.5.2 Calculating $\tilde{\zeta}^t$ and $\tilde{\Delta}^t$

```

from cupy import save, load, linspace, sum, asnumpy, meshgrid, pi, cos,
    array, column_stack,array
from numpy import exp
import matplotlib.pyplot as plt
from cupyx.scipy.fft import irfft2, rfft2
from functions_package.mesh_gpu import mesh, fouriermesh, laplacian_inv
from re import findall

print('input file name:')
# simulation parameters from file name
file = input()
split = file.split('-')
N = int(split[0])
Re = float(split[1])
tmax = float(split[2])
dt = float(split[3])
ax = int(split[4])
ay = int(split[5])
frame = int(split[6])
state = split[7]

'''
ENERGY FUNCTIONS
'''
def enstrophy_c(omegah, resolution):
    # compute enstrophy
    return float(1/(2*resolution)*sum(irfft2(omegah)**2))

def kinetic_c(omegah, resolution, kx, ky, k2_inv):
    # compute average kinetic energy
    u = irfft2(1j*ky*k2_inv*omegah)
    v = irfft2(-1j*kx*k2_inv*omegah)
    return float(1/(2*resolution)*sum(u**2 + v**2))

def Delta(omegah0,omegah1,dt,frame,resolution):
    # compute mean average time derivative
    return float(1/(dt*frame*resolution)*sum(abs(irfft2(omegah0)-irfft2(omegah1))))

'''
DIRECTORY SETUP
'''

```

```

# directory for simulation data
load_file = f'simulation_logs/{file}'
t = load(f'{load_file}/time_series.npy')

# initialise storage arrays
kinetic = []
enstrophy = []
change = []

# number of points in x and y
Nx = N*ax
Ny = N*ay
resolution = Nx*Ny
# domain in x and y
Dx = [0,2*ax*pi]
Dy = [0,2*ay*pi]
# create the meshgrid
(X,Y) = mesh(Nx,Ny,Dx,Dy)
[kx,ky] = fouriermesh(Nx,Ny,Dx,Dy)
k2_inv = laplacian_inv(kx,ky)

omegah_init = rfft2(-cos(Y)) # initial flow in spectral space
omegah0 = perturbation_data_h = load(f'{load_file}/0.0.npy')

for i in range(len(t)):
    omegah1 = omegah0
    # load data, removing steady state to leave perturbation
    omegah0 = load(f'{load_file}/{frame*i*dt}.npy') - omegah_init

    # append to arrays
    enstrophy.append(enstrophy_c(omegah0,resolution))
    kinetic.append(kinetic_c(omegah0, resolution, kx, ky, k2_inv))
    change.append(Delta(omegah0,omegah1,dt,frame,resolution))

# save data to file
data = column_stack((t, array(enstrophy),array(kinetic),array(change)))
save(f'energies/{file}',data)

```

A.5.3 Directional Energy Calculator

```

from cupy import save, load, linspace, sum, asnumpy, meshgrid, pi, cos,
    array, column_stack,array
from numpy import exp
import matplotlib.pyplot as plt
from cupyx.scipy.fft import irfft2, rfft2
from functions_package.mesh_gpu import mesh, fouriermesh, laplacian_inv

f01 = load('128-2.5-2000-0.4-2-1-2-True/1999.2.npy') # vorticity data
    for Re 2.5 at t=1999
f10 = load('128-5-2000-0.2-2-1-5-True/1999.0.npy') # vorticity data for
    Re 5 at t=1999
f11 = load('128-25-2000-0.04-2-1-25-True/1999.0.npy') # vorticity data
    for Re 25 at t=1999
f12 = load('128-100-2000-0.01-2-1-100-True/1999.0.npy') # vorticity
    data for Re 100 at t=1999

```

```

def directional_c(omegah, kx, ky, k2_inv):
    # compute average kinetic energy
    u = irfft2(1j*ky*k2_inv*omegah)
    v = irfft2(-1j*kx*k2_inv*omegah)
    u_kinetic = float(sum(u**2))
    v_kinetic = float(sum(v**2))
    return float(v_kinetic/u_kinetic)

ax = 2 # domain shape in x
ay = 1 # domain shape in y
N = 128 # grid resolution

Nx = N*ax # number of points in x
Ny = N*ay # number of points in y

# domain in x and y
Dx = [0,2*ax*pi]
Dy = [0,2*ay*pi]
# create the meshgrid
(X,Y) = mesh(Nx,Ny,Dx,Dy)
[kx,ky] = fouriermesh(Nx,Ny,Dx,Dy)
k2_inv = laplacian_inv(kx,ky)

print('Re = 2.5: ',directional_c(f01,kx,ky,k2_inv))
print('Re = 5: ',directional_c(f10,kx,ky,k2_inv))
print('Re = 25: ',directional_c(f11,kx,ky,k2_inv))
print('Re = 100: ',directional_c(f12,kx,ky,k2_inv))

```

A.5.4 Line Plots for (2,1)

```

from numpy import load, exp, log
import matplotlib.pyplot as plt

data2_5 = load('pe-128-2.5-2000-0.4-2-1-2-True.npy') # file for Re =
2.5
data5 = load('pe-128-5-2000-0.2-2-1-5-True.npy') # file for Re = 5
data25 = load('pe-128-25-2000-0.04-2-1-25-True.npy') # file for Re = 25
data100 = load('pe-128-100-2000-0.01-2-1-100-True.npy') # file for Re =
100

[t2_5,enstrophy2_5,change2_5] = [data2_5[:,0],data2_5[:,1],data2_5
[:,3]] # data for (2,1) Re = 2.5
[t5,enstrophy5,change5] = [data5[:,0],data5[:,1],data5[:,3]] # data for
(2,1) Re = 5
[t25,enstrophy25,change25] = [data25[:,0],data25[:,1],data25[:,3]] #
data for (2,1) Re = 25
[t100,enstrophy100,change100] = [data100[:,0],data100[:,1],data100
[:,3]] # data for (2,1) Re = 100

'''
ENSTROPY PLOTTING
'''

growth2_5 = 0.03800286486808335 # growth rate for Re=2.5 on (2,1) for
mode kx = 0.5
plt.semilogy(t2_5,enstrophy2_5,label=r'$\tilde{\zeta}(t)$') # enstrophy
semilogy

```

```

plt.semilogy(t2_5,exp(2*growth2_5*(t2_5-115)+log(1.63e-08)),'--k',label
            = r'$\lambda\approx 0.038$') # expected gradient line
plt.xlabel(r'$t$') # x label
plt.ylabel(r'$\tilde{\zeta}(t)$') # y label
plt.ylim(top=10)
plt.xlim([0,500])
plt.legend(loc='upper left')
plt.savefig('figures/enstrophy-(2,1)-Re-2_5') # save figure
plt.close()

growth5 = 0.1384715222134374 # growth rate for Re=5 on (2,1) for mode
    kx = 0.5
plt.semilogy(t5,enstrophy5,label=r'$\tilde{\zeta}(t)$') # enstrophy
    semilogy
plt.semilogy(t5,exp(2*growth5*(t5-30.2)+log(2.74e-08)),'--k',label = r'
    $\lambda \approx 0.138$') # expected gradient line
plt.xlabel(r'$t$') # x label
plt.ylabel(r'$\tilde{\zeta}(t)$') # y label
plt.ylim([10e-12,10])
plt.xlim([0,200])
plt.legend()
plt.savefig('figures/enstrophy-(2,1)-Re-5') # save figure
plt.close()

growth25 = 0.23534463753988982 # growth rate for Re=25 on (2,1) for
    mode kx = 0.5
plt.semilogy(t25,enstrophy25,label=r'$\tilde{\zeta}(t)$') # enstrophy
    semilogy
plt.semilogy(t25[:100],exp(2*growth25*(t100[:100]-24)+log(2.08e-06)),'
    --k',label = r'$\lambda \approx 0.235$') # expected gradient line
plt.xlabel(r'$t$') # x label
plt.ylabel(r'$\tilde{\zeta}(t)$') # y label
plt.ylim(top=10)
plt.xlim([0,200])
plt.legend()
plt.savefig('figures/enstrophy-(2,1)-Re-25') # save figure
plt.close()

growth100 = 0.25471048613238595 # growth rate for Re=100 on (2,1) for
    mode kx = 0.5
plt.semilogy(t100,enstrophy100,label=r'$\tilde{\zeta}(t)$') # enstrophy
    semilogy
plt.semilogy(t100[:100],exp(2*growth100*(t100[:100]-27.5)+log(1.164e
    -05)),'--k',label = r'$\lambda \approx 0.254$') # expected gradient
    line
plt.xlabel(r'$t$') # x label
plt.ylabel(r'$\tilde{\zeta}(t)$') # y label
plt.ylim(top=10)
plt.xlim([0,200])
plt.legend()
plt.savefig('figures/enstrophy-(2,1)-Re-100') # save figure
plt.close()

'''
MEAN AVERAGE TIME DERIVATIVE (MATD) PLOTS
'''
# plot of MATD for Re = 2.5
plt.semilogy(t2_5[1:],change2_5[1:])

```

```

plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\Delta}(t)$')
plt.savefig('figures/MATD-(2,1)-Re-2_5')
plt.close()

# plot of MATD for Re = 5
plt.semilogy(t5[1:],change5[1:])
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\Delta}(t)$')
plt.savefig('figures/MATD-(2,1)-Re-5')
plt.close()

# plot of MATD for Re = 25
plt.semilogy(t25[1:],change25[1:])
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\Delta}(t)$')
plt.savefig('figures/MATD-(2,1)-Re-25')
plt.close()

# plot of MATD for Re = 100
plt.semilogy(t100[1:],change100[1:])
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\Delta}(t)$')
plt.savefig('figures/MATD-(2,1)-Re-100')
plt.close()

```

A.5.5 Contour Plots for (2,1)

```

from numpy import load, exp, log, pi
import matplotlib.pyplot as plt
from functions_package.mesh import mesh
from scipy.fft import irfft2

ax = 2 # domain shape in x
ay = 1 # domain shape in y
N = 128 # grid resolution

Nx = N*ax # number of points in x
Ny = N*ay # number of points in y

# domain in x and y
Dx = [0,2*ax*pi]
Dy = [0,2*ay*pi]
# create the meshgrid
(X,Y) = mesh(Nx,Ny,Dx,Dy)

f01 = irfft2(load('128-2.5-2000-0.4-2-1-2-True/1999.2.npy')) #
    vorticity data for Re 2.5 at t=1999
f10 = irfft2(load('128-5-2000-0.2-2-1-5-True/1999.0.npy')) # vorticity
    data for Re 5 at t=1999
f11 = irfft2(load('128-25-2000-0.04-2-1-25-True/1999.0.npy')) #
    vorticity data for Re 25 at t=1999
f12 = irfft2(load('128-100-2000-0.01-2-1-100-True/1999.0.npy')) #
    vorticity data for Re 100 at t=1999

# plotting sub plots

```

```

fig1, axs = plt.subplots(nrows=2,ncols=2, figsize=(8,4),sharex=True,
sharey=True)
(ax1, ax2), (ax3, ax4) = axs # set axis

# plot for Re 2.5
cs1 = ax1.contourf(X, Y, f01,vmin=-1,vmax=1, levels=10)
ax1.set_ylabel(r'$y$')
ax1.tick_params()
ax1.set_title(r'$Re=2.5,t=1999$')
ax1.set_aspect('equal', adjustable="box")

# plot for Re 5
cs2 = ax2.contourf(X, Y, f10,vmin=-1,vmax=1, levels=10)
ax2.set_title(r'$Re=5,t=1999$')
ax2.set_aspect('equal', adjustable="box")

# plot for Re 25
cs3 = ax3.contourf(X, Y, f11,vmin=-1,vmax=1, levels=10)
ax3.set_ylabel(r'$y$')
ax3.set_xlabel(r'$x$')
ax3.tick_params()
ax3.set_title(r'$Re=25,t=1999$')
ax3.set_aspect('equal', adjustable="box")

# plot for Re 100
cs4 = ax4.contourf(X, Y, f12,vmin=-1,vmax=1, levels=10)
ax4.set_xlabel(r'$x$')
ax4.tick_params()
ax4.set_title(r'$Re=100,t=1999$')
ax4.set_aspect('equal', adjustable="box")

plt.tight_layout() # layout
cbar = plt.colorbar(cs1, ax=axs) # add colour bar
cbar.ax.set_ylabel(r'$Vorticity \omega(t)$', rotation=270,labelpad=20)
plt.savefig('2-1-steadystates') # save
plt.close()

```

A.5.6 Plots for (2, 2)

```

from numpy import load, exp, log, pi
import matplotlib.pyplot as plt
from functions_package.mesh import mesh
from scipy.fft import irfft2

data = load('pe-128-100-2000-0.005-2-2-200-True.npy')
[t, enstrophy, change] = [data[:,0], data[:,1], data[:,3]]

# plot enstrophy for Re = 100
growth = 0.25471048613238595
plt.semilogy(t, enstrophy, label=r'$\tilde{\zeta}(t)$')
plt.semilogy(t[:200], exp(2*growth*(t[:200]-29.4)+log(4.17e-05)), '--k',
label=r'$\lambda \approx 0.254$')
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\zeta}(t)$')
plt.ylim(top=10)
plt.xlim([0, 200])

```

```

plt.legend()
plt.savefig('figures/enstrophy-(2,2)-Re-100')
plt.close()

# plot MATD for Re = 100
plt.semilogy(t,change)
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\Delta}(t)$')
plt.savefig('figures/MATD-(2,2)-Re-100')
plt.close()

'''
RE 100 TURBULENCE PLOT
'''

ax = 2
ay = 2
N = 128

Nx = N*ax
Ny = N*ay
# domain in x and y
Dx = [0,2*ax*pi]
Dy = [0,2*ay*pi]
# create the meshgrid
(X,Y) = mesh(Nx,Ny,Dx,Dy)

f01 = irfft2(load('128-100-2000-0.005-2-2-200-True/50.0.npy')) # file
for t=50 (2,2) Re = 100
f10 = irfft2(load('128-100-2000-0.005-2-2-200-True/75.0.npy')) # file
for t=75 (2,2) Re = 100
f11 = irfft2(load('128-100-2000-0.005-2-2-200-True/100.0.npy')) # file
for t=100 (2,2) Re = 100
f12 = irfft2(load('128-100-2000-0.005-2-2-200-True/1000.0.npy')) # file
for t=1000 (2,2) Re = 100

fig1, axs = plt.subplots(nrows=2,ncols=2, figsize=(6.2,6),sharex=True,
sharey=True)
(ax1, ax2), (ax3, ax4) = axs # set axis

# plot for t = 50
cs1 = ax1.contourf(X, Y, f01,vmin=-1,vmax=1, levels=10)
ax1.set_ylabel(r'$y$')
ax1.tick_params()
ax1.set_title(r'$Re=100,t=50$')
ax1.set_aspect('equal', adjustable="box")

# plot for t = 75
cs2 = ax2.contourf(X, Y, f10,vmin=-1,vmax=1, levels=10)
ax2.set_title(r'$Re=100,t=75$')
ax2.set_aspect('equal', adjustable="box")

# plot for t=100
cs3 = ax3.contourf(X, Y, f11,vmin=-1,vmax=1, levels=10)
ax3.set_ylabel(r'$y$')
ax3.set_xlabel(r'$x$')
ax3.tick_params()
ax3.set_title(r'$Re=100,t=100$')

```

```

ax3.set_aspect('equal', adjustable="box")

# plot for t = 1000
cs4 = ax4.contourf(X, Y, f12, vmin=-1, vmax=1, levels=10)
ax4.set_xlabel(r'$x$')
ax4.tick_params()
ax4.set_title(r'$Re=100, t=1000$')
ax4.set_aspect('equal', adjustable="box")

# plot and save results
plt.tight_layout()
cbar = plt.colorbar(cs1, ax=axis)
cbar.ax.set_ylabel(r'Vorticity $\omega(t)$', rotation=270, labelpad=20)
plt.savefig('Re100Turbulence(2,2)')
plt.close()

```

A.5.7 Line Plots for (4,4)

```

from numpy import load, exp, log
import matplotlib.pyplot as plt
from scipy.signal import find_peaks

data2_5 = load('pe-128-2.5-2000-0.2-4-4-5-True.npy') # file for (4,4)
Re = 2.5
data5 = load('pe-128-5-2000-0.1-4-4-10-True.npy') # file for (4,4) Re =
5
data25 = load('pe-128-25-2000-0.02-4-4-50-True.npy') # file for (4,4)
Re = 25
data100 = load('pe-128-100-2000-0.005-4-4-200-True.npy') # file for
(4,4) Re = 100

[t2_5, enstrophy2_5, change2_5] = [data2_5[:,0], data2_5[:,1], data2_5
[:,3]] # data for (4,4) Re = 2.5
[t5, enstrophy5, change5] = [data5[:,0], data5[:,1], data5[:,3]] # data for
(4,4) Re = 5
[t25, enstrophy25, change25, kinetic25] = [data25[:,0], data25[:,1], data25
[:,3], data25[:,2]] # data for (4,4) Re = 25
[t100, enstrophy100, change100] = [data100[:,0], data100[:,1], data100
[:,3]] # data for (4,4) Re = 100

'''
ENSTROPY PLOTTING
'''

## plot enstrophy for Re = 2.5
growth2_5_5 = 0.03800286486808335 # growth rate for Re=2.5 on (4,4) for
mode kx = 0.5
plt.semilogy(t2_5, enstrophy2_5, label=r'$\tilde{\zeta}(t)$') # enstrophy
plot
plt.semilogy(t2_5, exp(2*growth2_5_5*(t2_5-153.38)+log(8.77e-08)), '--k',
label=r'$\lambda \approx 0.038$') # expected gradient line
plt.xlabel(r'$t$') # x axis label
plt.ylabel(r'$\tilde{\zeta}(t)$') # y axis label
plt.ylim(top=10) # plot limits
plt.xlim([0,500])
plt.legend(loc='upper left') # legend
plt.savefig('figures/enstrophy-(4,4)-Re-2_5_5') # save

```

```

plt.close()

growth2_5_25= 0.03500286486808335 # growth rate for Re=2.5 on (4,4) for
mode kx = 0.25
plt.semilogy(t2_5,enstrophy2_5,label=r'$\tilde{\zeta}(t)$')
plt.semilogy(t2_5,exp(2*growth2_5_25*(t2_5-153.38)+log(8.77e-08)), '--k',
label=r'$\lambda\approx 0.035$')
plt.xlabel(r'$t$') # x axis label
plt.ylabel(r'$\tilde{\zeta}(t)$') # y axis label
plt.ylim(top=10) # plot limits
plt.xlim([0,500])
plt.legend(loc='upper left') # legend
plt.savefig('figures/enstrophy-(4,4)-Re-2_5_25') # save
plt.close()

## plot enstrophy for Re = 5
growth5 = 0.1384715222134374
plt.semilogy(t5,enstrophy5,label=r'$\tilde{\zeta}(t)$')
plt.semilogy(t5,exp(2*growth5*(t5-53.39)+log(2.25e-06)), '--k',label=r'$\lambda\approx 0.138$')
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\zeta}(t)$')
plt.ylim([10e-12,10])
plt.xlim([0,500])
plt.legend()
plt.savefig('figures/enstrophy-(4,4)-Re-5')
plt.close()

## plot enstrophy for Re = 25
growth25 = 0.23534463753988982
plt.semilogy(t25,enstrophy25,label=r'$\tilde{\zeta}(t)$')
plt.semilogy(t25[0:200],exp(2*growth25*(t25[0:200]-34.948)+log(2.188e-05)), '--k',label=r'$\lambda\approx 0.235$')
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\zeta}(t)$')
plt.ylim(top=10)
plt.xlim([0,500])
plt.legend()
plt.savefig('figures/enstrophy-(4,4)-Re-25')
plt.close()

# plot enstrophy for Re = 100
plt.semilogy(t100,enstrophy100,label=r'$\tilde{\zeta}(t)$')
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\zeta}(t)$')
plt.ylim(top=10)
plt.xlim([0,500])
plt.legend()
plt.savefig('figures/enstrophy-(4,4)-Re-100')
plt.close()

'''
MATD PLOTTING
'''

# plot MATD for Re = 2.5
plt.semilogy(t2_5[1:],change2_5[1:])
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\Delta}(t)$')

```

```
plt.savefig('figures/MATD-(4,4)-Re-2_5')
plt.close()

# plot MATD for Re = 5
plt.semilogy(t5[1:], change5[1:])
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\Delta}(t)$')
plt.savefig('figures/MATD-(4,4)-Re-5')
plt.close()

# plot MATD for Re = 25
plt.semilogy(t25[1:], change25[1:])
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\Delta}(t)$')
plt.savefig('figures/MATD-(4,4)-Re-25')
plt.close()

# plot MATD for Re = 100
plt.semilogy(t100[1:], change100[1:])
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\Delta}(t)$')
plt.savefig('figures/MATD-(4,4)-Re-100')
plt.close()
```

A.5.8 Contour Plots for (4,4)

```
from numpy import load, exp, log, pi
import matplotlib.pyplot as plt
from functions_package.mesh import mesh
from scipy.fft import irfft2

ax = 4
ay = 4
N = 128

Nx = N*ax
Ny = N*ay
# domain in x and y
Dx = [0, 2*ax*pi]
Dy = [0, 2*ay*pi]
# create the meshgrid
(X,Y) = mesh(Nx,Ny,Dx,Dy)

'''
FRAME OF STEADY STATE FLOW ON (4,4) FOR RE 2.5
'''
data2_5_1999 = irfft2(load('128-2.5-2000-0.2-4-4-5-True/1999.0.npy')) #
    file for (4,4) Re = 2.5

# steady state at 1999 for RE 25
plt.contourf(X,Y,data2_5_1999)
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.colorbar(label=r'Vorticity $\omega$')
plt.savefig('(4,4)-steady-state-2_5-1999')
plt.close()
```

```

'''
(4,4) RE 25 TURBULENCE PLOT
'''
f01 = irfft2(load('128-25-2000-0.02-4-4-50-True/50.0.npy')) # file for
t=50 (4,4) Re = 25
f10 = irfft2(load('128-25-2000-0.02-4-4-50-True/60.0.npy')) # file for
t=500 (4,4) Re = 25
f11 = irfft2(load('128-25-2000-0.02-4-4-50-True/75.0.npy')) # file for
t=750 (4,4) Re = 25
f12 = irfft2(load('128-25-2000-0.02-4-4-50-True/200.0.npy')) # file for
t=1500 (4,4) Re = 25

fig1, axs = plt.subplots(nrows=2,ncols=2, figsize=(6.2,6),sharex=True,
sharey=True)
(ax1, ax2), (ax3, ax4) = axs # set axis

# plot for t = 50
cs1 = ax1.contourf(X, Y, f01,vmin=-1,vmax=1, levels=10)
ax1.set_ylabel(r'$y$')
ax1.tick_params()
ax1.set_title(r'$Re=25,t=50$')
ax1.set_aspect('equal', adjustable="box")

# plot for t = 60
cs2 = ax2.contourf(X, Y, f10,vmin=-1,vmax=1, levels=10)
ax2.set_title(r'$Re=25,t=60$')
ax2.set_aspect('equal', adjustable="box")

# plot for t = 85
cs3 = ax3.contourf(X, Y, f11,vmin=-1,vmax=1, levels=10)
ax3.set_ylabel(r'$y$')
ax3.set_xlabel(r'$x$')
ax3.tick_params()
ax3.set_title(r'$Re=25,t=75$')
ax3.set_aspect('equal', adjustable="box")

# plot for t = 200
cs4 = ax4.contourf(X, Y, f12,vmin=-1,vmax=1, levels=10)
ax4.set_xlabel(r'$x$')
ax4.tick_params()
ax4.set_title(r'$Re=25,t=200$')
ax4.set_aspect('equal', adjustable="box")

# save figure
plt.tight_layout()
cbar = plt.colorbar(cs1, ax=axs)
cbar.ax.set_ylabel(r'Vorticity $\omega(t)$', rotation=270,labelpad=20)
plt.savefig('Re25Turbulence')
plt.close()

```

A.5.9 Plots for (16, 16)

```

from numpy import load, exp, log, pi
import matplotlib.pyplot as plt
from functions_package.mesh import mesh

```

```

from scipy.fft import irfft2

data2_5 = load('pe-128-2.5-2000-0.2-16-16-5-True.npy')
[t2_5, enstrophy2_5, change2_5] = [data2_5[:,0], data2_5[:,1], data2_5[:,3]]

# plot MATD for Re = 2.5
plt.semilogy(t2_5, change2_5)
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\Delta}(t)$')
plt.savefig('figures/MATD-(16,16)-Re-2_5')
plt.close()

data100 = load('pe-128-100-2000-0.005-16-16-200-True.npy')
[t100, enstrophy100, change100] = [data100[:,0], data100[:,1], data100[:,3]]

# plot MATD for Re = 100
plt.semilogy(t100, change100)
plt.xlabel(r'$t$')
plt.ylabel(r'$\tilde{\Delta}(t)$')
plt.savefig('figures/MATD-(16,16)-Re-100')
plt.close()

'''
RE 100 TURBULENCE PLOT
'''
# grid selection
ax = 16
ay = 16
N = 128

Nx = N*ax
Ny = N*ay
# domain in x and y
Dx = [0, 2*ax*pi]
Dy = [0, 2*ay*pi]
# create the meshgrid
(X, Y) = mesh(Nx, Ny, Dx, Dy)

'''
CONTOUR PLOTS FOR RE = 2.5
'''
f01 = irfft2(load('128-2.5-2000-0.2-16-16-5-True/300.0.npy')) # file
    for t=300 (16,16) Re = 2.5
f10 = irfft2(load('128-2.5-2000-0.2-16-16-5-True/1000.0.npy')) # file
    for t=1000 (16,16) Re = 2.5
f11 = irfft2(load('128-2.5-2000-0.2-16-16-5-True/1250.0.npy')) # file
    for t=1250 (16,16) Re = 2.5
f12 = irfft2(load('128-2.5-2000-0.2-16-16-5-True/1500.0.npy')) # file
    for t=1500 (16,16) Re = 2.5

fig1, axs = plt.subplots(nrows=2, ncols=2, figsize=(6.2, 6), sharex=True,
    sharey=True)
(ax1, ax2), (ax3, ax4) = axs # set axis

# plot for t = 300

```

```

cs1 = ax1.contourf(X, Y, f01, vmin=-1, vmax=1, levels=10)
ax1.set_ylabel(r'$y$')
ax1.tick_params()
ax1.set_title(r'$Re=2.5, t=300$')
ax1.set_aspect('equal', adjustable="box")

# plot for t = 1000
cs2 = ax2.contourf(X, Y, f10, vmin=-1, vmax=1, levels=10)
ax2.set_title(r'$Re=2.5, t=1000$')
ax2.set_aspect('equal', adjustable="box")

# plot for t=1250
cs3 = ax3.contourf(X, Y, f11, vmin=-1, vmax=1, levels=10)
ax3.set_ylabel(r'$y$')
ax3.set_xlabel(r'$x$')
ax3.tick_params()
ax3.set_title(r'$Re=2.5, t=1250$')
ax3.set_aspect('equal', adjustable="box")

# plot for t = 1500
cs4 = ax4.contourf(X, Y, f12, vmin=-1, vmax=1, levels=10)
ax4.set_xlabel(r'$x$')
ax4.tick_params()
ax4.set_title(r'$Re=2.5, t=1500$')
ax4.set_aspect('equal', adjustable="box")

# plot and save results
plt.tight_layout()
cbar = plt.colorbar(cs1, ax=axis)
cbar.ax.set_ylabel(r'Vorticity $\omega(t)$', rotation=270, labelpad=20)
plt.savefig('Re2_5Turbulence(16,16)')
plt.close()

'''
CONTOUR PLOTS FOR RE = 100
'''
f01 = irfft2(load('128-100-2000-0.005-16-16-200-True/40.0.npy')) # file
    for t=40 (16,16) Re = 100
f10 = irfft2(load('128-100-2000-0.005-16-16-200-True/55.0.npy')) # file
    for t=55 (16,16) Re = 100
f11 = irfft2(load('128-100-2000-0.005-16-16-200-True/75.0.npy')) # file
    for t=75 (16,16) Re = 100
f12 = irfft2(load('128-100-2000-0.005-16-16-200-True/200.0.npy')) #
    file for t=200 (16,16) Re = 100

fig1, axs = plt.subplots(nrows=2, ncols=2, figsize=(6.2,6), sharex=True,
    sharey=True)
(ax1, ax2), (ax3, ax4) = axs # set axis

# plot for t = 40
cs1 = ax1.contourf(X, Y, f01, vmin=-1, vmax=1, levels=10)
ax1.set_ylabel(r'$y$')
ax1.tick_params()
ax1.set_title(r'$Re=100, t=40$')
ax1.set_aspect('equal', adjustable="box")

# plot for t = 55
cs2 = ax2.contourf(X, Y, f10, vmin=-1, vmax=1, levels=10)

```

```

ax2.set_title(r'$Re=100,t=55$')
ax2.set_aspect('equal', adjustable="box")

# plot for t=75
cs3 = ax3.contourf(X, Y, f11, vmin=-1, vmax=1, levels=10)
ax3.set_ylabel(r'$y$')
ax3.set_xlabel(r'$x$')
ax3.tick_params()
ax3.set_title(r'$Re=100,t=75$')
ax3.set_aspect('equal', adjustable="box")

# plot for t = 200
cs4 = ax4.contourf(X, Y, f12, vmin=-1, vmax=1, levels=10)
ax4.set_xlabel(r'$x$')
ax4.tick_params()
ax4.set_title(r'$Re=100,t=200$')
ax4.set_aspect('equal', adjustable="box")

# plot and save results
plt.tight_layout()
cbar = plt.colorbar(cs1, ax=axis)
cbar.ax.set_ylabel(r'Vorticity $\omega(t)$', rotation=270, labelpad=20)
plt.savefig('Re100Turbulence(16,16)')
plt.close()

'''
TURBULENT FLOW PLOT
'''

plt.figure(figsize=(10,8)) # figure scaling
plt.contourf(X, Y, f12, vmin=-1, vmax=1, levels=10) # plot the frame at t
=200
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.colorbar(label=r'Vorticity $\omega$') # color bar
plt.savefig('kolmogorov_turbulence_photo') # save fig
plt.tight_layout()
plt.axis('scaled')
plt.show()

```