Bachelor Thesis

Size Convergence of the $\mathbf{E} \times \mathbf{B}$ Staircase Pattern in Flux Tube Simulations of Ion Temperature Gradient-Driven Turbulence

Manuel Lippert

Submission date: 30.06.2023

Physics Department at the University of Bayreuth

Supervisors:
Prof. Arthur G. Peeters
Dr. Florian Rath

★ 29.05.2005 - † 12.05.2023

This thesis is dedicated to my cat **Blacky**
who almost reached the age of 18 years.
She followed me through life since I was 5 years old.
You always were a big part of my life and

I will miss you.

Manuel Lippert
June 24, 2023

Acknowledgement

First, I would like to extend my greatest gratitude to my supervisor Prof. Arthur Peeters and Dr. Florian Rath. I would like to thank both of you for your guidance and support during the submission process of the brief communication and the helping hands during the writing process of this thesis and development of the restart script.

I would like to thank Bernhard Winkler and Markus Hilt for the great technical support regarding my questions to the `btrzx1`-cluster which helped me develop the restart script as it is now.

Also, I would like to thank my best friend, Paul Schwanitz, for his support in many situations and the good discussions in- and outside of academia. In addition I would like to thank Anna-Maria Pleyer, Dominik Müller and Stefan Barthelmann for the great time of sharing the office together.

I thank Anna-Maria Pleyer and my sister Cornelia Lippert for proofreading this thesis and brief communication, which helps me getting better at English spelling.

Outside of academia, I would like to extend my gratitude to my parents, my brother, his wife and daughter, my sister and my girlfriend for their encouragement, endurance and support while writing this thesis.

Abstract

Ion temperature gradient-driven turbulence (ITG) close to marginal stability exhibits zonal flow pattern formation on mesoscales, so-called $\mathrm{E} \times \mathrm{B}$ staircase structures. Such pattern formation has been observed in local gradient-driven flux-tube simulations as well as global gradient-driven and global flux-driven studies.

To reduce the computational effort for the simulations lower input parameter of GKW (Gyro Kinetic Workshop) were tested to find the optimum of minimum resolution for the performed simulations.

For convenience, a `python` script (`slurm_monitor.py`) was written to monitor the simulation on the `btrzx1`-cluster and start/restart until the completion criterion is fulfilled.

Furthermore, it is shown by multiple box size convergence scans that a mesoscale pattern size of $\sim 57 - 76\,\rho$ is inherent to ITG-driven turbulence with Cyclone Base Case parameters in the local limit. This outcome also implies that a typical scale for avalanche-like transport is inherent to ITG-driven turbulence.

Zusammenfassung

Ionen-Temperaturgradienten getriebene Turbulenzen (ITG) weisen nahe niedriger Stabilität Zonal Flow Strukturbildungen, sogenannte $\mathrm{E} \times \mathrm{B}$ Treppenstrukturen, auf Mesoskalen auf. Solche Strukturbildungen wurden sowohl in lokal gradientengetriebene Flussschlauchsimulationen als auch in global gradientengetriebenen und global flussgetriebenen Untersuchungen entdeckt.

Um den Aufwand der Berechnungen der Simulationen zu reduzieren wurden mehrere kleinere Inputparameter für GKW (Gyro Kinetic Workshop) getestet um die optimal kleinste Auflösung für die ausgeführten Simulationen zu finden.

Der Einfachheit halber wurde ein `python`-Skript (`slurm_monitor.py`) geschrieben, was Simulationen auf den `btrzx1`-Cluster überwacht und gegebenenfalls startet/neustartet bis das Kriterium zur Vollendung erfüllt ist.

Weiterhin wurde doch mehrere radiale konvergierende Boxgrößen-Scans gezeigt, dass eine Mesoskalengröße von $\sim 57 - 76\,\rho$ inhärent zur ITG getriebenen Turbulenz mit Cyclone Base Parameter in lokalen Limit ist. Dieses Ergebnis impliziert auch, dass eine typische Skala für den lawinenenartig Transport inhärent für die ITG getriebene Turbulenz ist.

Declaration

The author declares that the materials presented in this thesis are his own work, unless explicitly stated otherwise. This thesis is based on

> LIPPERT, M., RATH, F. & PEETERS, A. G. 2023 Size convergence of the E×B staircase pattern in flux tube simulations of ion temperature gradient driven turbulence. *Physics of Plasma* **30** (7), 969–983

and is a further iteration of this publication (brief communication). It provides additional plots and paragraphs that were included in the publication. The brief communication can be found in Appendix 6.5.

Additionally, the author states that every information, except data, regarding this thesis can be found under the GitHub Repository with the link `https://github.com/ManeLippert/Bachelorthesis-Shearingrate-Convergence`.

Contents

CHAPTER 1

Motivation

Ion temperature gradient driven turbulence close to marginal stability exhibits zonal flow pattern formation on mesoscales, so-called $E \times B$ staircase structures[16]. Such pattern formation has been observed in local gradient-driven flux-tube simulations[43,46], including collisions[57] and background $E \times B$ shear[46], local flux-driven realizations including mean electric field shear[50], as well as global gradient-driven[38,53,51] and global flux-driven[16,17,56,28,29] studies. In global studies, spanning a larger fraction of the minor radius, multiple radial repetitions of staircase structures are usually observed, with a typical pattern size of several ten Larmor radii. By contrast, in the aforementioned local studies the radial size of $E \times B$ staircase structures is always found to converge to the radial box size of the flux tube domain. The above observations lead to the question:

Does the basic pattern size always converges to the box size, or is there a typical mesoscale size inherent to staircase structures also in a local flux-tube description?

The latter case would imply that it is not necessarily global physics, i.e., profile effects, that set

(i) the radial size of the $E \times B$ staircase pattern

(ii) the scale of avalanche-like transport events.

These transport events are usually restricted to $E \times B$ staircase structures and considered as a nonlocal transport mechanism[16].

In this bachlor thesis the above question is addressed through a box size convergence scan of the same cases close to the nonlinear threshold for turbulence generation as studied in Ref. 43.

CHAPTER

Plasma Physics, Zonal Flows and Gyrokinetic Theory

2

2.1 Charged Particle Motion in Magnetic and Electric Field

In magnetic confinement devices like the tokamak reactor, the charged particles experience forces caused by magnetic and electric fields which results in distinct motion under the associated force. Charged particles can be separated in species, e.g. electrons and ions, which will be later on not displayed in the governing equation. Throughout this thesis the charge q, the mass m or the temperature T indicate the quantities of a specific species, i.e., electrons or ions.

2.1.1 Particle Motion perpendicular to the magnetic field

Due to the Lorentz force, particles with a velocity component perpendicular to the homogenous magnetic field $v_\perp$ undergo a circular motion in the plane perpendicular to the magnetic field [Fig. 2.1(a)]. This type of motion has circular frequency, which is often referred to as *cyclotron frequency* and is defined as

$$\omega_\mathrm{c} = \frac{|q|B}{m} \ , \tag{2.1}$$

where m and q are the mass and the charge of the particle and B the strength of the magnetic field. The radius, the so called *Larmor radius*, of this motion is given by

$$\rho_\mathrm{L} = \frac{mv_\perp}{|q|B} \tag{2.2}$$

with the center often being referred to as *gyrocenter*. Note that since the Lorentz force depends on the species charge of the particle, the circulation direction is the opposite between electron in ions.

Due to Coulomb collisions the plasma gets thermalized. Together with the Maxwell-Boltzmann distribution the typical thermal velocity is

$$v_\mathrm{th} = \sqrt{\frac{2T}{m}} \ , \tag{2.3}$$

where T represents the species temperature. Based on the thermal velocity v_th the *thermal Larmor radius* gets introduced as

$$\rho = \frac{mv_\mathrm{th}}{|q|B} \ .^{58} \tag{2.4}$$

2.1.2 Particle Motion parallel to the magnetic field

In absence of forces in the direction parallel to the magnetic field the particles can move freely in parallel direction to the homogenous magnetic field. The velocity of this motion is of order of the thermal velocity v_{th} and is dominated by electrons due to their lighter mass compared to ions ($v_{\mathrm{th,e}}/v_{\mathrm{th,i}} = 60$).

When an electric field with a component parallel to the magnetic field $E_\parallel$ influences the plasma the charged particles are accelerated by the electric force

$$F_{\parallel,E} = qE_\parallel \ . \tag{2.5}$$

The parallel motion follows then from the equation of motion. Here the direction of the motion also depends on the species type [Fig. 2.1(b)].

Since magnetic fields are not always homogenous, an inhomogeneous magnetic field with its gradient ∇B containing a component parallel to the magnetic field which is given by

$$\nabla_\parallel B = \frac{\mathbf{B}}{B} \cdot \nabla B \tag{2.6}$$

causes the force

$$F_{\parallel,\nabla_\parallel B} = -\frac{mv_\perp^2}{2B}\nabla_\parallel B = -\mu\nabla_\parallel B \ ; \qquad \mu = \frac{mv_\perp^2}{2B} \tag{2.7}$$

with *magnetic moment* μ. The magnetic moment μ is an adabatic invariant (constant of motion) if the variation of the magnetic field over time is smaller than the inverse of the cyclotron frequency ω_c^{-1} and the spatial variation is larger the Larmor radius ρ_L. The resulting force has its application in the mirror effect where a charged particle gets reflected due to this force [Fig. 2.1(c)].[58]

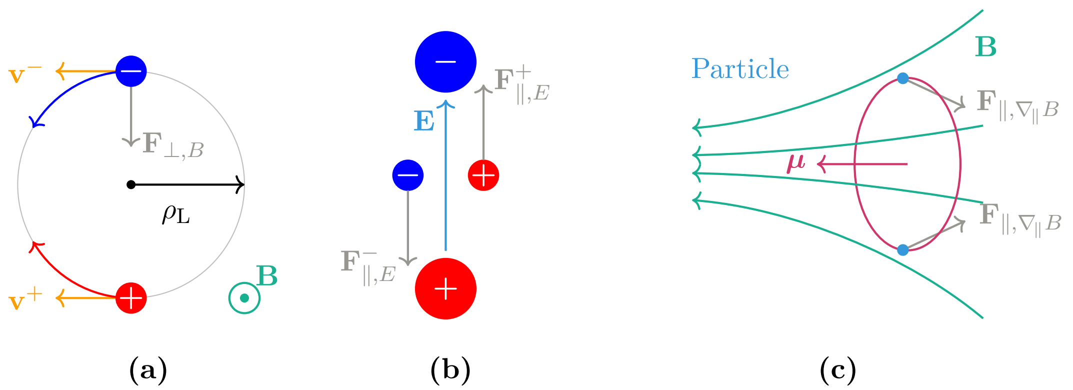


Figure 2.1: Forces acting on a charged particle:

(a) Lorentz force $\mathbf{F}_{\perp,B}$ perpendicular to velocity $\mathbf{v}^\pm$ and magnetic field $\mathbf{B}$ which causes, circular motion with different directions for electron and ions, Lamor radius ρ_L and cyclotron frequency ω_c,

(b) Electric force $\mathbf{F}^\pm_{\parallel,E}$ with electric field $\mathbf{E}$,

(c) Mirror effect with force $\mathbf{F}_{\parallel,\nabla_\parallel B}$ and magnetic moment μ caused by an inhomogeneous magnetic field $\mathbf{B}$.

2.1.3 Drifts in the Gyrocenter

In the presence of a magnetic field (homogenous, inhomogeneous or perturbed) and electric fields the gyrocenter undergoes slow (compared to the thermal velocity v_{th}) drift motions perpendicular to the magnetic field. There are several examples for this drift motion. According to this thesis topic only the main three drift types will be covered in the following.

1. **$\mathbf{E} \times \mathbf{B}$ Drift:**
   If an electric field $\mathbf{E}$ with a perpendicular component together with the magnetic field $\mathbf{B}$ (both fields are homogenous) is present the acting Coulomb force and Lorentz force results into a drift of the gyrocenter with
   $$\mathbf{v}_E = \frac{\mathbf{E} \times \mathbf{B}}{B^2} \tag{2.8}$$
   which is called the E × B drift. Since both acting forces direction depends on the species type the direction of the E × B drift is for every species the same [Fig. 2.2(a)].
2. **$\boldsymbol{\nabla} \boldsymbol{B}$ Drift:**
   Inhomogeneous magnetic field causes a gradient ∇B of the magnetic field. Because of that gradient the gyrocenter undergoes a ∇B drift defined by
   $$\mathbf{v}_{\nabla B} = \frac{m v_\perp^2}{2q} \frac{\mathbf{B} \times \nabla B}{B^3} \ . \tag{2.9}$$
   The gradient of the magnetic field ∇B varies thereby on scales larger compared to the Larmor radius. The direction of the ∇B drift depends on the species type [Fig. 2.2(b)].
3. **Curvature Drift:**
   Due to centrifugal force acting on the particle in a curved magnetic field the gyrocenter experiences a curvature drift according to
   $$\mathbf{v}_\kappa = \frac{m v_\parallel^2}{q} \frac{\mathbf{B} \times \boldsymbol{\kappa}}{B^2} = \frac{m v_\parallel^2}{q} \frac{\mathbf{B} \times \nabla B}{B^3} \ ; \qquad \boldsymbol{\kappa} = -(\mathbf{b} \cdot \nabla)\mathbf{b} = \frac{\nabla B}{B} \ , \tag{2.10}$$
   where $\mathbf{b}$ is the unit vector along the magnetic field. To obtain the result for the curvature $\boldsymbol{\kappa}$ in Eq. (2.10) the plasma pressure has to be small compared to the magnetic field strength B. In the form of Eq. (2.10) ∇B and curvature drift can be treated similarly.[58]

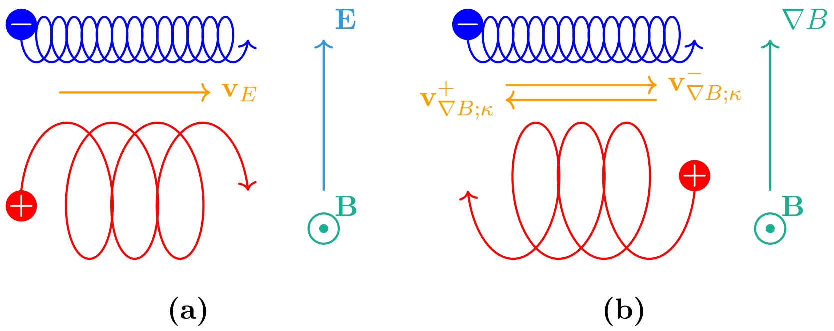


Figure 2.2: Dirft motion in gyrocenter:
(a) $\mathrm{E} \times \mathrm{B}$ Drift with drift velocity $\mathbf{v}_E$, electric field $\mathbf{E}$ and magnetic field $\mathbf{B}$,
(b) ∇B Drift/Curvature Drift with drift velocity $\mathbf{v}^{\pm}_{\nabla B;\kappa}$, magnetic field $\mathbf{B}$ and gradient of the magnetic field ∇B.

2.2 Magnetic Confinement in Tokamak

In tokamak devices strong magnetic fields confine the hot plasma. As mentioned in Chapter 2.1 a magnetic field forces a perpendicular particle motion and a motion which contains the gyro motion and slow perpendicular gyro center drifts. Because of the much smaller size of the Larmor radius compared to the device size R the particle and energy losses are caused by the gyro center drift. To avoid additional loss of particles because of the parallel motion the field lines of the magnetic field in the tokamak devices is shaped like a torus. This type of geometry has nested surfaces with constant magnetic flux, so-called *flux surfaces*, and magnetic field lines which lie on these surfaces. To maintain stability the magnetic field has a toroidal and a poloidal component. According to the force balance the magnetic field is equivalent to the plasma pressure which means on flux-surfaces the plasma pressure is constant.[52,58] The toroidal component is produced by external coils whereas the poloidal component is provided by the toroidal plasma current. Together the components result in a magnetic field which follows helical trajectories [Fig 2.3]. To characterize the quality of confinement the so-called *plasma beta* is used and is given as

$$\beta = \frac{nT}{\mu_0 B^2/2} \ , \tag{2.11}$$

with n the plasma density, T as temperature, μ_0 the permeability in vacuum and the magnetic field strength B. Respectively, the plasma beta compares the thermal plasma pressure nT to the ambient magnetic field pressure $\mu_0 B^2/2$. For fusion devices the plasma beta has to be a bit smaller than 1 ($\beta < 1$) for optimal confinement. In a tokamak reactor the plasma beta has a typical order of a few percent.[58]

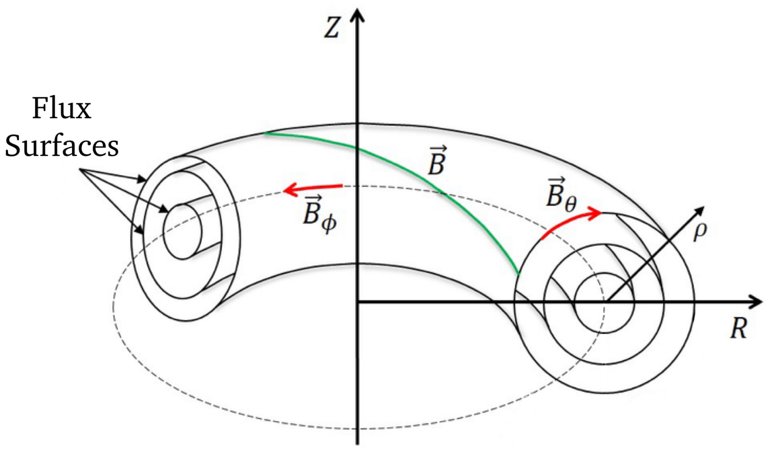


Figure 2.3: Toroidal flux surfaces in tokamak plasma with helical magnetic field (green line) in torus coordinates (ρ (radial), ϕ (toroidal), θ (poloidal)) or cylindrical coordinates (Z, R, ϕ).[3]

2.3 Ion Temperature Gradient (ITG) Driven Instability

The described electrostatic forces acting on the tokamak plasma cause a variety of collective phenomena. The occurrence of microinstabilities is one of these phenomena. They are instabilities on the spatial scale of the Larmor radius ρ_{L} and frequencies much smaller than the cyclotron frequency ω_{c}, driven by density and temperature gradients present in the tokamak devices. Turbulence by microinstabilities are considered as one of the main reason for the loss of particle energy and momentum from tokamak plasma[6,20,26] which limits the confinement time of tokamak reactors. The most dominant instability in tokamak plasma is the ion temperature gradient (ITG) driven instability.[11,12,47]

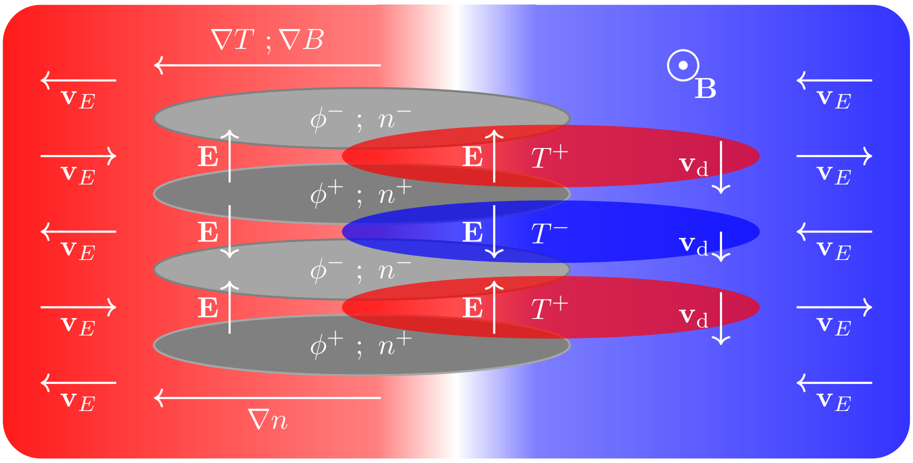


Figure 2.4: Sketch of ion temperature gradient driven instability on the outboard low field side of the tokamak.[10]

The ITG-driven instability is present when a sufficiently large ion temperature gradient occurs and is driven by the so-called bad curvature of the outboard low field side of the tokamak. The mechanism is briefly explained with the help of Fig. 2.4 and Ref. 4,10,13. An ion temperature perturbation ($T^{\pm} \gtrless 0$ blue and red cells) on the outboard low field side of the tokamak causes the modulation of curvature drift and ∇B drift with velocity $\mathbf{v}_{\mathrm{d}}$. The divergence of the velocity $\mathbf{v}_{\mathrm{d}}$ is therefore not zero which results in compression of the plasma ($n_{\pm} \gtrless 0$, gray cells). This relates to the perturbation of the potential ($\phi^{\pm} \gtrless 0$) in the adiabatic electron response. As stated in the equation for the electric field ($\mathbf{E} = -\nabla\phi$) the perturbation of the potential ϕ causes an electric field $\mathbf{E}$ which results in a $\mathrm{E} \times \mathrm{B}$ drift with velocity $\mathbf{v}_E$. The $\mathrm{E} \times \mathrm{B}$ drift direction transports hot plasma in the outer hot region of perturbation ($T^+ > 0$, red cell) and cooler plasma into inner cold region perturbation ($T^- < 0$, blue cell) which reinforce the process and instability grows.

The described mechanism causes perturbation on the outboard low field side of the tokamak. This region is also referred to as “bad curvature” because instability happens there. On the inboard high field side the opposing ∇B and ∇T results in a decay of perturbation which is referred to as “good curvature”. Furthermore, such turbulent structures are called *eddies* which have a typical radial scale of several Larmor radii.[61] The initial perturbation is growing if the normalized temperature gradient overcomes a critical value $R/L_{T,\mathrm{c}}$. For nonlinear turbulence with Cyclone Base Case parameters and circular concentric flux-surfaces the critical value is given by $R/L_{T,\mathrm{c}} \sim 6$.[18,25]

2.4 Zonal Flows and Shearing Rate $\omega_{\mathrm{E}\times\mathrm{B}}$

In the tokamak plasma zonal flows are linked to the $\mathrm{E}\times\mathrm{B}$ flows [Eq.(2.8)] that are connected to the electrostatic potential that is constant on flux-surfaces, also called *zonal potential.*[14] Because of the variation exclusively across flux-surfaces, the zonal flow is tangential to those surfaces and does not contribute to the turbulent transport across the flux-surfaces caused by the $\mathrm{E}\times\mathrm{B}$ motion. Zonal flows are driven nonlinearly by turbulent Reynolds stresses.[15] For more details about the mechanism behind zonal flow generation the reader is referred to Ref. 6,19,24,32.

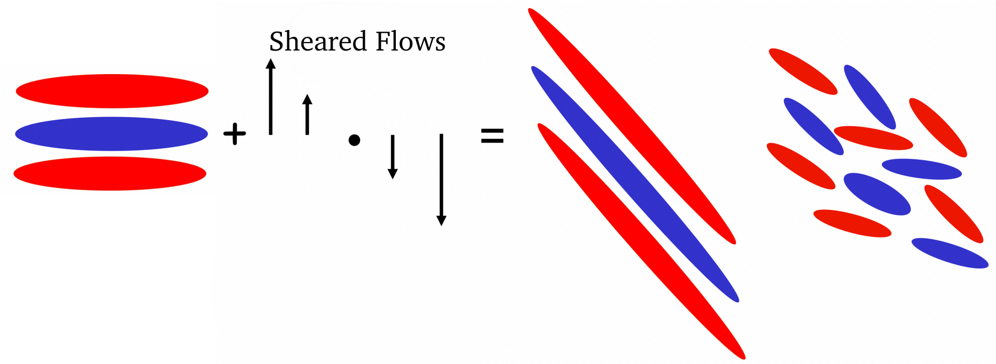


Figure 2.5: Subdued turbulence because of stretched eddies, which breaks in smaller structures caused by sheared $\mathrm{E}\times\mathrm{B}$ drifts.[23]

Furthermore, zonal flows are connected with the radially alternating zonal potential, which introduces a velocity shear, i.e., a radial variation of the zonal flow velocity. This causes a shear deformation of turbulent structures which is considered to play a significant role in the suppression of turbulence[5,18], especially the ion temperature gradient driven turbulence[41,37,36,59,60]. The shearing process tilts and stretches turbulent eddies, caused by the shear flow, until they break into smaller eddies with reduced radial correlation length [Fig. 2.5], characterizing the shearing as a decorrelation process. Turbulence will then be reduced because of this process.[5,14,7] The nonlinear generation mechanism causes a shift, also known as Dimits shift[18], of the critical temperature gradient $R/L_{T,\mathrm{c}}$ which leads to a development of ITG-driven turbulence at higher R/L_T compared to the linear threshold.

When the zonal flows are strong enough to suppress turbulence completely due to the shearing process, pattern formation occurs in the stabilized tokamak plasma the so-called $\mathrm{E} \times \mathrm{B}$ staircase structure. The $\mathrm{E} \times \mathrm{B}$ staircase pattern has the following properties:

(1) They occur on a *radial mesoscale* of order of $10\,\rho$ which is larger than the Larmor radius but smaller than the machine size R.

(2) The structure are *quasi stationary* in space and vary on time scales much longer than the typical turbulence time scales.

(3) They have a *typical amplitude* in terms of the $\mathrm{E} \times \mathrm{B}$ shearing rate of the order of $10^{-1} v_{\mathrm{th}}/R$.

(4) Turbulent transport occurs in *avalanches* with propagation strongly linked to the local $\mathrm{E} \times \mathrm{B}$ shearing rate.[38]

The $\mathrm{E} \times \mathrm{B}$ staircase pattern is manifest as radial structure formation in the $\mathrm{E} \times \mathrm{B}$ shearing rate defined as[45,44,43]

$$\omega_{\mathrm{E}\times\mathrm{B}} = \frac{1}{2}\frac{\partial^2 \langle\phi\rangle}{\partial x^2} \;, \tag{2.12}$$

where $\langle\phi\rangle$ is the zonal electrostatic potential normalized by $\rho_* T/e$ ($\rho_* = \rho/R$ is the thermal Larmor radius normalized with the major radius R, T is the temperature and e is the elementary charge). The zonal potential is calculated from the electrostatic potential ϕ on the two-dimensional x-y-plane at the low field side according to Ref. 46

$$\langle\phi\rangle = \frac{1}{L_y}\int_0^{L_y} \mathrm{d}y \; \phi(x, y, s = 0) \;. \tag{2.13}$$

The $\mathrm{E} \times \mathrm{B}$ shearing rate $\omega_{\mathrm{E}\times\mathrm{B}}$ is the radial derivative of the advecting zonal flow velocity[21,54] and quantifies the zonal flow induced shearing of turbulent structures[5,21,7]. This type of structure is characterized by intervals of almost constant shear with alternating sign satisfying the Waltz criterion $|\omega_{\mathrm{E}\times\mathrm{B}}| \approx \gamma$[55,54] ($\gamma$ is the growth rate of the most unstable linear ion temperature gradient (ITG)-driven Eigenmode), connected by steep flanks where $\omega_{\mathrm{E}\times\mathrm{B}}$ crosses zero.

As mentioned before the turbulent transport occurs through avalanches which also mediate the turbulence of the ITG-driven turbulence in the adabatic electron approximation. Avalanches are initiated when the $\mathrm{E}\times\mathrm{B}$ sharing rate has a zero crossing with a steep flank and propagates outwards with negative shearing and inwards with positive shearing.[27,38] The fully developed staircase results in almost completely quenched turbulence. Its development results in a discontinuous step in the dependency of turbulent fluxes on the temperature gradient also known as *finite heat flux threshold.*[43,57]

2.5 Gyrokinetic Theory

2.5.1 Vlasov Equation

Because of the large number of particles in the fusion plasma a prediction on the basis of Newton-Maxwell dynamics results in an impossible task for simulation, but this problem can be solved with a statistical approach. For that the particle density distribution function $f(\mathbf{x}, \mathbf{v}, t)$ in the six dimensional phase space $\{\mathbf{x},\ \mathbf{v}\}$ with the particles position $\mathbf{x}$ and velocity $\mathbf{v}$ is needed. Because collisions are happening at much smaller frequencies than the characteristic frequencies connected to turbulence, the collisionless model is often preferred[20] which results through evolution of the particle density distribution function in the *Vlasov equation*

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial \mathbf{x}} \cdot \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} + \frac{\partial f}{\partial \mathbf{v}} \cdot \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = 0 \ . \tag{2.14}$$

To obtain a closed system the Maxwell equations with the particle density n and current density j can be described with the distribution function as follows

$$n = \int \mathrm{d}\mathbf{v}\, f(\mathbf{x}, \mathbf{v}, t) \qquad j = q \int \mathrm{d}\mathbf{v}\, v f(\mathbf{x}, \mathbf{v}, t) \ , \tag{2.15}$$

which are then substituted into the Maxwell equations

$$\begin{aligned} \nabla \cdot \mathbf{B} &= 0 & \nabla \times \mathbf{B} &= \mu_0 \left(\sum_{\text{species}} j + \epsilon_0 \frac{\partial \mathbf{E}}{\partial t} \right) \\ \nabla \cdot \mathbf{E} &= \frac{1}{\epsilon_0} \sum_{\text{species}} qn & \nabla \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \ . \end{aligned} \tag{2.16}$$

The Vlasov equation (2.14) in combination with the Maxwell equations (2.15) and (2.16) is the basis of the gyrokinetic model.[31]

2.5.2 Gyrokinetic Ordering

The typical spatio-temporal scales connected to the dynamics in a tokamak plasma allow for the so-called *gyrokinetic ordering* as outlined below. The fast gyromotion is much smaller compared to typical time scales connected to turbulence ($\omega/\omega_\mathrm{c} \sim 10^{-3}$). The length scales of the turbulence are associated with the wave vector $\mathbf{k}$ which can be seperated into a perpendicular component $k_\perp = |\mathbf{k} \times \mathbf{b}|$ and a parallel component $k_\| = |\mathbf{k} \cdot \mathbf{b}|$ where $\mathbf{b}$ is parallel to the poloidal component of the magnetic field. The perpendicular component $k_\perp$ is of the order of the thermal Larmor radius $k_\perp^{-1} \sim \rho$ which is significantly smaller than the scale on which the equilibrium density n_0 varies, which can be expressed with the gradient length $L_\mathrm{n} = |\nabla \ln n_0|^{-1}$. The gradient length compares with the machine size R which leads to the normalized thermal Larmor radius $\rho_* = \rho/R \sim 10^{-3} - 10^{-4}$ with R as major radius of the tokamak. The parallel component $k_\|$ on the other hand scales with the machine size R. Experiments show that in the core plasma the fluctuation amplitude of the density perturbation $\delta n/n_0$ and magnetic field fluctuation $\delta B/B_0$, with B_0 as strength of the magnetic field in equilibrium, is of order $\sim 10^{-4}$. Together this separation result in the following

$$\frac{\omega}{\omega_\mathrm{c}} \sim \frac{k_\|}{k_\perp} \sim \frac{\rho}{L_\mathrm{n}} \sim \frac{\delta n}{n_0} \sim \frac{\delta B}{B_0} \sim \frac{v_\mathrm{d}}{v_\mathrm{th}} \sim \epsilon_\mathrm{g} \tag{2.17}$$

with $\epsilon_\mathrm{g} \ll 1$ which applies to the typical dynamics of a fusion core plasma.[6,20] The gyrokinetic ordering allows to formulate reduced governing equations referred to as the gyrokinetic formalism.

2.5.3 Gyrokinetic Equation

For the description of charged particle behaviour in the tokamak device the *guidingcenter coordinates* are used [Fig. 2.6]. In this set of coordinates the guidingcenter follows the magnetic field with the parallel velocity $v_\parallel$. The gyro motion is described together with the magnetic moment $\mu = \frac{1}{2} m v_\perp / \omega_c$, the gyrocenter $\mathbf{X}$ and the gyro phase ζ which gives a parameter set of six quantities $\{\mathbf{X}, v_\parallel, \mu, \zeta\}$. The gyrocenter position $\mathbf{X}$ is expressed with the particle position $\mathbf{x}$ and the position Larmor radius vector $\boldsymbol{\rho}_\mathbf{L}$ as $\mathbf{X} = \mathbf{x} - \boldsymbol{\rho}_\mathbf{L}$.

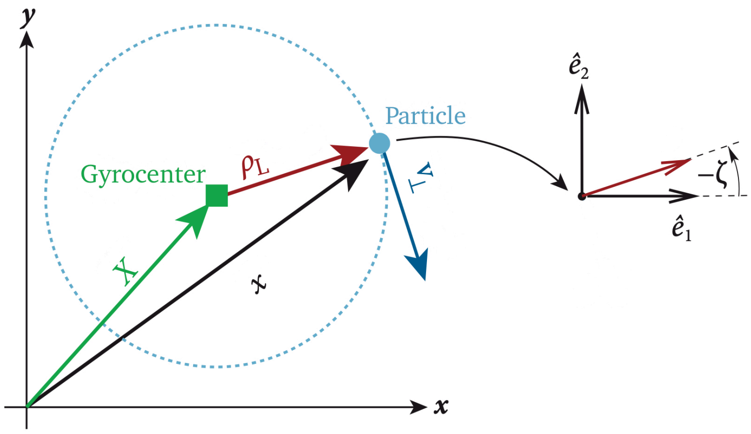


Figure 2.6: Sketch of guidingcenter coordinates where the charged particle performs a circular motion around the gyrocenter.[32]

In the next step the unperturbed Lagrangian is expressed through the guidingcenter coordinates. The particle velocity $\mathbf{v}$ is decomposed into a parallel component $v_\parallel = \mathbf{v} \cdot \mathbf{b}$ and a perpendicular component $v_\perp = |\mathbf{v} \times \mathbf{b}|$ which relates to the magnetic moment μ. As final step the Lagrangian is transformed into gyrocenter phase space, based on the so-called Lie perturbation theory. This method allows to eliminate the gyrophase-dependent contributions, rendering the gyrophase ζ an ignorable coordinate and the magnetic moment μ an exact constant of motion.[20,8,9] Taking everything into account the new coordinates of the Vlasov equation are $\{\mathbf{X}, v_\parallel, \mu\}$ which transforms the Vlasov equation into the *gyrokinetic equation*, describing the evolution of the gyrocenter distribution function.

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial \mathbf{X}} \cdot \frac{\mathrm{d}\mathbf{X}}{\mathrm{d}t} + \frac{\partial f}{\partial v_\parallel} \cdot \frac{\mathrm{d}v_\parallel}{\mathrm{d}t} = 0 \; . \tag{2.18}$$

The time derivative of the magnetic moment μ is zero because the magnetic moment is an exact invariant.

2.5.4 δf Approximation and Local Limit

All simulations in this thesis are performed with GKW (Gyro Kinetic Workshop) which solves the gyrokinetic equation in the δf approximation and local limit. For the δf approximation the particle density distribution function f is seperated into perturbation δf and equilibrium f_0 (constant in time) as expressed in the gyrokinetic ordering (2.17). This is given by

$$f = f_0 + \delta f \qquad \delta f \sim \rho_* f_0 \ . \tag{2.19}$$

In GKW the equilibrium distribution function f_0 is set to the Maxwell distribution function. The perturbed distribution function δf is considered to vary perpendicular to the poloidal magnetic field of order of the Larmor radius while the equilibrium changes with the system size which results in the ordering

$$\nabla_\perp(\delta f) = \nabla_\perp(f_0) \ . \tag{2.20}$$

The δf approximation applies for global and local description where the latter is used in this thesis. The *local limit* on the other hand uses the spatial separation of equilibrium and perturbation and implies that all equilibrium and geometry quantities are considered constant over the radial extent of the simulation volume. Because of the tokamak symmetry all equilibrium quantities are also constant in the toroidal direction where the dependency of the equilibrium magnetic field on the poloidal angle is retrained in general.

In addition to this, in GKW the following coordinates get applied with the parameter $\mathbf{X} = \{x, y, s\}$. Here, x is the radial coordinate that labels the flux surfaces normalized by the thermal Larmor radius ρ and y labels the field lines and is an approximate binormal coordinate. Together with the coordinate s which parameterizes the length along the field lines and is referred to as the parallel coordinate these quantities form the Hamada coordinates[22]. This results in the coordinates $\{x, y, s, v_\parallel, \mu\}$. Combining the δf approximation and the Hamada coordinates the gyrokinetic equation can be formulated as follows

$$\frac{\partial g}{\partial t} + \mathbf{v}_\chi \cdot \nabla g + (v_\parallel \mathbf{b} + \mathbf{v}_\mathrm{D}) \cdot \nabla(\delta f) - \frac{\mu B}{m} \frac{\mathbf{B} \cdot \nabla B}{B^2} \frac{\partial(\delta f)}{\partial v_\parallel} = S \ , \tag{2.21}$$

where g is a function which contains the perturbation δf and the equilibrium f_0. Furthermore, $\mathbf{v}_\mathrm{D}$ and $\mathbf{v}_\chi$ are drift velocities which are caused by the inhomogeneous magnetic field ($\mathbf{v}_\mathrm{D}$) and through the $\mathrm{E} \times \mathrm{B}$ drift ($\mathbf{v}_\chi$). The source term S contains the distribution function in the equilibrium f_0 and a correction term for the collisions as well as the energy injection term. Due to periodic boundary conditions in the radial direction the radial averaged gradients are fixed in time which is also referred to as *gradient-driven* approach.[42]

CHAPTER 3

Methods and Material

3.1 Simulation Setup

The gyrokinetic simulations are performed with the non-linear flux tube version of Gyrokinetic Workshop (GKW)[42] with adiabatic electron approximation. In agreement with Ref. 43, Cyclone Base Case (CBC) like parameters are chosen with an inverse background temperature gradient length $R/L_T = 6.0$ and circular concentric flux surfaces. The numerical resolution is compliant to the "Standard resolution with 6th order (S6)" set-up of the aforementioned reference. A summary of the numerical parameters is given in Table 4.1 and for more details about the definition of individual quantities the reader is referred to Refs. 42,43.

	N_m	N_x	N_s	$N_{\nu_\parallel}$	N_μ	D	ν_d	$D_{\nu_\parallel}$	D_x	D_y	Order	$k_y\rho$	$k_x\rho$
S6	21	83	16	64	9	1	$\lvert\nu_\parallel\rvert$	0.2	0.1	0.1	6	1.4	2.1

Table 3.1: Resolution used in this work: Number of toroidal modes N_m, number of radial modes N_x, number of grid points along the magnetic field N_s,number of parallel velocity grid points $N_{\nu_\parallel}$, number of magnetic moment grid points N_μ, dissipation coefficient used in convection along the magnetic field D,the velocity in the dissipation scheme ν_d, dissipation coefficient used in the trapping term $D_{\nu_\parallel}$, damping coefficient of radial modes D_x, damping coefficient of toroidal modes D_y, order of the scheme used for the zonal mode, maximum poloidal wave vector $k_y\rho$, and maximum radial wave vector $k_x\rho$.

Consistent with Ref. 43 the turbulence level is quantified by the turbulent heat conduction coefficient χ, which is normalized by $\rho^2 v_\mathrm{th}/R$ ($v_\mathrm{th} = \sqrt{2T/m}$ is the thermal velocity and m is the mass). Furthermore, quantities ρ, R, T, v_th and m are referenced quantities from Ref. 43,42.

3.2 Diagnostics

In order to diagnose the temporal evolution of the staircase pattern and to obtain an estimate of its amplitude the radial Fourier transformation of the $\mathrm{E} \times \mathrm{B}$ shearing rate is considered. It is defined by

$$\omega_{\mathrm{E}\times\mathrm{B}} = \sum_{k_{\mathrm{ZF}}} \widehat{\omega}_{\mathrm{E}\times\mathrm{B}}(k_{\mathrm{ZF}}, t)\ \exp(\mathrm{i}k_{\mathrm{ZF}}x)\ , \tag{3.1}$$

where $\widehat{\omega}_{\mathrm{E}\times\mathrm{B}}$ is the complex Fourier coefficient and $k_{\mathrm{ZF}} = 2\pi n_{\mathrm{ZF}}/L_x$ defines the zonal flow wave vector with the zonal flow mode number n_{ZF} ranging in $-(N_x-1)/2 \leq n_{\mathrm{ZF}} \leq (N_x-1)/2$ and the radial box size L_x. Based on the definitions above, the shear carried by the zonal flow mode with wave vector k_{ZF} is defined by $|\widehat{\omega}_{\mathrm{E}\times\mathrm{B}}|_{n_{\mathrm{ZF}}} = 2|\widehat{\omega}_{\mathrm{E}\times\mathrm{B}}(k_{\mathrm{ZF}}, t)|$. In general, the zonal flow mode that dominates the $\mathrm{E} \times \mathrm{B}$ staircase pattern, also referred to as the *basic mode* of the pattern in this work, exhibits the maximum amplitude in the spectrum $|\widehat{\omega}_{\mathrm{E}\times\mathrm{B}}|_{n_{\mathrm{ZF}}}$.

3.3 btrzx1 Cluster

The simulation itself will be performed on the `btrzx1`-cluster of the University Bayreuth. This cluster has a wall time of 24 hours, an total amount of 345 compute nodes with two AMD Epyc Processors (2nd generation) with 16 cores each and 128 GB of main memory each node and as resource manager `Slurm`.[49]

The simulations are performed on the `/scratch` directory because the `/home` folder has a disk space limit of 80 GB which is not enough to perform long runs of GKW.

3.4 Restart Script for Simulation

As stated in Chapter 3.3 the `btrzx1`-cluster has a wall time of 24 hours but since simulations typically run longer than 24 hours, the simulation has to be restarted after the wall time is exceeded. Some simulations need multiple weeks making a regular restart vital to obtaining complete sets of simulation data. The following restart script was written to outsource and simplify the process.

As script language `python` was chosen because of its simple syntax and the variety of tools compatible with the `bash` shell. `bash` was also considered, but after some issues occurred regarding the mechanism to check whether a file exist or not, `python` was selected. Furthermore, the script uses the standard `python3` library and is therefore executable with any `python3` environment.

The script was named `slurm_monitor.py` to indicate that this script works with the `Slurm` resource manager. The source code was pushed to the GKW repository on BitBucket[34] or can be found in Appendix 6.1

3.4.1 How to run Restart Script in Terminal

To run the restart script `python3` has to be installed on the system and should be accessible via the command line. The monitoring can be started with the following command:

```
python3 -u slurm_monitor.py --job-name $JOBNAME
```

Additional parser options can be added to the command above which will be summarized in the upcoming Chapter 3.4.2. The script was build to run in the background of the terminal for which there are two approaches:

1. `nohup`:
   Runs command in the background while terminal can still be used or closed.[1] The command for this method would be:

   ```
   nohup python3 -u slurm_monitor.py --job-name $JOBNAME &> /dev/null &
   ```

   and to cancel the monitoring:

   ```
   python3 -u slurm_monitor.py --job-name $JOBNAME --kill
   ```

2. `screen`:
   Opens a virtual terminal session which is detached from the main terminal (like a window) and can be entered and closed.[2] Here the command would be:

   ```
   screen -S $SESSION #Create screen session
   python3 -u slurm_monitor.py --job-name $JOBNAME --verbose
   ```

   and to cancel the monitoring use the keyboard shortcut ctrl+C or kill screen itself with ctrl+D. To leave the screen use (ctrl+A) + D and to enter the screen use:

   ```
   screen -r $SESSION
   ```

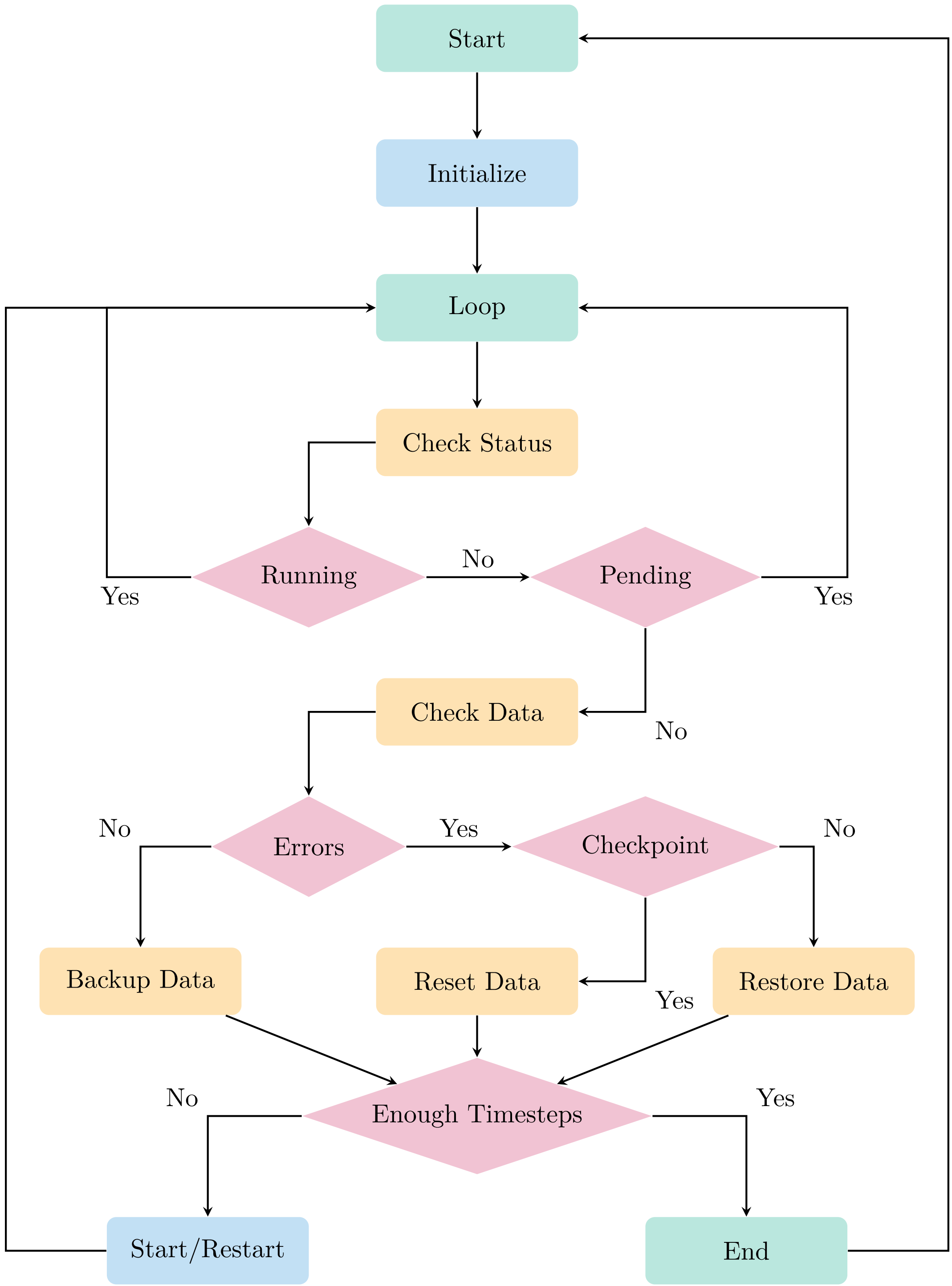
3.4.2 Features of Restart Script

The restart script has several features which can be used via the command line interface as parser options. The following features with the corresponding parser option were implementing:

- **!Required!** Set job name not longer than 8 characters to identify simulation
  `-j [JOBNAME], --job-name [JOBNAME]`
- Can be started from everywhere with simulation folder path
  `-d [DIRECTORY], --dir [DIRECTORY]` `(default=cwd)`
- Creates jobscript file with defined content (variable `jobscriptContent`)
  `--jobscript [JOBSCRIPTFILE]` `(default=jobscript-create)`
  `--nodes [NODES]` `(default=3)`
  `--ntask-per-node [TASKS]` `(default=32)`
  `--walltime [WALLTIME]` `(d-hh:mm:ss)` `(default=0-24:00:00)`
- Start/Restarts simulation until job criteria is suffused
  `-n [TIMESTEPS]` `(default=10000)`
  `--restartfile [RESTARTFILE]` `(default=FDS.dat)`
- Makes backup after each run before Restart and Restore files after failed run
  Quicklinks: `local` (simulation folder), `home` (home folder)
  `-b [LOCATION], --backup [LOCATION]` `(default=None)`
- Reset option after run fails and dump files were written
  (rely on `h5py`, `pandas` and `numpy` which get installed by the script itself)
  `-r, --reset` `(default=False)`
- Creates status file with current status and progress bars and updates it dynamically
  `--statusfile [STATUSFILE]` `(default=status.txt)`
- Set form of the output table in status file
  Options: fancy (round box), universial (crossplattform), None (No frame)
  `--format [FORMATTABLE]` `(default=None)`
- Sends mail at the beginning, end and restart with status file as attachment
  (`mailx` other equivalent has to be installed, look into `send_mail` function for more info)
  `--mail [MAILADDRESS]` `(mail@server.de)` `(default=None)`
  `--restart-mail` `(default=False)`
- Kills monitoring process by using the `PID` number
  `--kill` `(default=False)`
- Set sleep interval after which the status of the current simulation should be checked
  `--refresh-rate [SLEEPTIME]` `(default=300)`
- Print output of status file table to console
  `-v, --verbose` `(default=False)`

All boolean parameters are switches, i.e., the parser option only changes the value from `False` to `True`.

3.4.3 General Structure



3.4.4 Status File created from Restart Script

As stated in Chapter 3.4.2 the restart script is writing a status file with the current status and progress bars of the simulation. The output is formatted as a table and can be adjusted in its appearance with the `--format` parser option [Chapter 3.4.2]. The output types written in the status file are listed in Table 3.2. To each output the restart script writes the date, time and the duration of the monitoring additionally into the table.

Type	Information
`STARTING`	Start monitoring process or Start/Restart of simulation
`CONTINUE`	If the simulation has performed at least one run
`CONTROL`	Check the current time steps of the simulation
`SUCCESS`	Stop monitoring because the time steps condition has been fulfilled
`IMPORT`	Modules `h5py`, `pandas` and `numpy` will be loaded
`INSTALL`	Script installs the modules `h5py`, `pandas` and `numpy` on the user level
`CHECK`	Required modules to use the reset option are already installed
`WAITING`	Simulation is pending in the queue of Slurm
`RUNNING`	Simulation is running on the server
`BACKUP`	The script performs data backup to the predefined location
`RESTORE`	The script reloads the data from the backup folder if errors occur
`RESET`	The simulation restes with the use of dump files
`ERROR`	An error occur during monitoring Types: No executable (`gkw.x`), walltime, timeout, no config, `hdf5`, string error

Table 3.2: Output types in status file

To continue, the status file has different style options, e.g., `fancy`, `universal` and `None` which are displayed in Fig. 3.1.

```
╒════════════════╕   +---------------+   -----------------
│     fancy      │   |   universal   |         None
╞════════════════╡   +---------------+   -----------------
│                │   |               |
╘════════════════╛   +---------------+   -----------------
```

Figure 3.1: Styles of output table in status file.

In addition to that, it dynamically updates the progress of simulation to the end of the status file in form of a progress bar. The progress bar contains the current time step of the simulation in contrast to the required time steps and the progress of the run itself in seconds in relation to the wall time. It also contains the current output from `squeue`, if the simulation is running or pending, else the important information of the simulation. An example progress bar is shown in Fig. 3.2.

```
+------------------------------------------------------------+
|  JOBID  NAME     USER ST    TIME NODES  NODELIST(REASON)   |
| 970455 3x1.5 bt712347 CG 3:16:01     3  r03n03,r05n[15-16] |
+------------------------------------------------------------+
| PROGRESS  [===================>.....]  ( 80%)  20000/25000 |
| RUN 13    [===>.......................]  ( 13%)  11462/86400 |
+------------------------------------------------------------+
```

Figure 3.2: Progress bar example.

The progress bar displays all important information at one glance. To boost productivity the following command is recommended:

```
cd $DATA
find . -name $STATUSFILENAME -exec tail -8 {} \;
```

which prints all progress bars of every simulation to the data folder. If one wants to see the complete status file change `tail -8` to `cat`. An example status file is displayed in the Appendix 6.2.

3.4.5 Support for other Resource Managers

The general idea behind the restart script could be adapted for other resource managers. For example in the early stage of the development the script was changed to support the `PBS/Torque` resource manager which mainly runs on the `btrzx2`-cluster of the University of Bayreuth[48] and supports the research of Dominik Müller for his Bachelor Thesis[40] with the `RHMD`-code[39]. The script was named `torque_monitor.py` and can be found in the GitHub Repository of this thesis[33] or in the Appendix 6.3. But note that due to the early adaptation the code does lack many features of `slurm_monitor.py` and was not maintained actively.

CHAPTER 4

Results and Discussion

The performed simulations for this chapter are documented in Appendix 6.4.

4.1 Variation of Computational Resolution

At the beginning of this work the goal is to estimate the minimal resolution needed to run the simulation without numerical dissipation biasing the physical outcome. Numerical dissipation can therefore result to no formation of zonal flow structures, which cause a permanent turbulent state of the simulation. The goal behind this testing is to reduce **calculation time** and **costs** of the simulation.
Because of that, the criteria for the best resolution should be:

(1) Subdued turbulence after **short** time periods

(2) Stability for **long** time periods

and the following procedure will be applied for verification:

1. Reduce only one number of grid points and look if criterias **(1)**, **(2)** are satisfied
2. Reduce to known the minimum number of grid points to verify result in general.

4.1.1 Benchmark

Starting from the "Standard resolution with 6th order (S6)" [Table 3.1] the first step is to reproduce the result of Ref. 43 in Section IV. Note that, because of selected circular concentric geometry the used inverse background temperature gradient length is $R/L_T = 6.0$ instead of $R/L_T = 7.0$ which was used in Section IV of Ref. 43 for s-α geometry. In Fig. 4.1 the obtained data is similar to the results from Ref. 43 with subdued turbulence after $\sim 3000\, R/v_{\mathrm{th}}$.

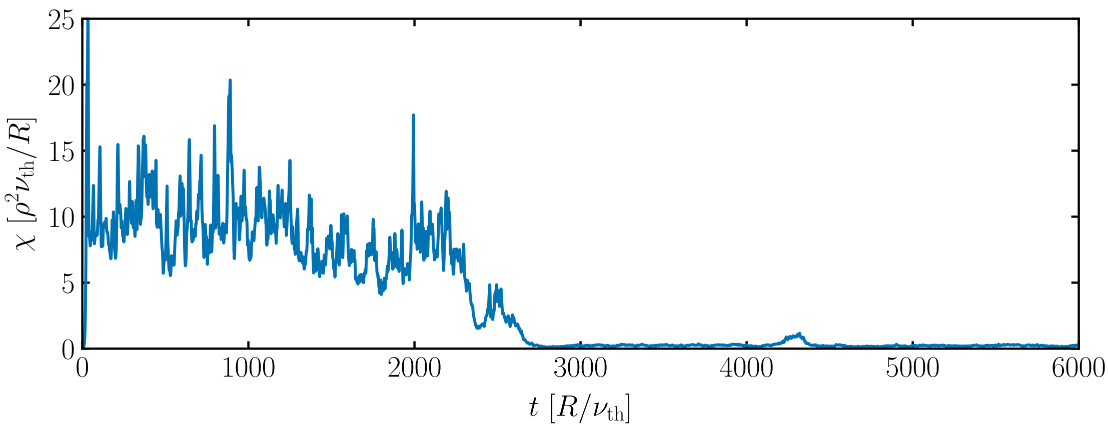


Figure 4.1: Time trace of the heat conduction coefficient χ for $R/L_T = 6.0$ for benchmark.

As next step to verify the selection of the gradient length R/L_T an approach to the finite heat flux threshold was made. As Ref. 43 in Section V concludes the finite heat flux threshold is approximately located at a gradient length of $R/L_T = 6.3$ for circular geometry [FIG. 4 of Ref. 43]. Therefore, following parameters were used

$$R/L_T \in [6.0, 6.3] \ .$$

As seen in Fig. 4.2 for $R/L_T = 6.3$ no suppression of turbulence occurs in the whole time domain, which is in agreement with Ref. 43.

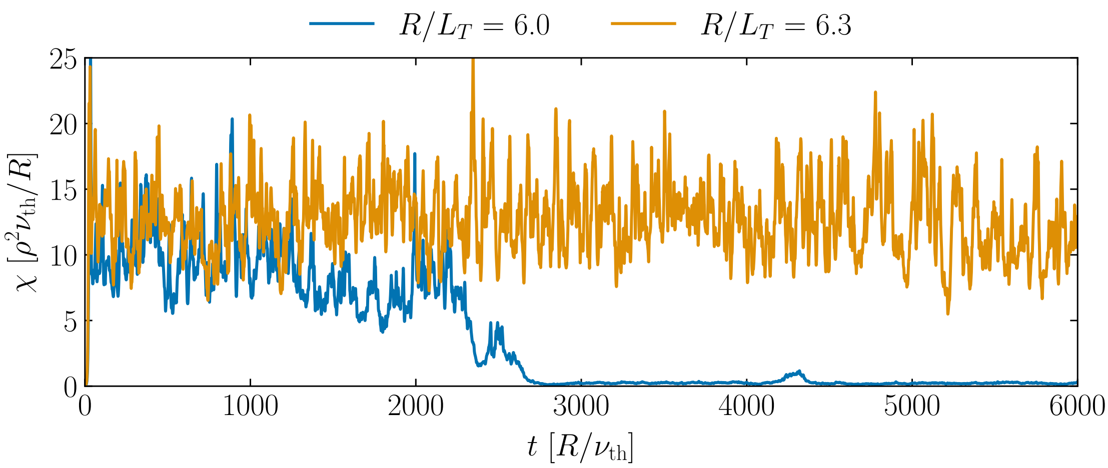


Figure 4.2: Time traces for $R/L_T = 6.0$ and $R/L_T = 6.3$ for benchmark.

4.1.2 Reduction of parallel Velocity Grid Points $N_{\nu_\parallel}$

In the following the number of grid points for the parallel velocity $N_{\nu_\parallel}$ is reduced to

$$N_{\nu_\parallel} \in [16, 32, 48, 64] \ .$$

In Fig. 4.3 is clearly visible that the resolution with $N_{\nu_\parallel} = 16$ is not suitable for criteria **(1)** because here the turbulence is not subdued after a long time period. But resolution $N_{\nu_\parallel} = 32$ is not acceptable as well according to criteria **(2)** since the suppressed turbulence state gain instability after $\sim 3000\, R/v_{\rm th}$.
So to conclude only grid points with $N_{\nu_\parallel} = 48, 64$ are satisfying the set of criteria. Due to criteria **(1)** the selected resolution is

$$\boxed{N_{\nu_\parallel} = 48} \ .$$

With this number of grid points the time until turbulence subdued is halfed compared to the benchmark case, i.e., turbulence suppression occur after $\sim 1500\, R/v_{\rm th}$.

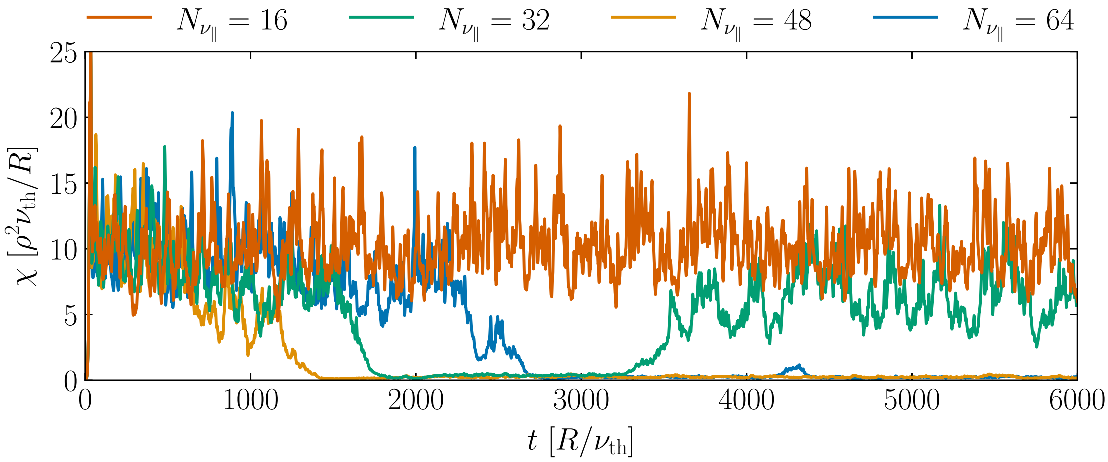


Figure 4.3: Time traces for reduced parallel velocity grid points $N_{\nu_\parallel}$.

4.1.3 Reduction of Magnetic Moment Grid Points N_μ

In the next step the number of grid points for the magnetic moment N_μ were reduced with

$$N_\mu \in [6, 9] \ .$$

As in Fig. 4.4 shown, the reduction of grid points for the magnetic moment does not significantly impact the suppression of turbulence. The turbulence enters the stationary state in both cases after $\sim 3000\, R/v_{\mathrm{th}}$.

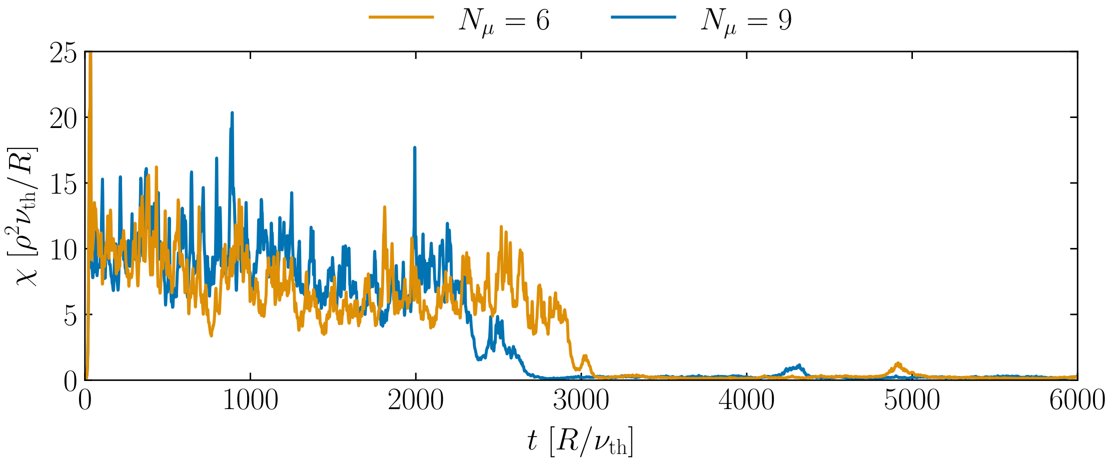


Figure 4.4: Time traces for reduced magnetic moment grid points N_μ.

To conclude a final result the number of grid points for the parallel velocity got reduced to $N_{\nu_\parallel} = 48$ according to Chapter 4.1.2. In this case the turbulence does not subdue for the resolution $N_\mu = 6$ which leads, with both criteria in mind, to the following resolution

$$\boxed{N_\mu = 9} \ .$$

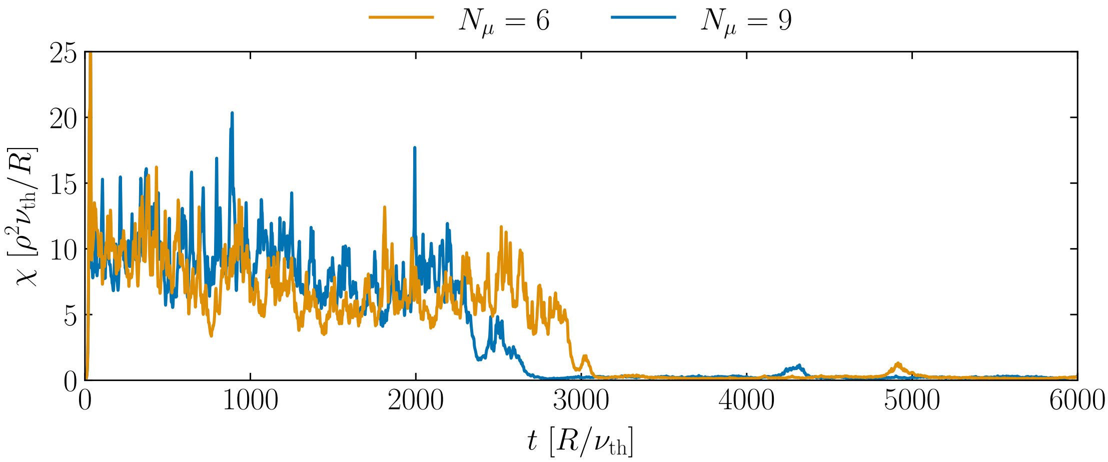


Figure 4.5: Time traces for reduced magnetic moment grid points N_μ.

4.1.4 Reduction Grid Points along of the Magnetic Field N_s

In the final step the number of grid points along the magnetic field N_s gets reduced with the following parameters

$$N_\mathrm{s} \in [12, 16] \ .$$

In Fig. 4.6 can be seen that the reduction to $N_\mathrm{s} = 12$ does not satisfy the criteria **(2)** because the stationary state of the turbulence gets instabil after $\sim 2500\, R/v_\mathrm{th}$. This concludes to the following resolution

$$\boxed{N_\mathrm{s} = 16} \ .$$

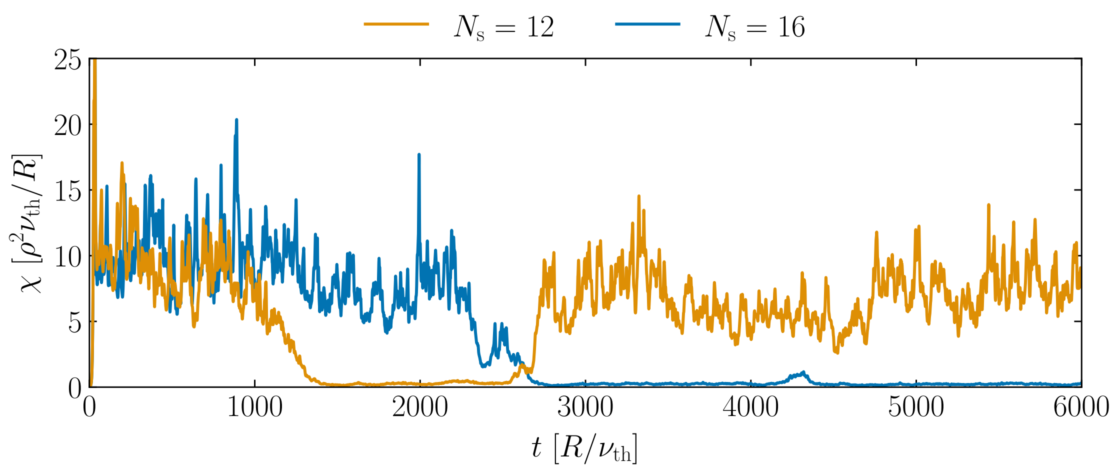


Figure 4.6: Time traces for reduced magnetic field grid points N_s.

4.1.5 Final Resolution for Simulation

Together with the results of Chapter 4.1.2, 4.1.3 and 4.1.4 the final resolution used in the upcoming simulations is displayed in Table 4.1, with reduced number of grid points for the parallel velocity $N_{v_\parallel}$.

	N_m	N_x	N_s	$N_{v_\parallel}$	N_μ	D	v_d	$D_{v_\parallel}$	D_x	D_y	Order	$k_y\rho$	$k_x\rho$
S6	21	83	16	48	9	1	$\lvert v_\parallel \rvert$	0.2	0.1	0.1	6	1.4	2.1

Table 4.1: Resolution used in this work: Number of toroidal modes N_m, number of radial modes N_x, number of grid points along the magnetic field N_s,number of parallel velocity grid points $N_{v_\parallel}$, number of magnetic moment grid points N_μ, dissipation coefficient used in convection along the magnetic field D,the velocity in the dissipation scheme v_d, dissipation coefficient used in the trapping term $D_{v_\parallel}$, damping coefficient of radial modes D_x, damping coefficient of toroidal modes D_y, order of the scheme used for the zonal mode, maximum poloidal wave vector $k_y\rho$, and maximum radial wave vector $k_x\rho$.

4.2 Size Convergence of $\mathbf{E} \times \mathbf{B}$ Staircase Pattern

In the following the box size is increased relative to the standard box size $(L_x,\ L_y) = (76.3,\ 89.8)\,\rho$ in the radial and binormal direction. The increased box sizes are indicated by the real parameter N_{R} for radial and N_{B} for the binormal direction with the nomenclature $N_{\mathrm{R}} \times N_{\mathrm{B}}$ throughout this work. Note that, the number of modes in the respective direction, i.e., N_x and N_m, respectively, is always adapted accordingly to retain a spatial resolution compliant to the standard resolution [Table 4.1] and standard box size.

4.2.1 Radial increased Box Size

In the first test the radial box size is increased while the binormal box size is kept fixed to the standard size. The scan covers the realizations

$$N_{\mathrm{R}} \times N_{\mathrm{B}} \in [1 \times 1,\ 2 \times 1,\ 3 \times 1,\ 4 \times 1]\ .$$

Each realization exhibits an initial quasi-stationary turbulent phase and a second final[43] phase with almost suppressed turbulence [Fig. 4.7(a)]. The latter state is indicative for the presence of a fully developed staircase pattern as depicted in Fig. 4.15. Figure 4.8 or Figure 4.15(a) shows a striking repetition of the staircase structure, with the number of repetitions equal to N_{R}. Hence, the basic size of the pattern not only converges with increasing radial box size, but the converged radial size also turns out to at least roughly agree with the standard radial box size of Ref. 43.

Due to the lack of a substantial turbulent drive in the final suppressed state no further zonal flow evolution is observed [Fig. 4.7(b)] and one might critically ask whether the structures shown in Fig. 4.15 represent the real converged pattern in a statistical sense. Note that in the 3×1 case the initial quasi-stationary turbulent state extends up to a few $\sim 10^4\,R/v_{\mathrm{th}}$. During this period the zonal flow mode with $n_{\mathrm{ZF}} = 3$, i.e., the mode that dominates the staircase pattern in final suppressed phase, undergoes a long-term evolution with a typical timescale of several $\sim 10^3\,R/v_{\mathrm{th}}$.
Hence, several of such cycles are covered by the initial turbulent phase, which is evident from the occurrence of phases with reduced amplitude around $t \approx 8000\,R/v_{\mathrm{th}}$ and $t \approx 18000\,R/v_{\mathrm{th}}$. It is the $n_{\mathrm{ZF}} = 4$ zonal flow mode, i.e., the next shorter radial scale mode, that dominates the shear spectrum $|\widehat{\omega}_{\mathrm{E\times B}}|_{n_{\mathrm{ZF}}}$ in the latter two phases (not shown). This demonstrates a competition between the $n_{\mathrm{ZF}} = 3$ and $n_{\mathrm{ZF}} = 4$ modes. Most importantly, no secular growth of the $n_{\mathrm{ZF}} = 1$ (box scale) zonal flow mode is observed during the entire quasi-stationary turbulent phase [Fig. 4.7(b) dotted line]. The above discussion indicates that although the $n_{\mathrm{ZF}} = 3,\ 4$ zonal modes compete, the pattern scale does not converge to the radial box scale but rather to a mesoscale of $\sim 57 - 76\,\rho$ (i.e., $n_{\mathrm{ZF}} = 4,\ 3$ in the 3×1 case).

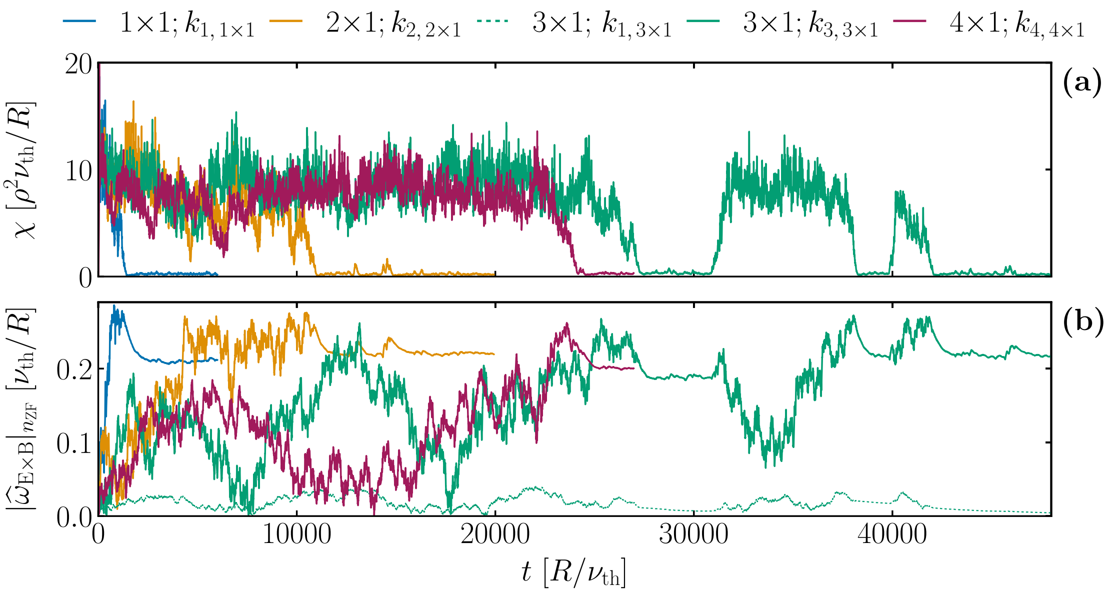


Figure 4.7: **(a)** Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for radial increased box sizes,
(b) Time traces of $|\widehat{\omega}_{\mathrm{E\times B}}|_{n_{\mathrm{ZF}}}$ for radial increased box sizes.

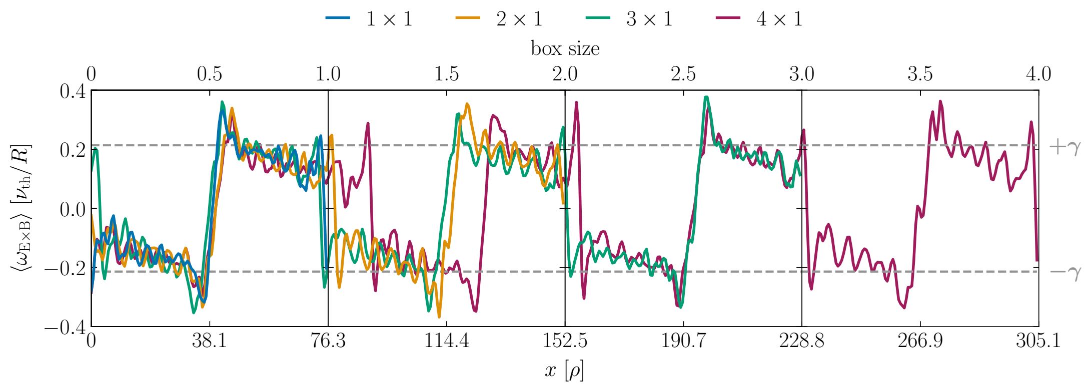


Figure 4.8: Comparison of shearing rate $\omega_{\mathrm{E\times B}}$ for the radial box size scan averaged over given a time interval and the growth rate $\pm\gamma$ of the most unstable linear ITG-driven Eigenmode. The staircase structures are radially shifted with respect to each over till alignment for better visibility:
$t_{1\times1} \in [2000, 5000]$, $t_{2\times1} \in [15000, 18000]$,
$t_{3\times1} \in [43000, 45000]$, $t_{4\times1} \in [26000, 28000]$.

4.2.2 Isotropic increased Box Size

Since the radially elongated simulation domain might inhibit the development of isotropic turbulent structures, in the second test the radial and binormal box size is increased simultaneously. This scan covers the realizations

$$N_{\mathrm{R}} \times N_{\mathrm{B}} \in [1 \times 1,\ 1.5 \times 1.5,\ 2 \times 2,\ 2.5 \times 2.5,\ 3 \times 3]\ .$$

Interestingly, suppression of the turbulence by the emergence of a fully developed staircase pattern almost always occurs after $\sim 1000\ R/v_{\mathrm{th}}$ [Fig. 4.9(a)], i.e., significantly faster compared to the 3×1 and 4×1 realizations. As shown in Fig. 4.10 or Fig. 4.15(b) this test also confirms the convergence of the staircase pattern size to a typical mesoscale that is distinct from the radial box size in the $N_{\mathrm{R}} > 1$ realizations.

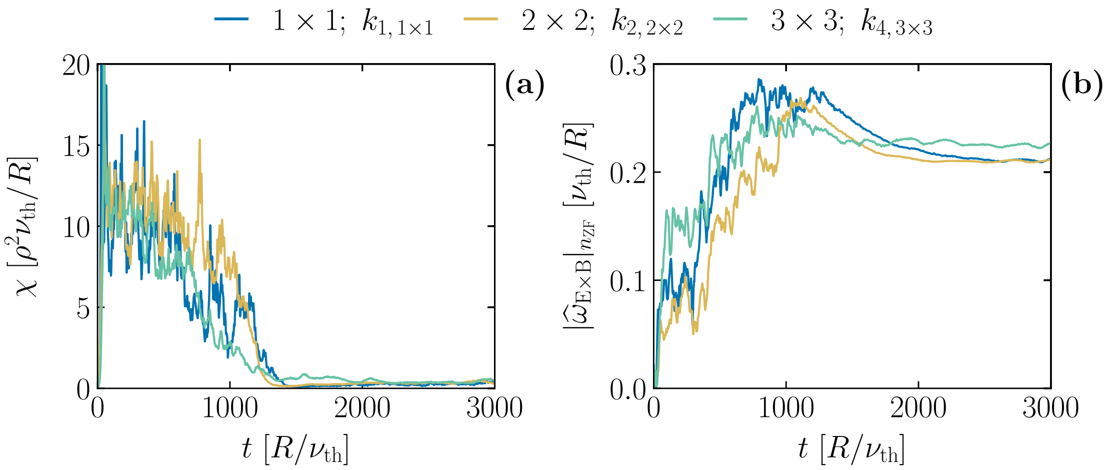


Figure 4.9: **(a)** Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for isotropic increased box sizes,
(b) Time traces of $|\widehat{\omega}_{\mathrm{E\times B}}|_{n_{\mathrm{ZF}}}$ for isotropic increased box sizes.

By contrast to the radial box size scan the 3×3 realization shows a stationary pattern with four repetitions of the fully developed staircase structure, i.e., a somewhat smaller pattern size [Fig. 4.9(b), Fig. 4.10 and Fig. 4.15(b)]. Whether this is related to a possible pattern size dependence on the binormal box size or to the competition between patterns with the two sizes $\lambda \in [57,\ 76]\,\rho$ as observed in the first test is addressed in the next paragraph. The scale of structures developing in the 1.5×1.5 and 2.5×2.5 realizations (not included in Fig. 4.15, Fig. 4.10 and Fig. 4.9 to preserve the clarity of this figure) also lie within the range given above.

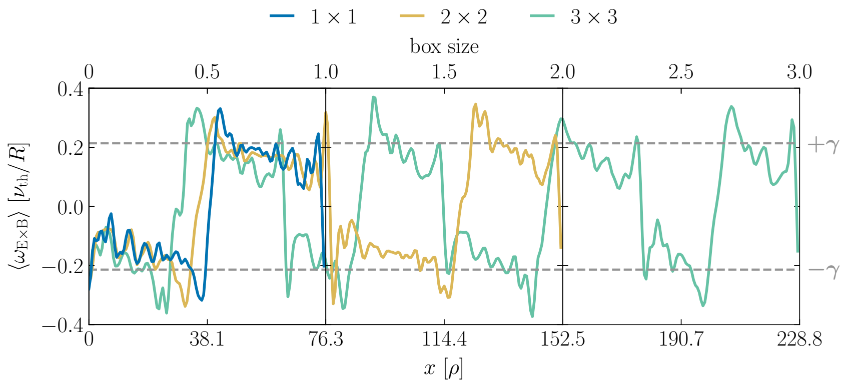


Figure 4.10: Comparison of shearing rate $\omega_{\mathrm{E\times B}}$ for the isotropic box size scan (1.5×1.5 not 2.5×2.5 included) averaged over given a time interval and the growth rate $\pm\gamma$ of the most unstable linear ITG-driven Eigenmode. The staircase structures are radially shifted with respect to each over till alignment for better visibility:

$$t_{1\times1} \quad \in [2000, 5000], \quad t_{2\times2} \quad \in [2000, 3000], \quad t_{3\times3} \quad \in [2000, 3000].$$

Note that two additional realizations of the 3×3 case with different initial conditions and otherwise identical parameters confirm structure formation on scales within the range given above [Fig. 4.11].

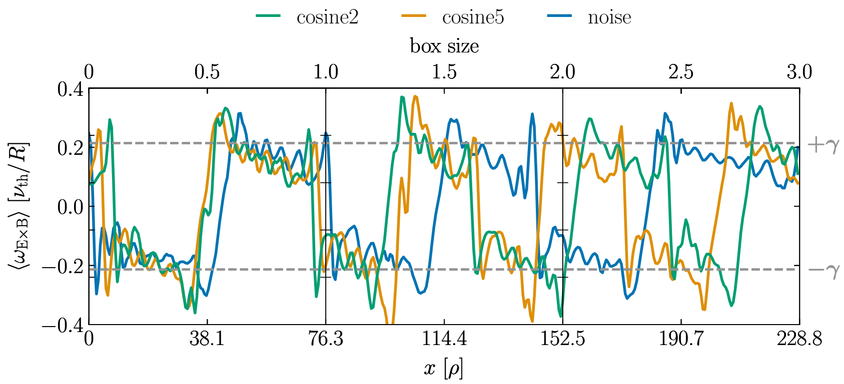


Figure 4.11: Comparison of shearing rate $\omega_{\mathrm{E\times B}}$ for the 3×3 box size with different initial conditions averaged over given a time interval and the growth rate $\pm\gamma$ of the most unstable linear ITG-driven Eigenmode. The staircase structures are radially shifted with respect to each over till alignment for better visibility:

$$t_{\mathrm{cosine2}} \quad \in [2000, 3000], \quad t_{\mathrm{cosine5}} \quad \in [2000, 3000], \quad t_{\mathrm{noise}} \quad \in [4000, 5000].$$

The negative gradient of the perturbed perpendicular and parallel ion pressure [Fig. 4.12] exhibit positive corrugations in regions with maximum $|\omega_{\mathrm{E\times B}}|$ and negative corrugations at zero crossings of $\omega_{\mathrm{E\times B}}$. A radial force balance analysis suggests that the structures in $\omega_{\mathrm{E\times B}}$ as depicted in Fig. 4.15 are not a consequence of the pressure gradient corrugations as discussed elsewhere[30]. Rather, the corrugations in the pressure gradient have to be interpreted as a consequence of the staircase structure in $\omega_{\mathrm{E\times B}}$ due to the stabilizing zonal flow - turbulence interaction.

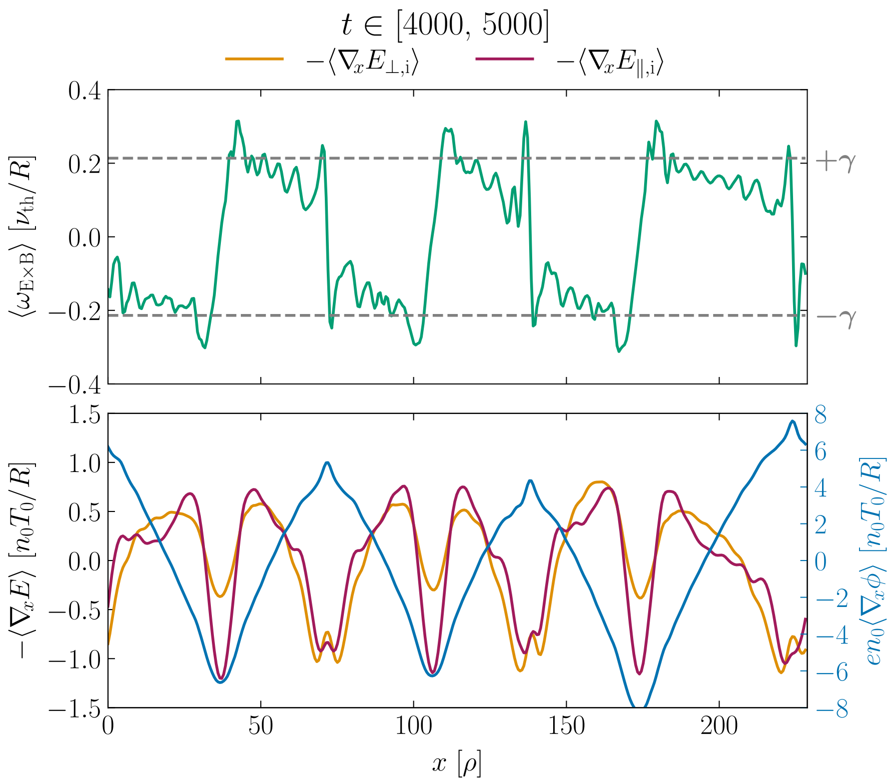


Figure 4.12: Time averaged radial profiles of (top) the $\mathrm{E\times B}$ shearing rate $\omega_{\mathrm{E\times B}}$ and (bottom) the negative gradient of the perpendicular and parallel energy moment $-\langle \nabla_x E_{\perp,\mathrm{i}} \rangle$ and $-\langle \nabla_x E_{\parallel,\mathrm{i}} \rangle$, respectively, as well as the electric field term $e n_0 \langle \nabla_x \phi \rangle$ of the force balance (on the right axis). Shown is a 3×3 realization with $R/L_T = 6.0$. Horizontal black and gray bars in the bottom panel indicate the modulation of the negative gradient of the perpendicular energy on the staircase scale.

4.2.3 Binormal increased Box Size

In a third test the binormal box size is varied with the radial box size fixed to $N_\mathrm{R} = 3$. This test covers the realizations

$$N_\mathrm{R} \times N_\mathrm{B} \in [3 \times 1.5,\ 3 \times 2.5,\ 3 \times 3,\ 3 \times 5]\ .$$

As in the isotropic scan the turbulence subdued and a fully developed staircase pattern forms after $\sim 2000\, R/v_\mathrm{th}$ [Fig. 4.13(a)]. The convergence of staircase pattern can be seen in Fig. 4.14 or Fig. 4.15(c) and confirms again a size of a typical mesoscale. Fig. 4.15(c) also confirms that indeed a competition between patterns with two sizes $\lambda \in [57,\ 76]\,\rho$ causing the different results for 3×1 and 3×3. The zonal flow mode number varies between $n_\mathrm{ZF} = 3, 4$ which can be seen in Fig. 4.15(c) in the 3×2.5 realization. The staircase structure has a pattern between three and four repetitions which get represented in the second repetition with no significant plateau at positive shear. Instead the pattern returns immediately after reaching the maximum shear $(+\gamma)$ to the minimum shear $(-\gamma)$ of the third repetition in a steep flank. The Fourier analysis of this case yields no definitely basic mode rather two dominating modes with $n_\mathrm{ZF} = 3, 4$ with a fraction of the maximum amplitude $|\widehat{\omega}_\mathrm{E\times B}|_{n_\mathrm{ZF}}$ each [Fig. 4.13(b)].

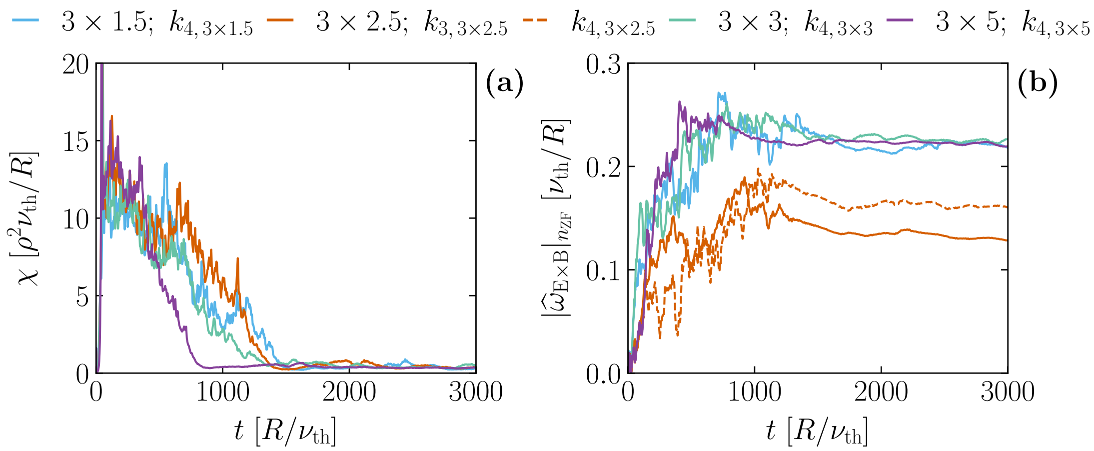


Figure 4.13: **(a)** Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for binormal increased box sizes,
(b) Time traces of $|\widehat{\omega}_\mathrm{E\times B}|_{n_\mathrm{ZF}}$ for binormal increased box sizes.

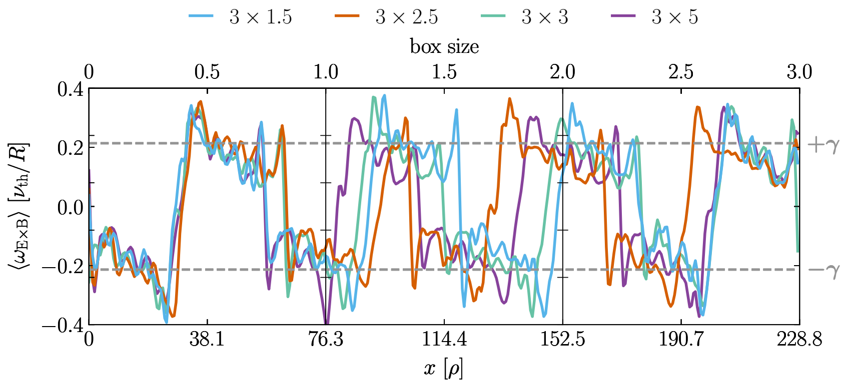


Figure 4.14: Comparison of shearing rate $\omega_{\mathrm{E\times B}}$ for the binormal box size scan averaged over given a time interval and the growth rate $\pm\gamma$ of the most unstable linear ITG-driven Eigenmode. The staircase structures are radially shifted with respect to each over till alignment for better visibility:

$t_{3\times1.5} \in [2000, 3000]$, $t_{3\times2.5} \in [2000, 3000]$,
$t_{3\times3} \in [2000, 3000]$, $t_{3\times5} \in [1000, 3000]$.

4.2.4 Staircase Structures in Comparison

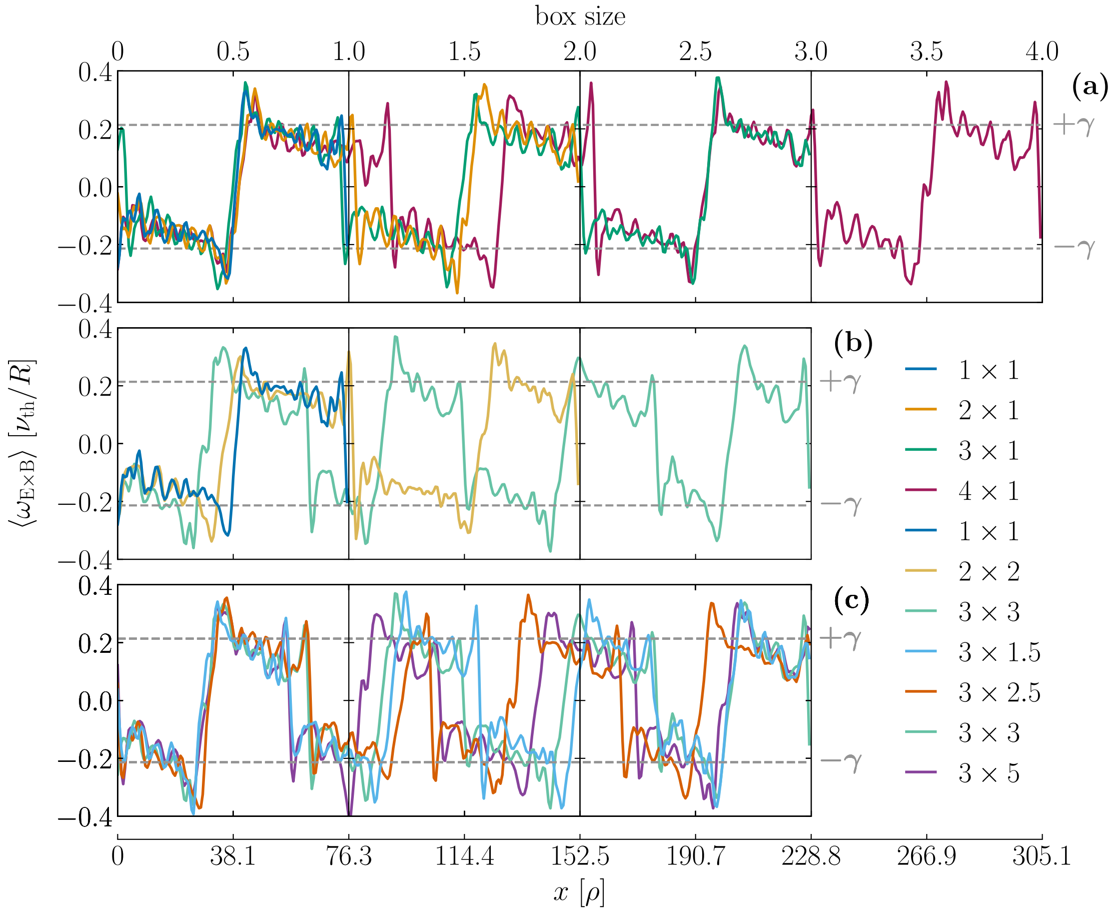


Figure 4.15: Comparison of shearing rate $\omega_{\mathrm{E}\times\mathrm{B}}$ for each box size scan averaged over given a time interval and the growth rate $\pm\gamma$ of the most unstable linear ITG-driven Eigenmode. The staircase structures are radially shifted with respect to each over till alignment for better visibility:

(a) radial: $t_{1\times1} \in [2000, 5000]$, $t_{2\times1} \in [15000, 18000]$, $t_{3\times1} \in [43000, 45000]$, $t_{4\times1} \in [26000, 28000]$;

(b) isotropic: $t_{1\times1} \in [2000, 5000]$, $t_{2\times2} \in [2000, 3000]$, $t_{3\times3} \in [2000, 3000]$;

(c) binormal: $t_{3\times1.5} \in [2000, 3000]$, $t_{3\times2.5} \in [2000, 3000]$, $t_{3\times3} \in [2000, 3000]$, $t_{3\times5} \in [1000, 3000]$.

4.3 The finite Heat Flux Threshold

In the final test the inverse background temperature gradient length R/L_T is varied at fixed 3×3 box size. Since suppression of turbulence usually occurs at later times when approaching the finite heat flux threshold from below[43], the analysis aims to lengthen the phase during which the zonal flow varies in time due to turbulent Reynolds stresses. This scan covers realizations with

$$R/L_T \in [6.0,\ 6.2,\ 6.4]\ .$$

In the case of $R/L_T = 6.2$ turbulence suppression is observed for $t > 11000\,R/v_{\mathrm{th}}$, while stationary turbulence during the entire simulation time trace of $12000\,R/v_{\mathrm{th}}$ is found for $R/L_T = 6.4$ [Fig. 4.16]. The finite heat flux threshold, hence, is

$$\boxed{R/L_T|_{\mathrm{finite}} = 6.3 \pm 0.1}$$

in accordance with Ref. 43. Although the initial quasi-stationary turbulence in the former case is significantly longer compared to the $R/L_T = 6.2$ realization discussed in the second test, a stationary pattern with basic zonal flow mode $n_{\mathrm{ZF}} = 3$ establishes. Again, the $n_{\mathrm{ZF}} = 1$ (box scale) zonal flow mode does not grow secularly during the entire turbulent phase. Also, this test confirms the statistical soundness of the converged pattern size of $\sim 57 - 76\,\rho$.

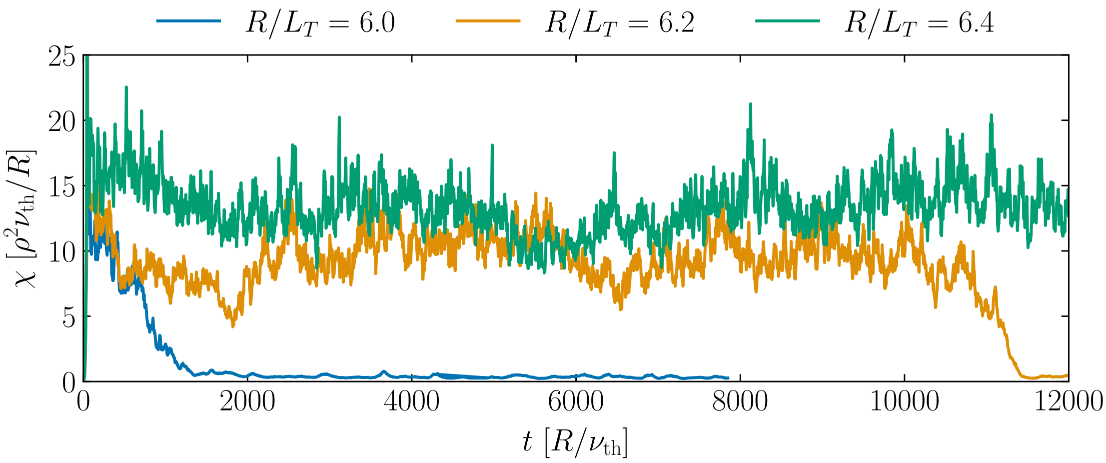


Figure 4.16: Time traces of the heat conduction coefficient χ for different gradient lengths R/L_T

CHAPTER 5

Conclusion

In this thesis the minimal resolution for simulations with GKW in the Cyclone Base parameter were observed in which the number of grid points for the parallel velocity $N_{\nu_\parallel}$ could be reduced from 64 to 48, which halfed the time until suppression of turbulence.

Additionally, the active development of a restart script in `python3` led to further convenience during the task of performing simulations on the `btrzx1` cluster.

Through careful tests this bachelor thesis confirms the radial size convergence of the $\mathrm{E} \times \mathrm{B}$ staircase pattern in local gyrokinetic flux tube simulations of ion temperature gradient (ITG)-driven turbulence. A mesoscale pattern size of $\sim 57 - 76\,\rho$ is found to be intrinsic to ITG-driven turbulence for Cyclone Base Case parameters. This length scale is somewhat larger compared to results from global studies with finite ρ_*, which report of a few $10\,\rho$[16], and has to be considered the proper mesoscale in the local limit $\rho_* \to 0$. The occurrence of this mesoscale implies that non-locality, in terms of Ref. 16, is inherent to ITG-driven turbulence, since avalanches are spatially organized by the $\mathrm{E} \times \mathrm{B}$ staircase pattern[38,16,45,43].

CHAPTER 6

Appendix

6.1 slurm_monitor.py

```
#!/usr/bin/env python3

# AUTHOR ================================================================
    ↪ ===============================================
# Manuel Lippert (GitHub: ManeLippert (https://github.com/ManeLippert))
# ======================================================================
    ↪ ===============================================

# MODULES ===============================================================
    ↪ ===============================================

import datetime, time, os, sys, subprocess, math, argparse, pkg_resources

# PARSER ================================================================
    ↪ ===============================================

description_text = """

===================================== DESCRIPTION ===========================
    ↪ ==========

FEATURES:
o NO REQUIREMENTS: Default script runs with standard python3 library
o Creates jobscript file with defined content
  (look into variable "jobscriptContent" to add more option)
o Start/Restarts job until criteria is suffused (default=10000)
o Makes backup after each run before Restart and Restore files after failed
    ↪ run
  (uses rsync command line utility)
o Reset option after run fails and dump files were written
  (rely on "h5py" & "pandas" & "numpy" which get installed by the script)
o Creates status file with current status and progress bars and updates it
    ↪ dynamically
  Progress: Total progress of simulation
  Run X    : Progress of current run
o Sends mail at the beginning, end and restart (default=False) with status
    ↪ file
  as attachment
  (mailx has to be installed, look into "send_mail" function for more info)

START SCRIPT IN BACKGROUND:

  o WITH NOHUP:
    Command:
    >>> nohup python3 -u slurm_monitor.py --job-name $JOBNAME &> /dev/null &

    Kill Process:
    >>> python3 -u slurm_monitor.py --job-name $JOBNAME --kill
```

```
  o WITH SCREEN (has to be installed):
    Create Screen:
    >>> screen -S $SESSION

    Command:
    >>> python3 -u slurm_monitor.py --job-name $JOBNAME --verbose

    Leave Screen:
    >>> ((Strg + a) + d)

    Enter Screen:
    >>> screen -r
    or
    >>> screen -r $SESSION

    Kill Process:
    >>> ^C (Strg + c)
    or
    >>> (Strg + d) (kill Screen itself)

OUTPUT STATUS:
Just run the file will create an dynamic output (recommended with using
    ↪ screen) or
>>> cd $DATA && find . -name $STATUSFILENAME -exec tail -8 {} \;

====================================== ARGUMENTS ============================
    ↪ ==========
"""

parser = argparse.ArgumentParser(description=description_text,
    ↪ formatter_class=argparse.RawTextHelpFormatter)
#parser._action_groups.pop()

required = parser.add_argument_group("required arguments")

required.add_argument("-j", "--job-name", dest="jobname", nargs="?",
    ↪ type=str, required= True,
                      help="job name not longer than 8 character")

additional = parser.add_argument_group("additional arguments")

additional.add_argument("-n", dest="timesteps", nargs="?", type=int,
    ↪ default=10000,
                        help="required timesteps
    ↪ (default=10000)")

additional.add_argument("-r", "--reset", dest="reset", action="store_true",
                        help="Uses Dumpfiles to reset Simulation
    ↪ (default=False)")

additional.add_argument("-v", "--verbose", dest="verbose",
    ↪ action="store_true",
                        help="activate output of script
```

```
    ↪ (default=False)")

additional.add_argument("-b", "--backup", dest="backup", nargs="?", type=str,
                        help="backup location for files
    ↪ (default=None)\n"+
                             "- local (creates backup in simulation
    ↪ folder)\n"+
                             "- home  (creates backup in home folder)")

additional.add_argument("-d", "--dir", dest="directory", nargs="?",
    ↪ type=str, default=os.getcwd(),
                        help="directory of simulation
    ↪ (default=cwd)\n")

additional.add_argument("--jobscript", dest="jobscriptFile", nargs="?",
    ↪ type=str, default="jobscript-create",
                        help="jobscript to run SLURM job
    ↪ (default=jobscript-create)")

additional.add_argument("--ntask-per-node", dest="tasks", nargs="?",
    ↪ type=str, default="32",
                        help="MPI task per node
    ↪ (default=32)")

additional.add_argument("--nodes", dest="nodes", nargs="?", type=str,
    ↪ default="3",
                        help="number of nodes
    ↪ (default=3)")

additional.add_argument("--walltime", dest="wallTime", nargs="?", type=str,
    ↪ default="0-24:00:00",
                        help="walltime of server (d-hh:mm:ss)
    ↪ (default=0-24:00:00)")

additional.add_argument("--mail", dest="mail", nargs="?", type=str,
                        help="mail address (mail@server.de)
    ↪ (default=None)")

additional.add_argument("--restart-mail", dest="restartmail",
    ↪ action="store_true",
                        help="mail after every restart
    ↪ (default=False)")

additional.add_argument("--statusfile", dest="statusFile", nargs="?",
    ↪ type=str, default="status.txt",
                        help="file with output from nohup command
    ↪ (default=status.txt)")

additional.add_argument("--restartfile", dest="restartFile", nargs="?",
    ↪ type=str, default="FDS.dat",
                        help="restart file with data
    ↪ (default=FDS.dat)")
```

```
additional.add_argument("--format", dest="formattable", nargs="?", type=str,
                        help="format of output table
    ↪ (default=None)\n"+
                             "- fancy (round box)\n"+
                             "- universal (crossplattform)\n"+
                             "- None (No frame)")

additional.add_argument("--refresh-rate", dest="sleepTime", nargs="?",
    ↪ type=int, default=300,
                        help="time interval to check status in sec
    ↪ (default=300)")

additional.add_argument("--kill", dest="kill", action="store_true",
                        help="kills monitor process
    ↪ (default=False)")

args = parser.parse_args()

# PARSER VARIABLES ============================================================
    ↪ ===============================================

emailAddress = args.mail
RESTARTMAIL = args.restartmail

backupLocation = args.backup
folder = args.directory

jobName = args.jobname
tasks = args.tasks
nodes = args.nodes
wallTime = args.wallTime

nTimestepsRequired = args.timesteps

jobscriptFilename = args.jobscriptFile
restartFilename = args.restartFile
restartBin = restartFilename.replace(".dat", "")

statusFilename = args.statusFile

formatTable = args.formattable
VERBOSE = args.verbose
RESET = args.reset

# Kill process of monitoring
kill = args.kill

# Changing this value can cause problems in writing status file
sleepTime = args.sleepTime

# VARIABLES ===================================================================
    ↪ ===============================================
```

```
outputCriteria = ["0", "Run successfully completed"]

slurmFiles = [f for f in os.listdir() if "slurm-" in f]
runCounter = len(slurmFiles)

nTimestepsCurrent = 0

currentTime = "00:00:00"

dataFilename = "gkwdata.h5"

check1Filename, check2Filename = "DM1.dat", "DM2.dat"
check1Bin, check2Bin = check1Filename.replace(".dat", ""),
    ↪ check2Filename.replace(".dat", "")

def outputFilename(info):
    return "slurm-" + info + ".out"

## KILL JOB =================================================================
    ↪ ===============================================

commandMonitorKill = "ps ax | grep " + jobName + " | grep -v grep | grep -v
    ↪ kill | awk '{print $1}'"

PID = subprocess.getoutput(commandMonitorKill)
if kill:
    subprocess.run(["kill", PID])
    quit()

## JOBSCRIPT CONTENT ========================================================
    ↪ ===============================================

jobscriptContent = """#!/bin/bash -l

# jobname
#SBATCH --job-name=""" + jobName + """

# MPI tasks
#SBATCH --ntasks-per-node=""" + tasks + """

# number of nodes
#SBATCH --nodes=""" + nodes + """

# walltime
#               d-hh:mm:ss
#SBATCH --time=""" + wallTime + """

# execute the job
time mpirun -np $SLURM_NTASKS """ + folder + "/" + """gkw.x > output.dat

# end
exit 0
"""
```

```
jobName = jobName[0:8]

## FLAGS ===================================================================
    ↪ ===============================================

startOutputFlag = "Submitted batch job "
restartFlag = "FILE_COUNT"

# SWITCHES =================================================================
    ↪ ===============================================

if emailAddress==None:
    EMAIL = False
else:
    EMAIL = True

if backupLocation==None:
    BACKUP = False
else:
    BACKUP = True

## PATHS ===================================================================
    ↪ ===============================================

user = os.getlogin()

if folder[-1] == "/":
    folder = folder[:-1]

path = folder.split(user + "/")[1]

# Set backup location
if BACKUP:
    # If local has been specified as backupLocation, then the backup is
    ↪ copied to the same directory as the simulation folder
    # (simFolder) is located in. The backup is copied to a directory with
    ↪ name simFolder + "-backup".
    if backupLocation == "local":
        simFolder = path.split("/")[-1]
        backupPath = folder + "/../" + simFolder + "-backup"

    # Create backup in home folder
    elif backupLocation == "home":
        simFolder = path.split("/")[-1]
        backupPath = "~/" + simFolder + "-backup"

    # Otherwise, use the backupLocation parsed as argument.
    else:
        if backupLocation[-1] != "/":
            backupLocation += "/"
        backupPath = backupLocation + path
```

```
    if not os.path.exists(backupPath):
        os.makedirs(backupPath)

## COMMANDS =================================================================
    ↪ ===============================================

commandJobRunning = "squeue --states=running --name " + jobName
commandJobPending = "squeue --states=pending --name " + jobName

commandJobStarting = "sbatch"

if BACKUP:
    commandBackup  = ["rsync", "-a", folder + "/", backupPath]
    commandRestore = ["rsync", "-a", "-I", "--exclude=" + statusFilename,
    ↪ backupPath + "/", folder]

# FUNCTIONS =================================================================
    ↪ ===============================================

## PROGRESSBAR ==============================================================
    ↪ ===============================================

def progressbar(required_value, current_value, progressbar_width,
                prefix="", prefix_width = 10, percent_width = 8, ratio_width
    ↪ = 13,
                progress_fill="=", progress_fill_top="",
    ↪ progress_fill_bot="",
                progress_unfill=".",
                progress_bracket=["[","]"]):


    prefix_format = "{:<" + str(prefix_width) + "}"

    percentage = int(100*current_value/required_value)
    percent = "(" + " "*(3-len(str(percentage))) + str(percentage) + "%)"
    percent_format = "{:>" + str(percent_width) + "}"

    ratio = str(current_value) + "/" + str(required_value)
    ratio_format = "{:>" + str(ratio_width) + "}"

    progressbar_bracket_width = (len(progress_bracket[0]) +
    ↪ len(progress_bracket[1])
                                + len(progress_fill_top) +
    ↪ len(progress_fill_bot))

    barsize = progressbar_width - prefix_width - percent_width - ratio_width
    ↪ - progressbar_bracket_width

    if percentage <= 100:
        x = int(barsize*current_value/required_value)
    else:
        x = barsize
```

```
    progress_format = progress_bracket[0] + progress_fill_bot + "{}" +
    ↪ progress_fill_top + "{}" + progress_bracket[1]

    bar_format =  prefix_format +progress_format + percent_format +
    ↪ ratio_format
    bar = bar_format.format(prefix, progress_fill*x,
    ↪ progress_unfill*(barsize-x), percent, ratio)

    return bar

## OUTPUT TABLE ==============================================================
    ↪ ===============================================

def message(string, add):
    string = string + add + "\n"
    return string

def print_table_row(content,
                    delete_line_index = -10, table_width = 80,
                    output_type = None,
                    TIMEINFO = True, WRITEFILE=True):

    current_value, required_value = nTimestepsCurrent, nTimestepsRequired,
    run_counter = runCounter
    current_time, required_time = currentTime, wallTime

    msg=""
    table_inner_width = table_width - 4
    table_content_width = table_inner_width - 47

    if formatTable == "fancy":
        table_outline = ["╭─", "─╮", "╰─", "─╯", "├─", "─┤", "│ ", " │", "─"]

    if formatTable == "universal":
        table_outline = ["+-", "-+", "+-", "-+", "+-", "-+", "| ", " |", "-"]

    if formatTable == None:
        table_outline = ["--", "--", "--", "--", "--", "--", "  ", "  ", "-"]

    sep_top = table_outline[0] + table_inner_width*table_outline[8] +
    ↪ table_outline[1]
    sep_mid = table_outline[4] + table_inner_width*table_outline[8] +
    ↪ table_outline[5]
    sep_end = table_outline[2] + table_inner_width*table_outline[8] +
    ↪ table_outline[3]

    row_cols   = [2, 10, table_content_width, 13, 11, 13, 2]
    row_format = "".join(["{:<" + str(col) + "}" for col in row_cols])

    progress_cols   = [2, table_inner_width, 2]
    progress_format = "".join(["{:<" + str(col) + "}" for col in
    ↪ progress_cols])
```

```
progressbar_content = [progressbar(required_value, current_value,
↪ table_inner_width,
                                   progress_fill_top=">",
↪ prefix="PROGRESS")]

progressbar_content.insert(0, table_outline[6])
progressbar_content.insert(len(progressbar_content), table_outline[7])

required_time = get_time_in_seconds(required_time)
current_time  = get_time_in_seconds(current_time)

jobStatus = subprocess.getoutput("squeue --name " +
↪ jobName).strip().split("\n")

try:
    jobStatusHeader = ["  " + jobStatus[0]]
    jobStatusInfo   = [jobStatus[1][11:11+table_inner_width]]
except IndexError:
    jobStatus_cols   = [12, 12, 10, table_inner_width - 71, 13, 11, 13]
    jobStatus_format = "".join(["{:<" + str(col) + "}" for col in
↪ jobStatus_cols])

    jobStatusHeader = ["OUTPUT", "NAME", "USER", "" ,"DATE", "TIME",
↪ "W:DD:HH:MM:SS"]
    jobStatusInfo   = [content[0], jobName, user, "", time_date(),
↪ time_time(), time_duration(startTime, pastTime)]

    jobStatusHeader   = [jobStatus_format.format(*jobStatusHeader)]
    jobStatusInfo     = [jobStatus_format.format(*jobStatusInfo)]

jobStatusHeader.insert(0, table_outline[6])
jobStatusHeader.insert(len(jobStatusHeader), table_outline[7])
jobStatusInfo.insert(0, table_outline[6])
jobStatusInfo.insert(len(jobStatusInfo), table_outline[7])

progressbartime_content = [progressbar(required_time, current_time,
↪ table_inner_width,
                                       progress_fill_top=">",
↪ prefix="RUN " + str(run_counter))]

progressbartime_content.insert(0, table_outline[6])
progressbartime_content.insert(len(progressbartime_content),
↪ table_outline[7])

content[1] = content[1][-(table_content_width-1):]
content.insert(0, table_outline[6])
if TIMEINFO:
    if output_type == "header":
        content.append("DATE")
        content.append("TIME")
        content.append("W:DD:HH:MM:SS")
    else:
        content.append(time_date())
```

```
            content.append(time_time())
            content.append(time_duration(startTime, pastTime))

    content.insert(len(content), table_outline[7])

    if output_type == "header":
        delete_line_index = -1

        msg += sep_top + "\n"
        msg += row_format.format(*content) + "\n"
        msg += sep_end + "\n"

    elif output_type == "middle":
        if VERBOSE:
            sys.stdout.write("\x1b[1A"*(-delete_line_index + 1))

        msg += sep_mid + "\n"
        msg += row_format.format(*content) + "\n"
        msg += sep_end + "\n"

    elif output_type == "update":
        if VERBOSE:
            sys.stdout.write("\x1b[1A"*(-delete_line_index + 1))

        msg += sep_end + "\n"

    else:
        if VERBOSE:
            sys.stdout.write("\x1b[1A"*(-delete_line_index + 1))

        msg += row_format.format(*content) + "\n"
        msg += sep_end + "\n"

    msg += " "*table_width + "\n" + " "*table_width + "\n"

    msg += sep_top + "\n"
    msg += progress_format.format(*jobStatusHeader) + "\n"
    msg += progress_format.format(*jobStatusInfo) + "\n"
    msg += sep_mid + "\n"
    msg += progress_format.format(*progressbar_content) + "\n"
    msg += progress_format.format(*progressbartime_content) + "\n"
    msg += sep_end + "\n"

    if VERBOSE:
        print(msg, flush=True)
    if WRITEFILE:
        delete_write_line_to_file(statusFilename, msg, end=delete_line_index)

## PIP INSTALL ===============================================================
    ↪ ===============================================

def pip_install(modules):
    required  = modules
```

```
    installed = {pkg.key for pkg in pkg_resources.working_set}
    missing   = required - installed

    if missing:
        subprocess.check_call([sys.executable, "-m", "pip", "install",
    ↪ "--user", *missing],
                                stdout=subprocess.DEVNULL,
    ↪ stderr=subprocess.DEVNULL)

        print_table_row(["INSTALL", "Required Modules installed"])
    else:
        print_table_row(["CHECK", "Modules already installed"])

## FILE =============================================================
    ↪ =============================================

def get_value_of_variable_from_file(filename, file_index, relative_index,
    ↪ string):

    try:
        content = [i.strip().split() for i in open(folder + "/" +
    ↪ filename).readlines()]
        index = [idx for idx, s in enumerate(content) if string in
    ↪ s][file_index]
        value = content[index][relative_index]
        return value
    except IndexError:
        print_table_row(["ERROR", "String not found in file"])
        quit()

def find_string_in_file(filename, string):

    with open(folder + "/" + filename) as f:
        if string in f.read():
            return True
        else:
            return False

def delete_write_line_to_file(filename, add = "", start=None, end=None):

    with open(folder + "/" + filename, "r") as file:

        try:
            lines = file.readlines()[start:end]
            content = "".join(lines)

        except IndexError:
            pass

    content += add

    write_file(filename, content)
```

```
def write_file(filename, content):

    with open(folder + "/" + filename, "w") as file:
        file.write(content)
        file.flush()

# Reset gkwdata.h5 with the use of dump files DM1, DM2
# AUTHOR: Florian Rath
# IMPORT: gkw_reset_checkpiont.py
   ↪ (https://bitbucket.org/gkw/gkw/src/develop/python/gkw_reset_checkpoint.py)
def reset_simulation(SIM_DIR, NTIME=None, use_ntime=False):

    # Function that delets all data in interval [nt_reset:nt_broke].
    # ncol considers, if data series is ordered by a multiple interger of
    # ntime.
    def reset_time_trace(indata, dim, ncol, nt_reset):

        # Shift dimension dim to 0.
        data_shifted = np.moveaxis(indata, dim, 0)

        # Reset data.
        data_shifted_trimmed = data_shifted[0:int(nt_reset*ncol),]

        # Shift dimension back.
        out = np.moveaxis(data_shifted_trimmed, 0, dim)

        # free memory
        del data_shifted
        del data_shifted_trimmed

        return out

    # Checks if file has binary format.
    def is_binary(filename):

        try:
            with open(filename, "tr") as check_file:
                check_file.read()
                return False
        except:
            return True



    # Check for specific files that are no ordinary data files.
    def is_file_exception(filename):

        # substrings that have to be checked
        check_list = ["geom.dat", "DM1.dat", "DM2.dat", "FDS.dat", ".o",
   ↪ "FDS",
                      "input.dat", "perform_first.dat", "perform.dat",
   ↪ "output.dat",
                      "gkwdata.meta", "gkw_hdf5_errors.txt",
```

```
↪ "kx_connect.dat",
                  "jobscript", "Poincare1.mat", "perfloop_first.dat",
↪ "par.dat",
                  "input_init.dat", "sgrid", "gkw", ".out",
↪ statusFilename]
    for key in check_list:
        if key in filename:
            return True

    return False


# Check if given file is a PBS or SLURM jobscript.
def is_jobscript(filename):

    with open(filename,"r") as file:
        for line in file:
            if "#PBS -l" in line:
                return True
            if "#SBATCH" in line:
                return True

    return False

def get_timestep(FILE):
    if(os.path.isfile(SIM_DIR+"/"+FILE+".dat")):
        EXISTS = True
        with open(SIM_DIR+"/"+FILE+".dat","r") as file:
            for line in file:
                if "NT_REMAIN" in line:
                    expr = line.replace(" ","")
                    expr = expr.replace(",","")
                    expr = expr.replace("\n","")
                    REMAIN = int(expr.split("=")[1])
                if "NT_COMPLETE" in line:
                    expr = line.replace(" ","")
                    expr = expr.replace(",","")
                    expr = expr.replace("\n","")
                    COMPLETE = int(expr.split("=")[1])

    else:
        REMAIN, COMPLETE, EXISTS = None, None, False

    return REMAIN, COMPLETE, EXISTS

HDF5_FILENAME = "gkwdata.h5"
RESTARTFILE,DUMPFILE1, DUMPFILE2 = "FDS","DM1", "DM2"

# Change to simulation directory.
if(not os.path.isdir(SIM_DIR)):
    return
else:
    os.chdir(SIM_DIR)
```

```
# Check if hdf5-file exists.
if(not os.path.isfile(SIM_DIR + "/" + HDF5_FILENAME)):
    return

# First, read hdf5 file and determine the number of big time steps NTIME,
# requested in the input.dat file.
f = h5py.File(SIM_DIR + "/" + HDF5_FILENAME, "r+")


# Get requested big time steps from the /control group in the hdf5-file.
if(NTIME==None):
    NTIME = int(f["input/control/ntime"][:])

# Get number of big time steps after which simulation broke.
# If time.dat exists read this file to obtain number of time steps after
# which simulation broke.
if(os.path.isfile(SIM_DIR + "/" + "time.dat")):
  tim = pd.read_csv(SIM_DIR+"time.dat", header=None, sep="\s+").values
  NT_BROKE = tim.shape[0]
# Else, get time from hdf5-file.
else:
  NT_BROKE = f["diagnostic/diagnos_growth_freq/time"].shape[1]


# Set NT_BROKE for output files holding temporal derivates and therefore
# one timestep less.
NT_BROKE_DERIV = NT_BROKE-1

# Close the hdf5-file again.
f.close()

# Get the number of remaining big time steps NT_REMAIN from checkpoint
# files FDS.dat. This is used lateron to determine the most recent
# checkpoint file.
NT_REMAIN, NT_COMPLETE, FDS_EXISTS = get_timestep(RESTARTFILE)

# Get the number of remaining big time steps NT_REMAIN[1/2] from
↪ checkpoint
# files DM[1/2]. This is used lateron to determine the most recent
# checkpoint file.
NT_REMAIN1, NT_COMPLETE1, DM1_EXISTS = get_timestep(DUMPFILE1)
NT_REMAIN2, NT_COMPLETE2, DM2_EXISTS = get_timestep(DUMPFILE1)


# Check if FDS is the most recent checkpoint file. In this case
# resetting the simulation makes no sense.
if(FDS_EXISTS):
    DM1_OLD, DM2_OLD = False, False
    if(DM1_EXISTS):
        if(NT_COMPLETE1 < NT_COMPLETE):
            DM1_OLD = True
    if(DM2_EXISTS):
```

```
        if(NT_COMPLETE2 < NT_COMPLETE):
            DM2_OLD = True
    if(DM1_OLD and DM2_OLD):
        return


# Now determine which checkpoint file is the recent one.
if(DM1_EXISTS and DM2_EXISTS):
    if(NT_REMAIN1 > NT_REMAIN2):
        NT_REMAIN, NT_COMPLETE, DUMPFILE = NT_REMAIN2, NT_COMPLETE2,
↪ DUMPFILE2
    else:
        NT_REMAIN, NT_COMPLETE, DUMPFILE = NT_REMAIN1, NT_COMPLETE1,
↪ DUMPFILE1

elif(DM1_EXISTS and not DM2_EXISTS):
    NT_REMAIN, NT_COMPLETE, DUMPFILE = NT_REMAIN1, NT_COMPLETE1,
↪ DUMPFILE1

elif(DM2_EXISTS and not DM1_EXISTS):
    NT_REMAIN, NT_COMPLETE, DUMPFILE = NT_REMAIN2, NT_COMPLETE2,
↪ DUMPFILE2

else:
    return


if(not use_ntime):
    # Find the total number ob big time steps the simulation time trace
↪ should have,
    # when considering the big time steps already completed as well as
↪ the big time
    # steps that remain. Can be different from NTIME, since the
↪ simulation could
    # have been restarted several times such that NTIME > NT_COMPLETE.
    NTOT = NT_COMPLETE + NT_REMAIN
    N_REQUEST = NTOT

    # Determine the time steps to which the time trace files have to be
↪ reset.
    # Use NTOT here, since NTIME could have been changed at some point,
↪ or NT_COMPLETE
    # could be larger than NTIME.
    NT_RESET = NTOT-NT_REMAIN
else:
    # Determine the time steps to which the time trace files have to be
↪ reset.
    NT_RESET = NTIME-NT_REMAIN
    N_REQUEST = NTIME

# Same for files holding time derivatives.
NT_RESET_DERIV = NT_RESET-1
```

```
# -----------------------------------------------------------------
# Cycle over all nodes of hdf5-file and reset time trace datasets.

# Check if hdf5-file exists.
if(os.path.isfile(SIM_DIR + "/" + HDF5_FILENAME)):

    # Find all possible keys items, i.e. both groups and datasets
    f = h5py.File(SIM_DIR + "/" + HDF5_FILENAME, "a")
    h5_keys = []
    f.visit(h5_keys.append)

    # Cycle over all keys items and check, if any dimension has size
↪ NT_BROKE,
    # i.e., it is a time trace file.
    for item in enumerate(h5_keys):

        data = f.get(item)

        # Consider datasets only.
        if(isinstance(data, h5py.Dataset)):

            #Check if any dimension is an integer multiple of NT_BROKE,
↪ by checking
            # the residual of the division.
            res = [None]*len(data.shape)
            for i in range(len(data.shape)):
                res[i] =
↪ data.shape[i]/NT_BROKE-np.floor(data.shape[i]/NT_BROKE)

            if 0.0 in res:

                # Check which dimension is integer multiple of NT_BROKE
                # and save dimension as well as integer.
                new_shape = data.shape
                for i in range(len(data.shape)):
                    ncol = data.shape[i]/NT_BROKE
                    res = ncol - np.floor(ncol)
                    if(res == 0):
                        dim = i
                        ncol = int(ncol)
                        # adjust new shape to ncol*NT_RESET
                        y = list(new_shape)
                        y[dim] = int(ncol*NT_RESET)
                        new_shape = tuple(y)
                        break

                # Reset dataset (.resize discards data with indices
↪ larger than
                # ncol*NT_RESET along dimension dim).
                dset = f[item]
                dset.resize(int(ncol*NT_RESET),dim)
```

```
    # After having repaired all datasets, close the hdf5-file again.
    f.close()


# ------------------------------------------------------------------------
# Cycle over all csv-files and reset time trace.
for filename in os.listdir(SIM_DIR):

    # First perform some checks on files; cycle if file is binary, an
↪ exception
    # or a jobscript.
    if(is_binary(filename)):
        continue
    if(is_file_exception(filename)):
        continue
    if(is_jobscript(filename)):
        continue

    # no file
    if(not os.path.isfile(filename)):
        continue

    # Load csv file.
    data =  pd.read_csv(filename, header=None, sep="\s+").values

    #Check if any dimension is an integer multiple of NT_BROKE, by
↪ checking
    #the residual of the division.
    res = [None]*len(data.shape)

    # residul for output that holds time derivatives and therefore nt-1
↪ datapoints
    res_deriv = [None]*len(data.shape)
    for i in range(len(data.shape)):
        res[i] = data.shape[i]/NT_BROKE-np.floor(data.shape[i]/NT_BROKE)
        res[i] =
↪ data.shape[i]/(NT_BROKE_DERIV)-np.floor(data.shape[i]/(NT_BROKE_DERIV))

    # ordinary files
    if 0.0 in res:

        #Check which dimension is integer multiple of NT_BROKE
        #and save dimension as well as integer.
        for i in range(len(data.shape)):
            ncol = data.shape[i]/NT_BROKE
            res = ncol - np.floor(ncol)
            if(res == 0):
                dim = i
                ncol = int(ncol)
                break

        # Load original dataset.
        original_data = data
```

```
            # print("\t Original shape: \t"+str(original_data.shape))

            # Reset time trace.
            reset_data = reset_time_trace(original_data,dim,ncol,NT_RESET)
            # print("\t Reset shape: \t" +str(reset_data.shape))

            # Save resetted data.
            pd.DataFrame(reset_data).to_csv(filename, sep="\t", header=None,
    ↪ index=None)

        # files holding time derivatives
        if 0.0 in res_deriv:

            # Check which dimension is integer multiple of NT_BROKE
            # and save dimension as well as integer.
            for i in range(len(data.shape)):
                ncol = data.shape[i]/NT_BROKE_DERIV
                res = ncol - np.floor(ncol)
                if(res == 0):
                    dim = i
                    ncol = int(ncol)
                    break

            # Load original dataset.
            original_data = data

            # Reset time trace.
            reset_data =
    ↪ reset_time_trace(original_data,dim,ncol,NT_RESET_DERIV)

            # Save resetted data.
            pd.DataFrame(reset_data).to_csv(filename, sep="\t", header=None,
    ↪ index=None)


    # -----------------------------------------------------------------------
    # Finally, copy most recent dump file to FDS[/.dat].

    # Copy the most recent dump file to FDS[/.dat].
    copyfile(SIM_DIR+"/"+DUMPFILE, SIM_DIR+"/"+"FDS")
    copyfile(SIM_DIR+"/"+DUMPFILE+".dat", SIM_DIR+"/"+"FDS.dat")

    # First line of the so produced FDS.dat has to be modified.
    old_text = "!Dump filename: "+DUMPFILE
    new_text = "!Dump filename: "+"FDS"

    # Replace first line in FDS.dat to set the correct file name.
    with fileinput.input(SIM_DIR+"/"+"FDS.dat",inplace=True) as f:
        for line in f:
            line.replace(old_text, new_text)

def check_and_delete_file(filenames):
```

```
    for filename in filenames:

        if(os.path.isfile(folder + "/" + filename)):
            os.remove(folder + "/" + filename)


def check_checkpoint_files():
    DM1, DM2 = False, False

    if(os.path.isfile(folder + "/" + check1Filename) and
    ↪ os.path.isfile(folder + "/" + check1Bin)):
        DM1 = True
    if(os.path.isfile(folder + "/" + check2Filename) and
    ↪ os.path.isfile(folder + "/" + check2Bin)):
        DM2 = True

    return DM1, DM2

def get_timestep_from_restartfile(filename, flag):

    return int(get_value_of_variable_from_file(filename, 0, 2,
    ↪ flag).replace(",", ""))

def get_ntimestepCurrent(filenames):

    ntimestep = 0

    for f in filenames:

        if(os.path.isfile(folder + "/" + f)):
            ntimestepFile = get_timestep_from_restartfile(f, restartFlag)

            if ntimestepFile > ntimestep:
                ntimestep = ntimestepFile

    return ntimestep

def get_time_from_statusfile(filename, line_index):

    with open(folder + "/" + filename, "r") as file:
        line = file.readlines()[line_index]

        content = line.split(" ")

        if formatTable == None:
            time = content[-3]
        else:
            time = content[-2]

        time_sec = get_time_in_seconds(time)

        return time_sec
```

```
## TIME ==================================================================
↪ ===============================================

def format_num(time):
    if time < 10:
        return "0" + str(time)
    else:
        return str(time)

def get_time_as_string(sec):

    mins,  sec   = sec   // 60, sec   % 60
    hours, mins  = mins  // 60, mins  % 60
    days,  hours = hours // 24, hours % 24
    weeks, days  = days  // 7,  days  % 7

    timeConvertedString = (str(int(weeks)) + ":" +
                     format_num(int(days)) + ":" + format_num(int(hours)) +
    ↪ ":" +
                     format_num(int(mins)) + ":" + format_num(int(sec)))

    return timeConvertedString

def get_time_in_seconds(time):

    # Format D-HH:MM:SS or HH:MM:SS or MM:SS

    time = time.replace("-", ":")
    time = time.replace("\n", "")

    time_split = time.split(":")

    seconds = [7*24*60*60,24*60*60, 60*60, 60, 1]
    seconds = seconds[-len(time_split):]

    time_sec = sum([a*b for a,b in zip(seconds, map(int,time_split))])

    return time_sec

def time_date():
    e = datetime.datetime.now()
    return "%s-%s-%s" % (format_num(e.year), format_num(e.month),
    ↪ format_num(e.day))

def time_time():
    e = datetime.datetime.now()
    return "%s:%s:%s" % (format_num(e.hour), format_num(e.minute),
    ↪ format_num(e.second))

def time_duration(startTime, pastTime):
    stop = time.time()
    return get_time_as_string(stop - startTime + pastTime)
```

```
## STATUS ==============================================================
↪ ===============================================

def get_job_status():

    jobStatusRunning =
    ↪ subprocess.getoutput(commandJobRunning).strip().split()
    jobStatusPending =
    ↪ subprocess.getoutput(commandJobPending).strip().split()

    return jobStatusRunning, jobStatusPending

def set_output_type():
    jobStatusRunning, jobStatusPending = get_job_status()

    if jobName in jobStatusRunning:
        outputType = "running"
    elif jobName in jobStatusPending:
        outputType = "pending"
    else:
        outputType ="no Output"

    return outputType

def get_error_type(filename):

    slurm_errors = {"executable":["error on file ./gkw.x (No such file or
    ↪ directory)", "No executable found"],
                    "walltime":["process killed (SIGTERM)", "Exceeded wall
    ↪ time"],
                    "timeout":["DUE TO TIME LIMIT", "Exceeded time limit"],
                    "config":["couldn't open config directory", "Config not
    ↪ loading"],
                    "hdf5":["HDF5-DIAG", "Writing h5 file failed"]}

    for key in slurm_errors:
        if find_string_in_file(filename, slurm_errors[key][0]):
            return slurm_errors[key][1]

    return "Unknown error occurred"

## MAIL ================================================================
↪ ===============================================

def send_mail(recipient, subject, body = None):

    recipient = recipient.encode("utf_8")

    subject = subject.replace(" ", "_")
    subject = subject.encode("utf_8")

    if body == None:
        body = "For futher information open attachment"
```

```
    body = body.encode("utf_8")

    attachmentPath = folder + "/" + statusFilename
    attachment = attachmentPath.encode("utf_8")

    process = subprocess.Popen(["ssh", "master", "/usr/bin/mailx", "-s",
    ↪ subject, "-a",  attachment, recipient],
                               stdin=subprocess.PIPE)
    process.communicate(body)

# START/RESTART JOB =========================================================
    ↪ ===============================================

startTime = time.time()
outputType = set_output_type()
jobStatusRunning, jobStatusPending = get_job_status()

# Set current Time for progress bar
if outputType == "running":
    jobStatusRunningNameIndex = [idx for idx, s in
    ↪ enumerate(jobStatusRunning) if jobName in s][0]
    currentTime = jobStatusRunning[jobStatusRunningNameIndex + 3]
elif outputType == "pending":
    jobStatusPendingNameIndex = [idx for idx, s in
    ↪ enumerate(jobStatusPending) if jobName in s][0]
    currentTime = jobStatusPending[jobStatusPendingNameIndex + 3]
else:
    currentTime = "00:00:00"

# Set pastTime and create status file if necessary. When status file exist
    ↪ append next lines
WRITEHEADER = True
if not os.path.isfile(folder + "/" + statusFilename):
    pastTime = 0
    write_file(statusFilename,"")
else:
    try:
        pastTime = get_time_from_statusfile(statusFilename, -11)
        WRITEHEADER = False
    except IndexError:
        pastTime = 0

print_table_row(["OUTPUT", "INFO"], output_type="header",
    ↪ WRITEFILE=WRITEHEADER)

## BEGIN ====================================================================
    ↪ ===============================================

# Check if timesteps criterion is satisfied, send mail and end monitoring
# else continue monitoring and set output type accordingly
nTimestepsCurrent = get_ntimestepCurrent([restartFilename, check1Filename,
    ↪ check2Filename])
```

```
if nTimestepsCurrent >= nTimestepsRequired:
    print_table_row(["CONTROL", "Current Timesteps " +
    ↪ str(nTimestepsCurrent)], output_type="middle")
    print_table_row(["SUCCESS", "Stop monitoring " + jobName])

    if EMAIL:
        send_mail(emailAddress, "Ended Job " + jobName)

    quit()

# Continue monitoring and send mail
else:

    if outputType != "no Output":
        print_table_row(["CONTINUE", "Continue monitoring " + jobName],
    ↪ output_type="middle")
        print_table_row(["CONTROL", "Current Timesteps " +
    ↪ str(nTimestepsCurrent)])

        if EMAIL:
            send_mail(emailAddress, "Continued Job " + jobName)

    elif os.path.isfile(folder + "/" + restartFilename):
        print_table_row(["CONTINUE", "Continue monitoring " + jobName],
    ↪ output_type="middle")

        if EMAIL:
            send_mail(emailAddress, "Continued Job " + jobName)

    else:
        print_table_row(["STARTING", "Start monitoring " + jobName],
    ↪ output_type="middle")

        if BACKUP:
            print_table_row(["BACKUP", backupPath])
            subprocess.run(commandBackup)

        if EMAIL:
            send_mail(emailAddress, "Started Job " + jobName)

if RESET:
    pip_install({"h5py", "pandas", "numpy"})

    import h5py, fileinput
    import pandas as pd
    import numpy as np
    from shutil import copyfile

    print_table_row(["IMPORT", "Load numpy, pandas, h5py"])

#else:
#    pip_install({"h5py"})
```

```
#    import h5py
#    print_table_row(["IMPORT", "Load module h5py"])

## MONITOR ROUTINE ==========================================================
   ↪ ===============================================

while True:

    # Check job status to monitor current state
    jobStatusRunning, jobStatusPending = get_job_status()

    # Job running
    if jobName in jobStatusRunning:

        jobStatusRunningNameIndex = [idx for idx, s in
    ↪ enumerate(jobStatusRunning) if jobName in s][0]
        jobID = jobStatusRunning[jobStatusRunningNameIndex - 2]

        currentTime = jobStatusRunning[jobStatusRunningNameIndex + 3]
        nTimestepsCurrent = get_ntimestepCurrent([restartFilename,
    ↪ check1Filename, check2Filename])

        if outputType == "running":
            print_table_row(["RUNNING", "Job is executed"])
            outputType = "no Output"
        else:
            print_table_row(["RUNNING", "Job is executed"],
    ↪ output_type="update")

        time.sleep(sleepTime)

    # Job pending
    elif jobName in jobStatusPending:

        jobStatusPendingNameIndex = [idx for idx, s in
    ↪ enumerate(jobStatusPending) if jobName in s][0]
        jobID = jobStatusPending[jobStatusPendingNameIndex - 2]

        currentTime = jobStatusPending[jobStatusPendingNameIndex + 3]
        nTimestepsCurrent = get_ntimestepCurrent([restartFilename,
    ↪ check1Filename, check2Filename])

        if outputType == "pending":
            print_table_row(["WAITING", "Job is pending"])
            outputType = "running"
        else:
            print_table_row(["WAITING", "Job is pending"],
    ↪ output_type="update")

        time.sleep(sleepTime)

    # Job start/restart
    else:
```

```
    # Check errors and making Backup
    while True:
        try:
            outputContent = open(folder + "/" +
↪ outputFilename(jobID)).readlines()[-5].replace("\n","")

            # Create Backup if run is successful
            # If scan of output.dat is needed: Scans for string "Run
↪ Successful in output.dat and returns bool value"
            #runSuccess = find_string_in_file("output.dat",
↪ outputCriteria[1])
            #if outputCriteria[0] in outputContent and runSuccess:

            if outputCriteria[0] in outputContent:

                # Check if h5 file is closed before start/restart
↪ simulation
                # than check if FDS/FDS.dat is updated
                try:
                    f = open(folder + "/" + dataFilename)
                    #f = h5py.File(dataFilename)
                    f.close()

                    # Check if FDS/FDS.dat is updated after run and has
↪ equally time stamp as gkwdata.h5
                    timestamp_data    = int(os.path.getmtime(folder +
↪ "/" + dataFilename))
                    timestamp_restart = int(os.path.getmtime(folder +
↪ "/" + restartFilename))

                    wallTime_sec = get_time_in_seconds(wallTime)
                    timestamp_remain = timestamp_data - timestamp_restart

                    # FDS/FDS.dat does not get written at the same time
↪ as gkwdata.h5
                    # For that a time interval have to be considered
                    # To be certain the half wall time is set aus time
↪ interval
                    if timestamp_remain > wallTime_sec/2:

                        print_table_row(["ERROR", "FDS/FDS.dat not
↪ updated"])

                        # Reset simulation and save as backup
                        if RESET:

                            DM1, DM2 = check_checkpoint_files()

                            if (DM1 or DM2):
                                print_table_row(["RESET", "Reset to last
↪ checkpoint."])
                                reset_simulation(folder)
```

```
                                # Update backup
                                if BACKUP:
                                    print_table_row(["BACKUP",
↪ backupPath])
                                    subprocess.run(commandBackup)

                            # Restore backup to rerun simulation
                            elif BACKUP:
                                print_table_row(["RESTORE", backupPath])
                                subprocess.run(commandRestore)

                        elif BACKUP:
                            print_table_row(["BACKUP", backupPath])
                            subprocess.run(commandBackup)

                        break
                    except OSError:
                        time.sleep(sleepTime)

                else:
                    print_table_row(["ERROR",
↪ get_error_type(outputFilename(jobID))])

                    # Reset simulation and save as backup
                    if RESET:

                        DM1, DM2 = check_checkpoint_files()

                        if (DM1 or DM2):
                            print_table_row(["RESET", "Reset to last
↪ checkpoint."])
                            reset_simulation(folder)

                            # Update backup
                            if BACKUP:
                                print_table_row(["BACKUP", backupPath])
                                subprocess.run(commandBackup)

                    # Restore backup to rerun simulation
                    elif BACKUP:
                        print_table_row(["RESTORE", backupPath])
                        subprocess.run(commandRestore)

                    break

        # Wait sleepTime and check output file again
        except (IndexError, FileNotFoundError):
            time.sleep(sleepTime)

        # If jobID undefined break cycle
        except NameError:
            break
```

```
        # Check if timesteps criterion is satisfied, send mail and end
    ↪ monitoring
        nTimestepsCurrent = get_ntimestepCurrent([restartFilename,
    ↪ check1Filename, check2Filename])
        print_table_row(["CONTROL", "Current Timesteps " +
    ↪ str(nTimestepsCurrent)])

        if nTimestepsCurrent >= nTimestepsRequired:
            print_table_row(["SUCCESS", "Stop monitoring " + jobName])

            if EMAIL:
                send_mail(emailAddress, "Ended Job " + jobName)
            break

        # Delete checkpoint files
        check_and_delete_file([check1Bin, check1Filename, check2Bin,
    ↪ check2Filename])

        # Create jobscript
        if jobscriptFilename == "jobscript-create":
            jobscriptFilename = "jobscript"
            write_file(jobscriptFilename, jobscriptContent)

        # Start Job and send restart mail (if activated)
        startOutput = subprocess.check_output([commandJobStarting, folder +
    ↪ "/" + jobscriptFilename]).decode("utf-8").replace("\n", "")
        jobID = startOutput.split(startOutputFlag)[1]

        runCounter += 1

        print_table_row(["STARTING", startOutput], output_type="middle")

        try:
            if RESTARTMAIL and EMAIL:
                send_mail(emailAddress, "Restart Job " + jobName)
        except NameError:
            continue

        time.sleep(30)

        # Set output type to running or pending
        outputType = set_output_type()

## RESTART ==================================================================
    ↪ ==============================================
```

6.2 Status File

```
--------------------------------------------------------------------------------
 OUTPUT    INFO                           DATE        TIME
   ↪ W:DD:HH:MM:SS
--------------------------------------------------------------------------------
 CONTINUE  Continue monitoring noise      2023-05-22  14:13:22
   ↪ 0:00:00:00:00
 CONTROL   Current Timesteps 1000         2023-05-22  14:13:22
   ↪ 0:00:00:00:00
 RUNNING   Job is executed                2023-05-22  14:13:23
   ↪ 0:00:00:00:01
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
   JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
 1111400    normal    noise bt712347  R   13:57:17      6
   ↪ r01n[35-36],r06n[08
--------------------------------------------------------------------------------
 PROGRESS  [=========>.................................]  ( 20%)
   ↪ 1000/5000
 RUN 0     [=========================>.................]  ( 58%)
   ↪ 50237/86400
--------------------------------------------------------------------------------
```

6.3 torque_monitor.py

```
#!/usr/bin/env python3

# AUTHOR: Manuel Lippert (GitHub: ManeLippert
    ↪ (https://github.com/ManeLippert))
#
# DESCRIPTION:
# Script that starts a given job (shell script) with Sun  Grid  Engine until
    ↪ the previous defined timestep are completed.
# It also sends an mail alert when the job gets started and ended.

# IMPORTANT:
# Take a look in the Variable Section but overall everthing should work
    ↪ automatically

# START SCRIPT:
# To start the script in the background following command is needed:
#
# >>> nohup python3 -u monitor_job.py &> status.txt &
#
# Output:
#
# >>> [1] 10537
#
# This will write every output in the file jobstatus.txt that will be send
    ↪ as attachment to the defined mail address.

# LIST PRGRESS:
# To see which progress is in background running following command is needed:
#
# >>> ps ax | grep monitor_job.py
#
# Output:
#
# >>> 10537 pts/1    S      0:00 python3 -u monitor_job.py
# >>> 23426 pts/1    S+     0:00 grep --color=auto monitor_job.py
#
# This will give you the ID to kill monitoring script with the command:
#
# >>> kill 10537
#
# This will kill the monitor script.

# OUTPUT STATUS:
# To see all status.txt files with one command follwing command is needed:
#
# >>> cd $DATA && find . -name status.txt -exec tail --lines=+10 {} \;


'''
```

```
########################################################
#                                                      #
#                       Modules                        #
#                                                      #
########################################################
'''

import datetime
import time
from time import sleep
import os
import subprocess

'''
########################################################
#                                                      #
#                      VARIABLES                       #
#                                                      #
########################################################
'''

##################### ADDITIONAL #######################

#emailAddress = 'Dominik.Mueller@uni-bayreuth.de'
#backupLocation = '/scratch/bt712347/backup'

##################### FILENAMES ########################

# Finds File that ends with .sh
for file in os.listdir():
    if file.endswith('.sh'):
        jobscriptFilename = file

# Rename jobscript
dirName = os.getcwd().split('/')[-1]
os.rename(jobscriptFilename, dirName + '.sh')
jobscriptFilename = dirName + '.sh'

monitorFilename = 'status.txt'

# Finds File that ends with .json
for file in os.listdir():
    if file.endswith('.json'):
        inputFilename = file

restartFilename = inputFilename

# Declared as function for dynamic changes
def outputFilename(info):
    return jobscriptFilename + '.o' + info

####################### FLAGS ##########################
```

```
walltimeFlag = 'time='

startOutputFlag = '.mgmt'

outputCriteria = 'WARNING'

inputFlag = '"Number'

restartFlag = 'ETA:'
restartString = '        "Restart from step": '

#################### SLEEP TIME #########################

sleepTime = 5*60

##################### COMMANDS ##########################

commandJobStatus = 'qstat -u'
commandJobStarting = 'qsub'

'''
#########################################################
#                                                       #
#                      FUNCTIONS                        #
#                                                       #
#########################################################
'''

#################### INFORMATIONS #######################

def read_file_to_string(file):
    content = ''.join(open(file).readlines())

    return content

def get_value_of_variable_from_input_file(file, string, idx):
    try:
        content = [i.strip().split() for i in open(file).readlines()]
        index = [idx for idx, s in enumerate(content) if string in s][idx]
        value = int(content[index][4].split(',')[0])
        return value
    except IndexError:
        print('! String not in input file !')
        quit()

def get_value_of_variable_from_output_file(file, string, idx):
    try:
        content = [i.strip().split() for i in open(file).readlines()]
        index = [idx for idx, s in enumerate(content) if string in s][idx]
        value = int(content[index - 1][0])
        return value
    except IndexError:
        print('! String not in input file !')
```

```
        quit()

def get_job_information_from_jobscript_flag(content, flag):
    index = [idx for idx, s in enumerate(content) if flag in s][0]
    info = content[index].split(flag, 1)[1]

    return info

######################## FILE ###########################

def write_add_string_into_file(file, substring, add, comment = None):
    # open file
    with open(file, 'r') as f:
        # read a list of lines into data
        data = f.readlines()

    try:
        index = [idx for idx, s in enumerate(data) if substring in s][0]
        data[index] = substring + add + '\n'

    except IndexError:
        index = [idx for idx, s in enumerate(data) if '\n' in s][1]
        data.insert(index, '\n')
        data.insert(index + 1, comment)
        data.insert(index + 2, substring + add + '\n')

    # replace line
    with open(file, 'w') as f:
        f.writelines(data)

######################## TIME ###########################

def get_time_in_seconds(time):
    # Check time format of time
    if len(time) < 5:
        ## d:hh:mm:ss
        if len(time) == 4:
            timeSeconds = int(time[0])*24*60*60 + int(time[1])*60*60 +
    ↪ int(time[2])*60 + int(time[3])
        ## hh:mm:ss
        elif len(time) == 3:
            timeSeconds = int(time[0])*60*60 + int(time[1])*60 + int(time[2])
        ## mm:ss
        elif len(time) == 2:
            timeSeconds = int(time[0])*60 + int(time[1])
        ## ss
        elif len(time) == 1:
            timeSeconds = int(time[0])
    else:
        print('! Time format is not supported !')
        quit()

    return timeSeconds
```

```
def format_num(time):
    if time < 10:
        return '0' + str(time)
    else:
        return str(time)

def get_time_converted(sec):
    mins, sec  = sec // 60, sec % 60
    hours, mins = mins // 60, mins % 60
    days, hours = hours // 24, hours % 24
    weeks, days = days // 7, days % 7

    timeConverted = (str(int(weeks)) + ':' +
                     format_num(int(days)) + ':' + format_num(int(hours)) + ':' +
                     format_num(int(mins)) + ':' + format_num(int(sec)))

    return timeConverted

def time_date():
    e = datetime.datetime.now()
    return '%s-%s-%s' % (format_num(e.year), format_num(e.month),
    ↪ format_num(e.day))

def time_time():
    e = datetime.datetime.now()
    return '%s:%s:%s' % (format_num(e.hour), format_num(e.minute),
    ↪ format_num(e.second))

def time_duration(startTime):
    stop = time.time()
    return get_time_converted(stop - startTime)

####################### TABLE ##########################

def table_row_format(content):
    if len(content) == 7:
        cols = [2, 10, 29, 13, 11, 13, 2]
    elif len(content) == 6:
        cols = [2, 10, 29, 13, 24, 2]
    elif len(content) == 5:
        cols = [2, 10, 25, 41, 2]
    elif len(content) == 4:
        cols = [2, 10, 66, 2]
    else:
        cols = [2, 76, 2]

    i, sep = 0, []

    while i < len(cols):
        if i == 0:
            sep.append('o' + (cols[i]-1)*'-')
        elif i == (len(cols)-1):
```

```
            sep.append((cols[i]-1)*'-' + 'o')
        else:
            sep.append(cols[i]*'-')

        i += 1

    row_format = "".join(["{:<" + str(col) + "}" for col in cols])

    return row_format, sep

def print_table_row(content, output_type = None, time_info = True):

    if isinstance(content[0], list):

        i = 0
        while i < len(content):
            content[i].insert(0, '| ')
            if time_info:
                if output_type == 'header':
                    content[i].append('DATE')
                    content[i].append('TIME')
                    content[i].append('W:DD:HH:MM:SS')
                else:
                    content[i].append(time_date())
                    content[i].append(time_time())
                    content[i].append(time_duration(startTime))
            content[i].insert(len(content[i]), ' |')

            i += 1

        row_format, sep = table_row_format(content[0])

        if output_type == 'header':
            print('\n')
            print(row_format.format(*sep))
            for row in content:
                print(row_format.format(*row))
            print(row_format.format(*sep))

        elif output_type == 'end':
            for row in content:
                print(row_format.format(*row))
            print(row_format.format(*sep))

        else:
            for row in content:
                print(row_format.format(*row))
    else:
        content.insert(0, '| ')
        if time_info:
            if output_type == 'header':
                content.append('DATE')
                content.append('TIME')
```

```
                content.append('W:DD:HH:MM:SS')
            else:
                content.append(time_date())
                content.append(time_time())
                content.append(time_duration(startTime))
        content.insert(len(content), ' |')

        row_format, sep = table_row_format(content)

        if output_type == 'header':
            print('\n')
            print(row_format.format(*sep))
            print(row_format.format(*content))
            print(row_format.format(*sep))

        elif output_type == 'end':
            print(row_format.format(*content))
            print(row_format.format(*sep))

        else:
            print(row_format.format(*content))

####################### STATUS #########################

def get_job_status(user):
    jobStatus = subprocess.getoutput(commandJobStatus + user).strip().split()

    return jobStatus

def set_output_type(user):
    jobStatus = get_job_status(user)

    if jobName in jobStatus:
        outputType = 'running'
    else:
        outputType ='no Output'

    return outputType

####################### MAIL ###########################

def send_mail(recipient, subject, body = None):

    recipient = recipient.encode('utf_8')

    subject = '"' + subject + '"'
    subject = subject.encode('utf_8')

    if body == None:
        body = 'For futher information open attachment'

    body = body.encode('utf_8')
```

```
    attachmentPath = folder + '/' + monitorFilename
    attachment = attachmentPath.encode('utf_8')

    process = subprocess.Popen(['ssh', 'master', '/usr/bin/mailx', '-s',
    ↪ subject, '-a',  attachment, recipient],
                               stdin=subprocess.PIPE)
    process.communicate(body)

'''
#########################################################
#                                                       #
#                  JOB INFORMATIONS                     #
#                                                       #
#########################################################
'''

################### OUTPUT STRING ######################

jobInformations = []

#################### START WATCH #######################

startTime = time.time()

#################### OUTPUT INIT #######################

print_table_row(['OUTPUT', 'JOB INITIALIZE'], output_type='header')

####################### USER ###########################

user = os.getlogin()

####################### JOB NAME #######################

jobName = jobscriptFilename

if len(jobName) > 16:
    jobName = jobName[0:16]

jobInformations.append(['INFO', 'Name', jobName])

#################### FILE PATH #########################

folder = os.path.dirname(os.path.abspath(__file__))
path = os.path.dirname(os.path.abspath(__file__)).split(user + '/')[1]

################## MAIL ADDRESS ########################

try:
    emailNotification = True
    jobInformations.append(['INFO', 'E-Mail', emailAddress])
except NameError:
    emailNotification = False
```

```
#################### JOB SCRIPT ########################

try:
    jobscript = [filename for filename in os.listdir('.') if
    ↪ filename.startswith(jobscriptFilename)][0]
    print_table_row(['SUCCESS', 'Found ' + 'jobscript file'])

except IndexError:
    print_table_row(['ERROR', 'No jobscript found'])
    # Send error mail
    if emailNotification:
        send_mail(emailAddress, 'Failed Job ' + jobName)
    quit()

jobscriptContent = open(jobscript, 'r').read().splitlines()

##################### TIMESTEPS ########################

# Number of required timesteps
try:
    nTimestepsRequired = 0

    for i in range(10):
        nTimestepsRequired += get_value_of_variable_from_input_file('./' +
    ↪ inputFilename, inputFlag, i)

    print_table_row(['SUCCESS', 'Found ' + 'input file'], output_type='end',
    ↪ time_info=True)
except FileNotFoundError:
    print_table_row(['ERROR', 'No input file found'], output_type='end')
    # Send error mail
    if emailNotification:
        send_mail(emailAddress, 'Failed Job ' + jobName)
    quit()

jobInformations.append(['INFO', 'Required Timesteps', nTimestepsRequired])

#################### WALLTIME ###########################

walltime = get_job_information_from_jobscript_flag(jobscriptContent,
    ↪ walltimeFlag).replace('-', ':').split(':')
walltimeSeconds = get_time_in_seconds(walltime)

jobInformations.append(['INFO', 'Walltime/s', walltimeSeconds])

################### BACKUP PATH ########################

try:
    # Backup location
    if backupLocation[-1] != '/':
        backupLocation += '/'
```

```
    backupPath = backupLocation + path

    # Create backup directory if do not exist
    if not os.path.exists(backupPath):
        os.makedirs(backupPath)

    # BackUp switch
    backup = True

    jobInformations.append(['INFO', 'Backup Path', backupLocation])

except NameError:
    # BackUp switch
    backup = False

################### OUTPUT INFO #########################

print_table_row(['OUTPUT', 'JOB INFORMATIONS', 'VALUE'],
    ↪ output_type='header', time_info=False)
print_table_row(jobInformations, output_type='end', time_info=False)


'''
#########################################################
#                                                       #
#                   START/RESTART JOB                   #
#                                                       #
#########################################################
'''

################## OUTPUT MONITOR #######################

print_table_row(['OUTPUT', 'JOB MONITORING'], output_type='header')

####################    BEGIN    ########################

outputType = set_output_type(user)

# read restart file
## If gkw has run requiered timesteps stop already here
while True:
    try:

        identity = ['']
        # find last output file
        for file in os.listdir():
            if '.sh.o' in file:
                identity.append(file.split('.sh.o')[1])

        nTimestepsCurrent = get_value_of_variable_from_output_file('./' +
    ↪ outputFilename(max(identity)), restartFlag, -1)
        nTimestepsCurrent = int((nTimestepsCurrent // 1e4) * 1e4)
```

```
        # Output current timesteps
        if outputType == 'running':
            print_table_row(['CONTROL', 'Current Timesteps ' +
    ↪ str(nTimestepsCurrent)])

        # Check if gkw has run requiered timesteps
        if nTimestepsCurrent >= nTimestepsRequired:
            print_table_row(['SUCCESS', 'Stop monitoring'],
    ↪ output_type='end')

            # Send end email
            if emailNotification:
                send_mail(emailAddress, 'Ended Job ' + jobName)

            quit()

        # Continue
        else:
            print_table_row(['CONTINUE', 'Continue monitoring'])

            # Send continue mail
            if emailNotification:
                send_mail(emailAddress, 'Continued Job ' + jobName)
            break

    # Start
    except (FileNotFoundError, NameError):
        print_table_row(['STARTING', 'Start monitoring'])

        # Making backup
        if backup:
            print_table_row(['BACKUP', backupLocation], output_type='end')
            subprocess.run(['rsync', '-a', '', backupPath])

        # Send start mail
        if emailNotification:
            send_mail(emailAddress, 'Started Job ' + jobName)
        break

################# MONITOR ROUTINE #####################

while True:

    jobStatus = get_job_status(user)

    # Job running
    if jobName in jobStatus:
        # Job ID Running
        jobStatusNameIndex = [idx for idx, s in enumerate(jobStatus) if
    ↪ jobName in s][0]
        jobID = jobStatus[jobStatusNameIndex -3].split(startOutputFlag)[0]

        # Set output type
```

```
    if outputType == 'running':
        print_table_row(['RUNNING', 'Job is executed'])
        outputType = 'no Output'

    sleep(sleepTime)

# Job start/restart
else:

    # Check error and making Backup
    while True:
        try:
            outputContent = read_file_to_string('./' +
↪ outputFilename(jobID))

            if outputCriteria in outputContent:

                print_table_row(['ERROR', 'NaN Value in surf_dens'])

                quit()

            else:

                # Making backup
                if backup:
                    print_table_row(['BACKUP', backupLocation],
↪ output_type='end')
                    subprocess.run(['rsync', '-a', '', backupPath])

                outputType = 'restart'

                break

        # If jobID is not defined or file is not generated
        except (IndexError, FileNotFoundError):
            sleep(30)
        except NameError:
            break

    # Check Timesteps
    try:
        nTimestepsCurrent = get_value_of_variable_from_output_file('./'
↪ + outputFilename(jobID), restartFlag, -1)
        nTimestepsCurrent = int((nTimestepsCurrent // 1e4) * 1e4)

        print_table_row(['CONTROL', 'Current Timesteps ' +
↪ str(nTimestepsCurrent)])

        # Check if gkw has run requiered timesteps
        if nTimestepsCurrent >= nTimestepsRequired:
            print_table_row(['SUCCESS', 'Stop monitoring'],
↪ output_type='end')
```

```
                # Send end email
                if emailNotification:
                        send_mail(emailAddress, 'Ended Job ' + jobName)
                break

            # write restart timestep in input file
            else:
                write_add_string_into_file(restartFilename, restartString,
↪ str(nTimestepsCurrent))

        except (FileNotFoundError, NameError):
            pass

        # Start Job
        startOutput = subprocess.check_output([commandJobStarting,
↪ jobscriptFilename]).decode('utf-8').replace('\n', '')
        jobID = startOutput.split(startOutputFlag)[0]

        print_table_row(['STARTING', startOutput])

        # Set output type
        if outputType == 'restart':
            # Send end email
            if emailNotification:
                    send_mail(emailAddress, 'Restart Job ' + jobName)

        sleep(30)

        outputType = set_output_type(user)

##################### RESTART ##########################
```

6.4 Simulation parameter

	R/L_T	box size	N_s	$N_{\nu_\parallel}$	N_μ	d_{tim}	$k\rho^{max}$	N_{mod}	N_x	ikx	finit
1	6.0	1x1	12	16	6	0.02	1.4	21	83	5	cosine2
2	6.0	1x1	12	32	6	0.02	1.4	21	83	5	cosine2
3	6.0	1x1	12	48	9	0.02	1.4	21	83	5	cosine2
4	6.0	1x1	12	64	9	0.02	1.4	21	83	5	cosine2
5	6.0	1x1	16	16	9	0.02	1.4	21	83	5	cosine2
6	6.0	1x1	16	32	9	0.02	1.4	21	83	5	cosine2
7	6.0	1x1	16	48	6	0.02	1.4	21	83	5	cosine2
8	6.0	1x1	16	48	9	0.02	1.4	21	83	5	cosine2
9	6.0	1x1	16	48	9	0.025	1.4	21	83	5	cosine2
10	6.0	1x1	16	48	9	0.02	0.7	11	83	5	cosine2
11	6.0	1x1	16	48	9	0.02	1.4	21	43	5	cosine2
12	6.0	1x1	16	48	9	0.02	1.4	21	63	5	cosine2
13	6.0	1x1	16	64	6	0.02	1.4	21	83	5	cosine2
14	6.0	1x1	16	64	9	0.02	1.4	21	83	5	cosine2
15	6.0	1.5x1.5	16	48	9	0.02	1.4	31	125	5	cosine2
16	6.0	2x1	16	48	9	0.02	1.4	21	83	10	cosine2
17	6.0	2x2	16	48	9	0.02	1.4	21	83	5	cosine2
18	6.0	2.5x2.5	16	48	9	0.02	1.4	51	207	5	cosine2
19	6.0	3x1	16	48	9	0.02	1.4	21	83	15	cosine2
20	6.0	3x1.5	16	48	9	0.02	1.4	21	83	10	cosine2
21	6.0	3x1.5	16	48	9	0.02	1.4	21	83	10	cosine2
22	6.0	3x2.5	16	48	9	0.02	1.4	21	83	6	cosine2
23	6.0	3x3	16	48	9	0.02	1.4	21	83	5	cosine2
24	6.0	3x3	16	48	9	0.02	1.4	21	83	5	cosine5
25	6.0	3x3	16	48	9	0.02	1.4	21	83	5	noise
26	6.0	3x5	16	48	9	0.02	1.4	21	83	3	cosine2
27	6.0	4x1	16	48	9	0.02	1.4	21	83	20	cosine2
28	6.2	2x2	16	64	9	0.02	1.4	21	83	5	cosine2
29	6.2	3x3	16	48	9	0.02	1.4	21	83	5	cosine2
30	6.3	1x1	16	64	9	0.02	1.4	21	83	5	cosine2
31	6.4	3x3	16	48	9	0.02	1.4	21	83	5	cosine2

Table 6.1: Parameters of simulations performed for this thesis

	R/L_T	box size	time	timestep	error_index	stable	n_{ZF}	backup
1	6.0	1x1	6000.0	10000		False	0	True
2	6.0	1x1	12000.0	20000		False	0	True
3	6.0	1x1	6000.0	10000		False	0	True
4	6.0	1x1	6000.0	10000		False	0	True
5	6.0	1x1	12000.0	20000		False	0	True
6	6.0	1x1	12000.0	20000		False	0	True
7	6.0	1x1	6000.0	10000		False	0	True
8	6.0	1x1	6000.0	10000		True	1	True
9	6.0	1x1	7125.0	10000		True	1	True
10	6.0	1x1	6000.0	10000		True	1	True
11	6.0	1x1	6000.0	10000		False	0	True
12	6.0	1x1	6000.0	10000		False	0	True
13	6.0	1x1	12000.0	20000		True	1	True
14	6.0	1x1	12000.0	20000		True	1	True
15	6.0	1.5x1.5	6000.0	10000		True	2	True
16	6.0	2x1	18000.0	30000		True	2	True
17	6.0	2x2	18000.0	30000		True	2	True
18	6.0	2.5x2.5	3402.1	5672		True	3	True
19	6.0	3x1	48000.0	80000		True	3	True
20	6.0	3x1.5	16840.3	28068		True	4	True
21	6.0	3x1.5	14170.0	23618	23618	True	3	True
22	6.0	3x2.5	6000.0	10000		True	3, 4	True
23	6.0	3x3	7847.0	13083		True	4	True
24	6.0	3x3	3454.8	5758	1840-1845	True	4	True
25	6.0	3x3	5808.0	9680	1840-1842	True	3	True
26	6.0	3x5	4769.0	7958		True	4	True
27	6.0	4x1	30000.0	50000	47084	True	4	True
28	6.2	2x2	6000.0	10000		True	2	True
29	6.2	3x3	14682.7	24475	14473-16132	True	3	True
30	6.3	1x1	6000.0	10000		False	0	True
31	6.4	3x3	12000.0	20000		False	0	True

Table 6.2: Time steps with index which should be exclude in data analysis. Additional the stable column which indicates if turbulence subdued in simulation with the corresponding zonal flow mode number n_{ZF} (0 stands for turbulence not stable) and if the data is saved on the NAS of TPV (backup column)

	path
1	./data/S6_rlt6.0/boxsize1x1/Ns12/Nvpar16/Nmu6
2	./data/S6_rlt6.0/boxsize1x1/Ns12/Nvpar32/Nmu6
3	./data/S6_rlt6.0/boxsize1x1/Ns12/Nvpar48/Nmu9
4	./data/S6_rlt6.0/boxsize1x1/Ns12/Nvpar64/Nmu9
5	./data/S6_rlt6.0/boxsize1x1/Ns16/Nvpar16/Nmu9
6	./data/S6_rlt6.0/boxsize1x1/Ns16/Nvpar32/Nmu9
7	./data/S6_rlt6.0/boxsize1x1/Ns16/Nvpar48/Nmu6
8	./data/S6_rlt6.0/boxsize1x1/Ns16/Nvpar48/Nmu9
9	./data/S6_rlt6.0/boxsize1x1/Ns16/Nvpar48/Nmu9/dtim0.025
10	./data/S6_rlt6.0/boxsize1x1/Ns16/Nvpar48/Nmu9/krhomax0.70/Nmod11
11	./data/S6_rlt6.0/boxsize1x1/Ns16/Nvpar48/Nmu9/Nx43
12	./data/S6_rlt6.0/boxsize1x1/Ns16/Nvpar48/Nmu9/Nx63
13	./data/S6_rlt6.0/boxsize1x1/Ns16/Nvpar64/Nmu6
14	./data/S6_rlt6.0/boxsize1x1/Ns16/Nvpar64/Nmu9
15	./data/S6_rlt6.0/boxsize1.5x1.5/Ns16/Nvpar48/Nmu9
16	./data/S6_rlt6.0/boxsize2x1/Ns16/Nvpar48/Nmu9
17	./data/S6_rlt6.0/boxsize2x2/Ns16/Nvpar48/Nmu9
18	./data/S6_rlt6.0/boxsize2.5x2.5/Ns16/Nvpar48/Nmu9
19	./data/S6_rlt6.0/boxsize3x1/Ns16/Nvpar48/Nmu9
20	./data/S6_rlt6.0/boxsize3x1.5/Ns16/Nvpar48/Nmu9
21	./data/S6_rlt6.0/boxsize3x1.5/Ns16/Nvpar48/Nmu9/Broken
22	./data/S6_rlt6.0/boxsize3x2.5/Ns16/Nvpar48/Nmu9
23	./data/S6_rlt6.0/boxsize3x3/Ns16/Nvpar48/Nmu9
24	./data/S6_rlt6.0/boxsize3x3/Ns16/Nvpar48/Nmu9/cosine5
25	./data/S6_rlt6.0/boxsize3x3/Ns16/Nvpar48/Nmu9/noise
26	./data/S6_rlt6.0/boxsize3x5/Ns16/Nvpar48/Nmu9
27	./data/S6_rlt6.0/boxsize4x1/Ns16/Nvpar48/Nmu9
28	./data/S6_rlt6.2/boxsize2x2/Ns16/Nvpar64/Nmu9
29	./data/S6_rlt6.2/boxsize3x3/Ns16/Nvpar48/Nmu9
30	./data/S6_rlt6.3/boxsize1x1/Ns16/Nvpar64/Nmu9
31	./data/S6_rlt6.4/boxsize3x3/Ns16/Nvpar48/Nmu9

Table 6.3: Data location for each simulation

Size convergence of the $\mathrm{E}\times\mathrm{B}$ staircase pattern in flux tube simulations of ion temperature gradient-driven turbulence

M. Lippert,[1, a)] F. Rath,[1, b)] and A. G. Peeters[1]
Physics Department, University of Bayreuth, 95440 Bayreuth, Germany

(Dated: 23 June 2023)

The radial size convergence of the $\mathrm{E}\times\mathrm{B}$ staircase pattern is addressed in local gradient-driven flux tube simulations of ion temperature gradient (ITG)-driven turbulence. Its is shown that a mesoscale pattern size of $\sim 57-76\rho$ is inherent to ITG-driven turbulence with Cyclone Base Case parameters in the local limit.

Ion temperature gradient-driven turbulence close to marginal stability exhibits zonal flow pattern formation on mesoscales, so-called $\mathrm{E}\times\mathrm{B}$ staircase structures[1]. Such pattern formation has been observed in local gradient-driven flux-tube simulations[2,3], including collisions[4] and background $\mathrm{E}\times\mathrm{B}$ shear[3], local flux-driven realizations including mean electric field shear[5], as well as global gradient-driven[6–8] and global flux-driven[1,9–12] studies. In global studies, spanning a larger fraction of the minor radius, multiple radial repetitions of staircase structures are usually observed, with a typical pattern size of several ten Larmor radii. By contrast, in the aforementioned local studies the radial size of $\mathrm{E}\times\mathrm{B}$ staircase structures is always found to converge to the radial box size of the flux tube domain. The above observations lead to the question: *Does the basic pattern size always converges to the box size, or is there a typical mesoscale size inherent to staircase structures also in a local flux-tube description?* The latter case would imply that it is not necessarily global physics, i.e., profile effects, that set (i) the radial size of the $\mathrm{E}\times\mathrm{B}$ staircase pattern and (ii) the scale of avalanche-like transport events. These transport events are usually restricted to $\mathrm{E}\times\mathrm{B}$ staircase structures and considered as a nonlocal transport mechanism[1]. In this brief communication the above question is addressed through a box size convergence scan of the same cases close to the nonlinear threshold for turbulence generation as studied in Ref. 2.

The gyrokinetic simulations are performed with the nonlinear flux tube version of Gyrokinetic Workshop (GKW)[13] with adiabatic electron approximation. In agreement with Ref. 2, Cyclone Base Case (CBC) like parameters are chosen with an inverse background temperature gradient length $R/L_T = 6.0$ and circular concentric flux surfaces. The numerical resolution is compliant to the "Standard resolution with 6th order (S6)" set-up of the aforementioned reference, with a somewhat lowered number of parallel velocity grid points. It has been carefully verified that this modification preserves the same physical outcome as the original study. A summary of the numerical parameters is given in Table I and for more details about the definition of individual quantities the reader is referred to Ref. 2 and 13.

	N_m	N_x	N_s	$N_{v_\parallel}$	N_μ	D	v_d	$D_{v_\parallel}$	D_x	D_y	Order	$k_y\rho$	$k_x\rho$
S6	21	83	16	48	9	1	$\lvert v_\parallel\rvert$	0.2	0.1	0.1	6	1.4	2.1

TABLE I: Resolution used in this paper for further information the author links to Ref. 2.

In the following the box size is increased relative to the standard box size $(L_x,\ L_y) = (76.3,\ 89.8)\,\rho$ in the radial and binormal direction. Here, x is the radial coordinate that labels the flux surfaces normalized by the thermal Larmor radius ρ and y labels the field lines and is an approximate binormal coordinate. Together with the coordinate s which parameterizes the length along the field lines and is referred to as the parallel coordinate these quantities form the Hamada coordinates[14]. The increased box sizes are indicated by the real parameter N_R for radial and N_B for the binormal direction with the nomenclature $N_\mathrm{R}\times N_\mathrm{B}$ throughout this work. Note that, the number of modes in the respective direction, i.e., N_x and N_m, respectively, is always adapted accordingly to retain a spatial resolution compliant to the standard resolution [Table I] and standard box size.

The $\mathrm{E}\times\mathrm{B}$ staircase pattern is manifest as radial structure formation in the $\mathrm{E}\times\mathrm{B}$ shearing rate defined as[2,15,16]

$$\omega_{\mathrm{E}\times\mathrm{B}} = \frac{1}{2}\frac{\partial^2\langle\phi\rangle}{\partial x^2}\ , \tag{1}$$

where $\langle\phi\rangle$ is the zonal electrostatic potential normalized by $\rho_* T/e$ ($\rho_* = \rho/R$ is the thermal Larmor radius normalized with the major radius R, T is the temperature and e is the elementary charge). The zonal potential is calculated from the electrostatic potential ϕ on the two-dimensional x-y-plane at the low field side according to Ref. 3

$$\langle\phi\rangle = \frac{1}{L_y}\int_0^{L_y} \mathrm{d}y\ \phi(x,y,s=0)\ . \tag{2}$$

The $\mathrm{E}\times\mathrm{B}$ shearing rate $\omega_{\mathrm{E}\times\mathrm{B}}$ is the radial derivative of the advecting zonal flow velocity[17,18] and quantifies the zonal flow induced shearing of turbulent structures[17,19,20].

Consistent with Ref. 2 the turbulence level is quantified by the turbulent heat conduction coefficient χ, which is normalized by $\rho^2 v_\mathrm{th}/R$ ($v_\mathrm{th} = \sqrt{2T/m}$ is the thermal velocity and m is the mass). Furthermore, quantities ρ, R, T, v_th and m are referenced quantities from Ref. 2 and 13.

[a)]Repository of this work:
https://github.com/ManeLippert/Bachelorthesis-Shearingrate-Convergence
[b)]Author to whom correspondence should be addressed:
Florian.Rath@uni-bayreuth.de

In order to diagnose the temporal evolution of the staircase pattern and to obtain an estimate of its amplitude the radial Fourier transform of the $\mathrm{E}\times\mathrm{B}$ shearing rate is considered. It is defined by

$$\omega_{\mathrm{E}\times\mathrm{B}} = \sum_{k_{\mathrm{ZF}}} \widehat{\omega}_{\mathrm{E}\times\mathrm{B}}(k_{\mathrm{ZF}},t)\exp(\mathrm{i}k_{\mathrm{ZF}}x)\ , \tag{3}$$

where $\widehat{\omega}_{\mathrm{E}\times\mathrm{B}}$ is the complex Fourier coefficient and $k_{\mathrm{ZF}} = 2\pi n_{\mathrm{ZF}}/L_x$ defines the zonal flow wave vector with the zonal flow mode number n_{ZF} ranging in $-(N_x-1)/2 \leq n_{\mathrm{ZF}} \leq (N_x-1)/2$. Based on the definitions above, the shear carried by the zonal flow mode with wave vector k_{ZF} is defined by $|\widehat{\omega}_{\mathrm{E}\times\mathrm{B}}|_{n_{\mathrm{ZF}}} = 2|\widehat{\omega}_{\mathrm{E}\times\mathrm{B}}(k_{\mathrm{ZF}},t)|$. In general, the zonal flow mode that dominates the $\mathrm{E}\times\mathrm{B}$ staircase pattern, also referred to as the *basic mode* of the pattern in this work, exhibits the maximum amplitude in the spectrum $|\widehat{\omega}_{\mathrm{E}\times\mathrm{B}}|_{n_{\mathrm{ZF}}}$.

In the first test the radial box size is increased while the binormal box size is kept fixed to the standard size. The scan covers the realizations $N_{\mathrm{R}}\times N_{\mathrm{B}} \in [1\times1,\ 2\times1,\ 3\times1,\ 4\times1]$. Each realization exhibits an initial quasi-stationary turbulent phase and a second final[2] phase with almost suppressed turbulence [Fig. 1(a)]. The latter state is indicative for the presence of a fully developed staircase pattern as depicted in Fig. 2. This type of structure is characterized by intervals of almost constant shear with alternating sign satisfying the Waltz criterion $|\omega_{\mathrm{E}\times\mathrm{B}}| \approx \gamma$[18,21] [$\gamma$ is the growth rate of the most unstable linear ion temperature gradient (ITG)-driven Eigenmode], connected by steep flanks where $\omega_{\mathrm{E}\times\mathrm{B}}$ crosses zero. The negative gradient of the perturbed perpendicular and parallel ion pressure (not shown) exhibit positive corrugations in regions with maximum $|\omega_{\mathrm{E}\times\mathrm{B}}|$ and negative corrugations at zero crossings of $\omega_{\mathrm{E}\times\mathrm{B}}$. A radial force balance analysis suggests that the structures in $\omega_{\mathrm{E}\times\mathrm{B}}$ as depicted in Fig. 2 are not a consequence of the pressure gradient corrugations as discussed elsewhere[22]. Rather, the corrugations in the pressure gradient have to be interpreted as a consequence of the staircase structure in $\omega_{\mathrm{E}\times\mathrm{B}}$ due to the stabilizing zonal flow - turbulence interaction. Figure 2(a) shows a striking repetition of the staircase structure, with the number of repetitions equal to N_{R}. Hence, the basic size of the pattern not only converges with increasing radial box size, but the converged radial size also turns out to at least roughly agree with the standard radial box size of Ref. 2. Due to the lack of a substantial turbulent drive in the final suppressed state no further zonal flow evolution is observed [Fig. 1(b)] and one might critically ask whether the structures shown in Fig. 2 represent the real converged pattern in a statistical sense. Note that in the 3×1 case the initial quasi-stationary turbulent state extends up to a few $\sim 10^4\,R/v_{\mathrm{th}}$. During this period the zonal flow mode with $n_{\mathrm{ZF}} = 3$, i.e., the mode that dominates the staircase pattern in final suppressed phase, undergoes a long-term evolution with a typical timescale of several $\sim 10^3\,R/v_{\mathrm{th}}$. Hence, several of such cycles are covered by the initial turbulent phase, which is evident from the occurrence of phases with reduced amplitude around $t \approx 8000\,R/v_{\mathrm{th}}$ and $t \approx 18000\,R/v_{\mathrm{th}}$. It is the $n_{\mathrm{ZF}} = 4$ zonal flow mode, i.e., the next shorter radial scale mode, that dominates the shear spectrum $|\widehat{\omega}_{\mathrm{E}\times\mathrm{B}}|_{n_{\mathrm{ZF}}}$ in the latter two phases (not shown). This demonstrates a competition between the $n_{\mathrm{ZF}} = 3$ and $n_{\mathrm{ZF}} = 4$ modes. Most importantly, no secular growth of the $n_{\mathrm{ZF}} = 1$ (box scale) zonal flow mode is observed during the entire quasi-stationary turbulent phase [Fig. 1(b) dotted line]. The above discussion indicates that although the $n_{\mathrm{ZF}} = 3,\ 4$ zonal modes compete, the pattern scale does not converge to the radial box scale but rather to a mesoscale of $\sim 57-76\,\rho$ (i.e., $n_{\mathrm{ZF}} = 4,\ 3$ in the 3×1 case).

Since the radially elongated simulation domain might inhibit the development of isotropic turbulent structures, in the second test the radial and binormal box size is increased simultaneously. This scan covers the realizations $N_{\mathrm{R}}\times N_{\mathrm{B}} \in [1\times1,\ 1.5\times1.5,\ 2\times2,\ 2.5\times2.5,\ 3\times3]$. Interestingly, suppression of the turbulence by the emergence of a fully developed staircase pattern almost always occurs after $\sim 1000\,R/v_{\mathrm{th}}$ [Fig. 3], i.e., significantly faster compared to the 3×1 and 4×1 realizations. As shown in Fig. 2(b) this test also confirms the convergence of the staircase pattern size to a typical mesoscale that is distinct from the radial box size in the $N_{\mathrm{R}} > 1$ realizations.

FIG. 1: **(a)** Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for radial increased box sizes,
(b) Time traces of $|\widehat{\omega}_{\mathrm{E}\times\mathrm{B}}|_{n_{\mathrm{ZF}}}$ for radial increased box sizes.

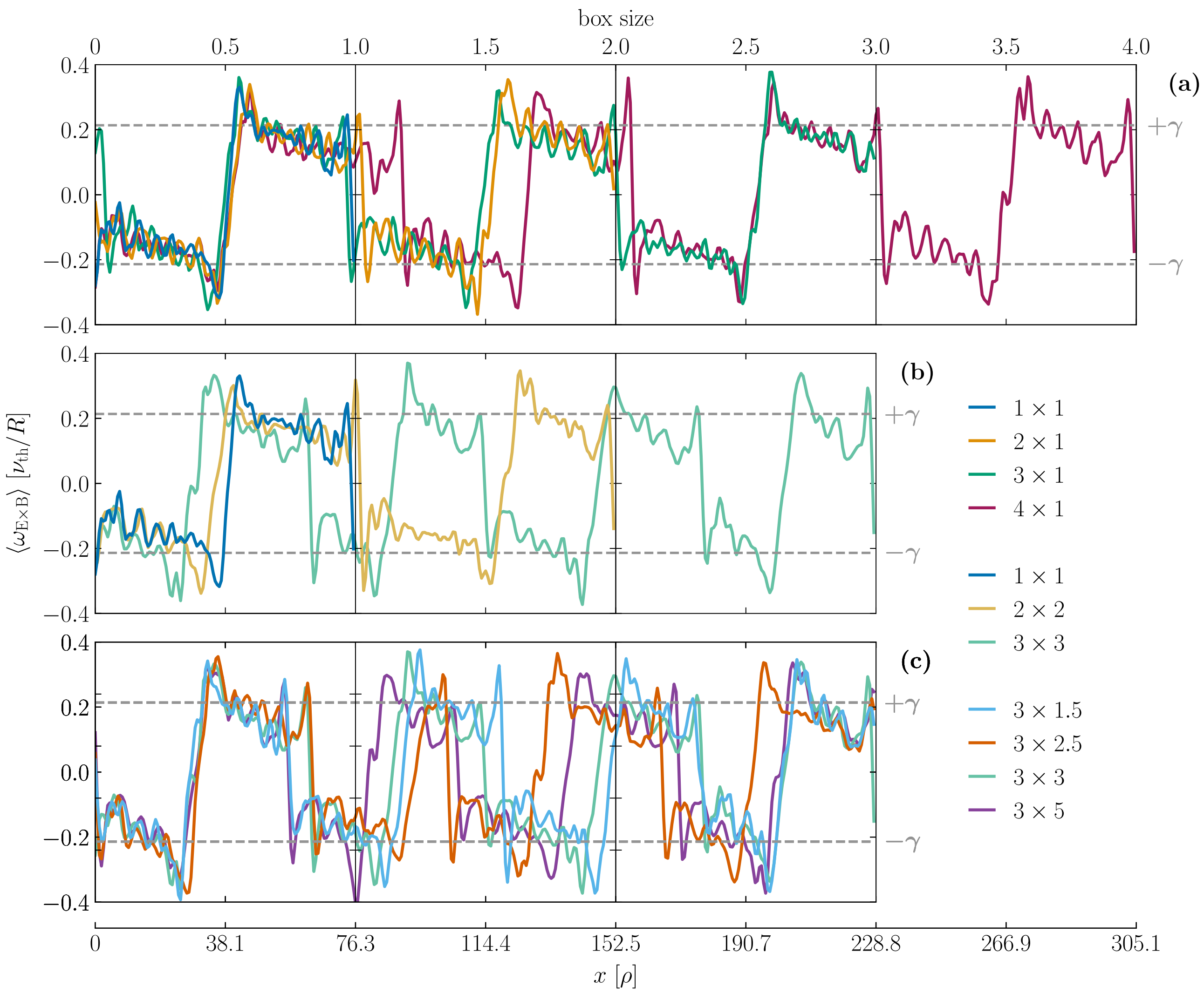


FIG. 2: Comparison of shearing rate $\omega_{E\times B}$ for each box sizes scan averaged over given time interval and the growth rate $\pm\gamma$ of the most unstable linear ITG-driven Eigenmode. The staircase structures are radially shifted with respect to each over till alignment for better visibility:
(a) radial: $t_{1\times1} \in [2000, 5000]$, $t_{2\times1} \in [15000, 18000]$, $t_{3\times1} \in [43000, 45000]$, $t_{4\times1} \in [26000, 28000]$;
(b) isotropic: $t_{1\times1} \in [2000, 5000]$, $t_{2\times2} \in [2000, 3000]$, $t_{3\times3} \in [2000, 3000]$;
(c) binormal: $t_{3\times1.5} \in [2000, 3000]$, $t_{3\times2.5} \in [2000, 3000]$, $t_{3\times3} \in [2000, 3000]$, $t_{3\times5} \in [1000, 3000]$.

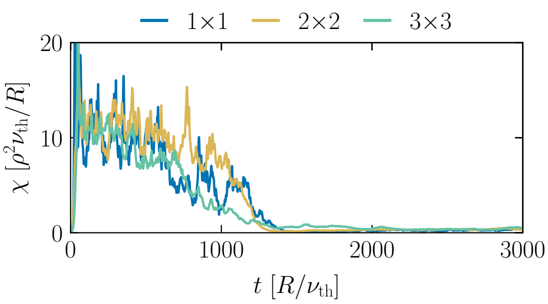


FIG. 3: Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for isotropic increased box sizes.

By contrast to the radial box size scan, the 3×3 realization shows a stationary pattern with four repetitions of the fully developed staircase structure, i.e., a somewhat smaller pattern size. Whether this is related to a possible pattern size dependence on the binormal box size or to the competition between patterns with the two sizes $\lambda \in [57,\ 76]\rho$ as observed in the first test is addressed in the next paragraph. The scale of structures developing in the 1.5×1.5 and 2.5×2.5 realizations (not included in Fig. 2 to preserve the clarity of this figure) also lie within the range given above. Note that two additional realizations of the 3×3 case with different initial conditions and otherwise identical parameters confirm structure formation on scales within the range given above.

In a third test the binormal box size is varied with the radial box size fixed to $N_\mathrm{R}=3$. This test covers the realizations $N_\mathrm{R}\times N_\mathrm{B}\in[3\times1.5,\ 3\times2.5,\ 3\times3,\ 3\times5]$. As in the isotropic scan the turbulence subdued and a fully developed staircase pattern forms after $\sim 2000R/v_\mathrm{th}$ [Fig. 4]. The convergence of staircase pattern can be seen in Fig. 2(c) and confirms again a size of a typical mesoscale. Fig. 2(c) also confirms that indeed a competition between patterns with two sizes $\lambda\in[57,\ 76]\rho$ causing the different results for 3×1 and 3×3. The zonal flow mode number varies between $n_\mathrm{ZF}=3,4$ which can be seen in Fig. 2(c) in the 3×2.5 realization. The staircase structure has a pattern between three and four repetitions which get represented in the second repetition with no significant plateau at positive shear. Instead, the pattern returns immediately after reaching the maximum shear $(+\gamma)$ to the minimum shear $(-\gamma)$ of the third repetition in a steep flank. The Fourier analysis of this case yields no definitely basic mode rather two dominating modes with $n_\mathrm{ZF}=3,4$ with a fraction of the maximum amplitude $|\widehat{\omega}_{\mathrm{E}\times\mathrm{B}}|_{n_\mathrm{ZF}}$ each (not shown).

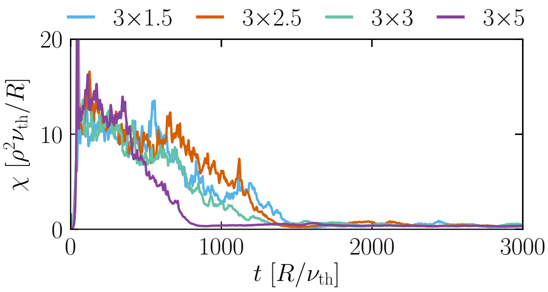


FIG. 4: Time traces of the heat conduction coefficient χ for $R/L_T=6.0$ for binormal increased box sizes.

In the final test the inverse background temperature gradient length R/L_T is varied at fixed 3×3 box size. Since suppression of turbulence usually occurs at later times when approaching the finite heat flux threshold from below[2], the analysis aims to lengthen the phase during which the zonal flow varies in time due to turbulent Reynolds stresses. This scan covers realizations with $R/L_T\in[6.0,\ 6.2,\ 6.4]$. In the case of $R/L_T=6.2$ turbulence suppression is observed for $t>11000R/v_\mathrm{th}$, while stationary turbulence during the entire simulation time trace of $12000R/v_\mathrm{th}$ is found for $R/L_T=6.4$. The finite heat flux threshold, hence, is $R/L_T|_\mathrm{finite}=6.3\pm0.1$ in accordance with Ref. 2. Although the initial quasi-stationary turbulence in the former case is significantly longer compared to the $R/L_T=6.2$ realization discussed in the second test, a stationary pattern with basic zonal flow mode $n_\mathrm{ZF}=3$ establishes. Again, the $n_\mathrm{ZF}=1$ (box scale) zonal flow mode does not grow secularly during the entire turbulent phase. Also, this test confirms the statistical soundness of the converged pattern size of $\sim 57-76\rho$.

Through careful tests this brief communication confirms the radial size convergence of the $\mathrm{E}\times\mathrm{B}$ staircase pattern in local gyrokinetic flux tube simulations of ion temperature gradient (ITG)-driven turbulence. A mesoscale pattern size of $\sim 57-76\rho$ is found to be intrinsic to ITG-driven turbulence for Cyclone Base Case parameters. This length scale is somewhat larger compared to results from global studies with finite ρ_*, which report of a few 10ρ[1], and has to be considered the proper mesoscale in the local limit $\rho_*\to0$. The occurrence of this mesoscale implies that non-locality, in terms of Ref. 1, is inherent to ITG-driven turbulence, since avalanches are spatially organized by the $\mathrm{E}\times\mathrm{B}$ staircase pattern[1,2,6,15].

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

[1] G. Dif-Pradalier, P. H. Diamond, V. Grandgirard, Y. Sarazin, J. Abiteboul, X. Garbet, P. Ghendrih, A. Strugarek, S. Ku, and C. S. Chang, Phys. Rev. E **82**, 025401 (2010).
[2] A. G. Peeters, F. Rath, R. Buchholz, Y. Camenen, J. Candy, F. J. Casson, S. R. Grosshauser, W. A. Hornsby, D. Strintzi, and A. Weikl, Phys. Plasmas **23**, 082517 (2016).
[3] F. Rath, A. G. Peeters, and A. Weikl, Phys. Plasmas **28**, 072305 (2021).
[4] A. Weikl, A. G. Peeters, F. Rath, S. R. Grosshauser, R. Buchholz, W. A. Hornsby, F. Seiferling, and D. Strintzi, Phys. Plasmas **24**, 102317 (2017).
[5] F. Seiferling, A. G. Peeters, S. R. Grosshauser, F. Rath, and A. Weikl, Phys. Plasmas **26**, 102306 (2019).
[6] B. F. McMillan, S. Jolliet, T. M. Tran, L. Villard, A. Bottino, and P. Angelino, Phys. Plasmas **16**, 022310 (2009).
[7] L. Villard, P. Angelino, A. Bottino, S. Brunner, S. Jolliet, B. F. McMillan, T. M. Tran, and T. Vernay, Plasma Physics and Controlled Fusion **55**, 074017 (2013).
[8] J. Seo, H. Jhang, and J.-M. Kwon, Phys. Plasmas **29**, 052502 (2022).
[9] G. Dif-Pradalier, G. Hornung, P. Ghendrih, Y. Sarazin, F. Clairet, L. Vermare, P. H. Diamond, J. Abiteboul, T. Cartier-Michaud, C. Ehrlacher, D. Estève, X. Garbet, V. Grandgirard, O. D. Gürcan, P. Hennequin, Y. Kosuga, G. Latu, P. Maget, P. Morel, C. Norscini, R. Sabot, and A. Storelli, Phys. Rev. Lett. **114**, 085004 (2015).
[10] W. Wang, Y. Kishimoto, K. Imadera, H. Liu, J. Li, M. Yagi, and Z. Wang, Nuclear Fusion **60**, 066010 (2020).
[11] Y. J. Kim, K. Imadera, Y. Kishimoto, and T. S. Hahm, Journal of the Korean Physical Society **81**, 636 (2022).
[12] Y. Kishimoto, K. Imadera, A. Ishizawa, W. Wang, and J. Q. Li, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **381**, 20210231 (2023).
[13] A. G. Peeters, Y. Camenen, F. J. Casson, W. A. Hornsby, A. P. Snodin, D. Strintzi, and G. Szepesi, Comput. Phys. Commun. **180**, 2650 (2009).
[14] S. Hamada, Kakuyugo Kenkyu **1**, 542 (1958).
[15] F. Rath, A. G. Peeters, R. Buchholz, S. R. Grosshauser, P. Migliano, A. Weikl, and D. Strintzi, Phys. Plasmas **23**, 052309 (2016).
[16] M. J. Pueschel, M. Kammerer, and F. Jenko, Phys. Plasmas **15**, 102310 (2008).
[17] T. S. Hahm and K. H. Burrell, Phys. Plasmas **2**, 1648–1651 (1995).
[18] R. E. Waltz, R. L. Dewar, and X. Garbet, Phys. Plasmas **5**, 1784–1792 (1998).
[19] H. Biglari, P. H. Diamond, and P. W. Terry, Phys. Fluids B: Plasma Physics **2**, 1–4 (1990).
[20] K. H. Burrell, Phys. Plasmas **4**, 1499–1518 (1997).
[21] R. E. Waltz, G. D. Kerbel, and J. Milovich, Phys. Plasmas **1**, 2229 (1994).
[22] Y. Kosuga, P. H. Diamond, and O. D. Gürcan, Phys. Rev. Lett. **110**, 105002 (2013).

CHAPTER

7

Bibliography

[1] 2018 nohup. URL https://wiki.ubuntuusers.de/nohup/ – Accessed: 2023-04-15.

[2] 2021 Screen. URL https://wiki.ubuntuusers.de/Screen/ – Accessed: 2023-04-15.

[3] Barton, Justin E., Wehner, William P., Schuster, Eugenio, Felici, Federico & Sauter, Olivier 2015 Simultaneous closed-loop control of the current profile and the electron temperature profile in the tcv tokamak. URL https://ieeexplore.ieee.org/document/7171844.

[4] Beer, M.A. 1994 *Gyrofluid models of turbulent transport in tokamaks*. PhD thesis, Princeton University.

[5] Biglari, H., Diamond, P. H. & Terry, P. W. 1990 Influence of sheared poloidal rotation on edge turbulence. *Phys. Fluids B: Plasma Physics* **2** (1), 1–4.

[6] Brizard, A. J. & Hahm, T. S. 2007 Foundations of nonlinear gyrokinetic theory. *Rev. Mod. Phys.* **79**, 421–468.

[7] Burrell, K. H. 1997 Effects of E×B velocity shear and magnetic shear on turbulence and transport in magnetic confinement devices. *Phys. of Plasmas* **4** (5), 1499–1518.

[8] Cary, John R. 1981 Lie transform perturbation theory for Hamiltonian systems. *Physics Reports* **79** (2), 129–159.

[9] Cary, John R & Littlejohn, Robert G 1983 Noncanonical Hamiltonian mechanics and its application to magnetic field line flow. *Annals of Physics* **151** (1), 1–34.

[10] Casson, F.J. 2011 *Turbulent transport in rotating tokamak plasmas*. PhD thesis, University of Warwick.

[11] Coppi, B., Rosenbluth, M. N. & Sagdeev, R. Z. 1967 Instabilities due to temperature gradients in complex magnetic field configurations. *The Physics of Fluids* **10** (3), 582–587.

[12] Cowley, S. C., Kulsrud, R. M. & Sudan, R. 1991 Considerations of ion-temperature-gradient-driven turbulence. *Physics of Fluids B: Plasma Physics* **3** (10), 2767–2782.

[13] Dannert, T. 2005 *Gyrokinetische Simulation von Plasmaturbulenz mit gefangenen Teilchen und Elektromagnetischen Effekten*. PhD thesis, Technische Universtät München.

[14] Diamond, P. H., Itoh, S.-I., Itoh, K. & Hahm, T. S. 2005 Zonal flows in plasma—a review. *Plasma Phys. Controlled Fusion* **47**, R35.

[15] Diamond, P. H. & Kim, Y.-B. 1991 Theory of mean poloidal flow generation by turbulence. *Physics of Fluids B: Plasma Physics* **3** (7), 1626–1633.

[16] Dif-Pradalier, G., Diamond, P. H., Grandgirard, V., Sarazin, Y., Abiteboul, J., Garbet, X., Ghendrih, Ph., Strugarek, A., Ku, S. & Chang, C. S. 2010 On the validity of the local diffusive paradigm in turbulent plasma transport. *Phys. Rev. E* **82**, 025401.

[17] Dif-Pradalier, G., Hornung, G., Ghendrih, Ph., Sarazin, Y., Clairet, F., Vermare, L., Diamond, P. H., Abiteboul, J., Cartier-Michaud, T., Ehrlacher, C., Estève, D., Garbet, X., Grandgirard, V., Gürcan, Ö. D., Hennequin, P., Kosuga, Y., Latu, G., Maget, P., Morel, P., Norscini, C., Sabot, R. & Storelli, A. 2015 Finding the elusive E×B staircase in magnetized plasmas. *Phys. Rev. Lett.* **114**, 085004.

[18] Dimits, A. M., Bateman, G., Beer, M. A., Cohen, B. I., Dorland, W., Hammett, G. W., Kim, C., Kinsey, J. E., Kotschenreuther, M., Kritz, A. H., Lao, L. L., Mandrekas, J., Nevins, W. M., Parker, S. E., Redd, A. J., Shumaker, D. E., Sydora, R. & Weiland, J. 2000 Comparisons and physics basis of tokamak transport models and turbulence simulations. *Physics of Plasma* **7** (3), 969–983.

[19] Dubin, Daniel H. E., Krommes, John A., Oberman, C. & Lee, W. W. 1983 Nonlinear gyrokinetic equations. *The Physics of Fluids* **26** (12), 3524–3535.

[20] Garbet, X., Idomura, Y., Villard, L. & Watanabe, T. H. 2010 Gyrokinetic simulations of turbulent transport. *Nuclear Fusion* **50** (4).

[21] Hahm, T. S. & Burrell, K. H. 1995 Flow shear induced fluctuation suppression in finite aspect ratio shaped tokamak plasma. *Phys. of Plasmas* **2** (5), 1648–1651.

[22] Hamada, S. 1958 *Kakuyugo Kenkyu* **1**, 542.

[23] Hammett, Greg 2009 The Ion Temperature Gradient (ITG) Instability. CM-PD/CMSO Winter School, UCLA, 1/09/2009.

[24] Hasegawa, Akira & Mima, Kunioki 1978 Pseudo-three-dimensional turbulence in magnetized nonuniform plasma. *The Physics of Fluids* **21** (1), 87–92.

[25] H.Isliker, Pisokas, Th., Strintzi, D. & Vlahos, L. 2010 A self-organized criticality model for ion temperature gradient mode driven turbulence in confined plasma. *Phys. of Plasmas* **17**.

[26] Horton, W. 1999 Drift waves and transport. *Rev. Mod. Phys.* **71**, 735–778.

[27] Idomura, Y., Urano, H., Aiba, N. & Tokuda, S. 2009 Study of ion turbulent transport and profile formations using global gyrokinetic full- f vlasov simulation. *Nuclear Fusion* **49** (6), 065029.

[28] Kim, Y. J., Imadera, K., Kishimoto, Y. & Hahm, T. S. 2022 Transport events and E×B staircase in flux-driven gyrokinetic simulation of ion temperature gradient turbulence. *Journal of the Korean Physical Society* **81**, 636.

[29] Kishimoto, Y., Imadera, K., Ishizawa, A., Wang, W. & Li, J. Q. 2023 Characteristics of constrained turbulent transport in flux-driven toroidal plasmas. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **381** (2242), 20210231.

[30] Kosuga, Y., Diamond, P. H. & Gürcan, Ö. D. 2013 How the propagation of heat-flux modulations triggers $e{\times}b$ flow pattern formation. *Phys. Rev. Lett.* **110**, 105002.

[31] Krommes, John A. 2012 The Gyrokinetic Description of Microturbulence in Magnetized Plasmas. *Annual Review of Fluid Mechanics* .

[32] Krommes, John A. & Kim, Chang-Bae 2000 Interactions of disparate scales in drift-wave turbulence. *Phys. Rev. E* **62**, 8508–8539.

[33] Lippert, M. 2022 torque_monitor.py. URL `https://github.com/ManeLippert/Bachelorthesis-Shearingrate-Convergence/blob/main/python/torque_monitor.py` – Accessed: 2023-04-14.

[34] Lippert, M. & Rath, F. 2023 slurm_monitor.py. URL https://bitbucket.org/gkw/gkw/src/develop/python/slurm_monitor.py – Accessed: 2023-04-12.

[35] Lippert, M., Rath, F. & Peeters, A. G. 2023 Size convergence of the E×B staircase pattern in flux tube simulations of ion temperature gradient driven turbulence. *Physics of Plasma* **30** (7), 969–983.

[36] Maeyama, S., Ishizawa, A., Watanabe, T.-H., Nakata, M., Miyato, N., Yagi, M. & Idomura, Y. 2014 Comparison between kinetic-ballooning-mode-driven turbulence and ion-temperature-gradient-driven turbulence. *Phys. of Plasmas* **21** (5), 052301.

[37] Makwana, K. D., Terry, P. W., Pueschel, M. J. & Hatch, D. R. 2014 Subdominant modes in zonal-flow-regulated turbulence. *Phys. Rev. Lett.* **112**, 095002.

[38] McMillan, B. F., Jolliet, S., Tran, T. M., Villard, L., Bottino, A. & Angelino, P. 2009 Avalanchelike bursts in global gyrokinetic simulations. *Phys. of Plasmas* **16** (2), 022310.

[39] Mittendorf, J., Schobert, B. & Müller, D. 2023 Rmhd-code. URL https://bitbucket.org/astro_bayreuth/rmhdcode – Accessed: 2023-04-14.

[40] Müller, D. 2023 Numerical simulations of exor events in protoplanetary disks: Numerical stability and growth of ring structures in the surface density. Bachelorthesis, University of Bayreuth.

[41] Nakata, M., Watanabe, T.-H. & Sugama, H. 2012 Nonlinear entropy transfer via zonal flows in gyrokinetic plasma turbulence. *Phys. of Plasmas* **19**, 022303.

[42] Peeters, A. G., Camenen, Y., Casson, F. J., Hornsby, W. A., Snodin, A. P., Strintzi, D. & Szepesi, G. 2009 The nonlinear gyro-kinetic flux tube code gkw. *Comput. Phys. Commun.* **180**, 2650.

[43] Peeters, A. G., Rath, F., Buchholz, R., Camenen, Y., Candy, J., Casson, F. J., Grosshauser, S. R., Hornsby, W. A., Strintzi, D. & Weikl, A. 2016 Gradient-driven flux-tube simulations of ion temperature gradient turbulence close to the non-linear threshold. *Phys. of Plasmas* **23** (8), 082517.

[44] Pueschel, M. J., Kammerer, M. & Jenko, F. 2008 Gyrokinetic turbulence simulations at high plasma beta. *Phys. of Plasmas* **15** (10), 102310.

[45] Rath, F., Peeters, A. G., Buchholz, R., Grosshauser, S. R., Migliano, P., Weikl, A. & Strintzi, D. 2016 Comparison of gradient and flux driven gyro-kinetic turbulent transport. *Phys. of Plasmas* **23** (5), 052309.

[46] Rath, F., Peeters, A. G. & Weikl, A. 2021 Analysis of zonal flow pattern formation and the modification of staircase states by electron dynamics in gyrokinetic near marginal turbulence. *Phys. of Plasmas* **28** (7), 072305.

[47] Rudakov, L.I. & Sagdeev, R.Z. 1961 On the instability of a nonuniform rarefied plasma in a strong magnetic field. *Dokl. Akad. Nauk. SSSR* **138** (3), 581–583.

[48] Schelter, Dr.rer.nat. Ingo 2016 btrzx2 (2016). URL `https://www.bzhpc.uni-bayreuth.de/de/keylab/Cluster/btrzx2_page/index.html` – Accessed: 2023-04-14.

[49] Schelter, Dr.rer.nat. Ingo 2020 btrzx1 (2020). URL `https://www.bzhpc.uni-bayreuth.de/de/keylab/Cluster/btrzx1_page/index.html` – Accessed: 2023-04-12.

[50] Seiferling, F., Peeters, A. G., Grosshauser, S. R., Rath, F. & Weikl, A. 2019 The interplay of an external torque and e×b structure formation in tokamak plasmas. *Phys. of Plasmas* **26** (10), 102306.

[51] Seo, Janghoon, Jhang, Hogun & Kwon, Jae-Min 2022 Effects of light impurities on zonal flow activities and turbulent thermal transport. *Phys. of Plasmas* **29** (5), 052502.

[52] Stroth, Ulrich 2011 *Plasmaphysik*. Wiesbaden: Viewg+Teubner.

[53] Villard, L, Angelino, P, Bottino, A, Brunner, S, Jolliet, S, McMillan, B F, Tran, T M & Vernay, T 2013 Global gyrokinetic ion temperature gradient turbulence simulations of iter. *Plasma Physics and Controlled Fusion* **55** (7), 074017.

[54] Waltz, R. E., Dewar, R. L. & Garbet, X. 1998 Theory and simulation of rotational shear stabilization of turbulence. *Phys. of Plasmas* **5** (5), 1784–1792.

[55] Waltz, R. E., Kerbel, G. D. & Milovich, J. 1994 Toroidal gyro-landau fluid modelturbulence simulations in a nonlinearballooning mode representation with radial modes. *Phys. of Plasmas* **1**, 2229.

[56] Wang, W., Kishimoto, Y., Imadera, K., Liu, H.R., Li, J.Q., Yagi, M. & Wang, Z.X. 2020 Statistical study for itg turbulent transport in flux-driven tokamak plasmas based on global gyro-kinetic simulation. *Nuclear Fusion* **60** (6), 066010.

[57] Weikl, A., Peeters, A. G., Rath, F., Grosshauser, S. R., Buchholz, R., Hornsby, W. A., Seiferling, F. & Strintzi, D. 2017 Ion temperature gradient turbulence close to the finite heat flux threshold. *Phys. of Plasmas* **24** (10), 102317.

[58] Wesson, John 2011 *Tokamaks*. Oxford: Oxford University Press.

[59] Whelan, G. G., Pueschel, M. J. & Terry, P. W. 2018 Nonlinear electromagnetic stabilization of plasma microturbulence. *Phys. Rev. Lett.* **120**, 175002.

[60] Whelan, G. G., Pueschel, M. J., Terry, P. W., Citrin, J., McKinney, I. J., Guttenfelder, W. & Doerk, H. 2019 Saturation and nonlinear electromagnetic stabilization of itg turbulence. *Phys. of Plasmas* **26** (8), 082302.

[61] W.M.Newins, J.Candy, S.Cowley, T.Dannert, A.Dimits, W.Dorland, C.Estrada-Mila, G.W.Hammet, F.Jenko, M.J.Pueschel & D.E.Shumaker 2006 Characterizing electron temperature gradient turbulence via numerical simulations. *Phys. of Plasmas* **13**.

Eidesstattliche Erklärung

Hiermit erkläre ich, Manuel Lippert, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen,

die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit hat in gleicher oder

ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Bayreuth, den 30.06.2023

Manuel Lippert