Theoretical Plasmaphysics

Bachelor Thesis

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

Manuel Lippert

CHAPTER

Information

Day	April 18, 2023
Place	Universität Bayreuth
Supervisor	Professor Arthur Peeters, Florian Rath

Author	Manuel Lippert (Manuel.Lippert@uni-bayreuth.de)

CHAPTER

Abstract

The radial size convergence of the $\mathrm{E} \times \mathrm{B}$ staircase pattern is adressed in local gradient-driven flux tube simulations of ion temperature gradient (ITG) driven turbulence. Its is shown that a mesoscale pattern size of $\sim 57.20 - 76.27\,\rho$ is inherent to ITG driven turbulence with Cyclone Base Case parameters in the local limit.

Zusammenfassung

CHAPTER

Dedication

CHAPTER

Contents

CHAPTER

1 Introduction

Ion temperature gradient driven turbulence close to marginal stability exhibits zonal flow pattern formation on mesoscales, so-called $E \times B$ staircase structures[5]. Such pattern formation has been observed in local gradient-driven flux-tube simulations[18,29,21] as well as global gradient-driven[14,25,24] and global flux-driven[5,6,28,9,10] 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[5].

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. 18.

CHAPTER

2 Theory

In the following the box size is increased relative to the standard box size $(L_x,\ L_y) = (76.27,\ 89.76)\,\rho$ in the radial and binormal direction. Here, x is the radial coordinate that labels the flux surfaces normalized by the thermal Larmor radius ρ, 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[8]. 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 [Tab. 4.1] 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 by[20,19,18]

$$\omega_{\mathrm{E}\times\mathrm{B}} = \frac{1}{2}\frac{\partial^2 \langle\phi\rangle}{\partial x^2}, \tag{2.1}$$

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, 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[21]

$$\langle\phi\rangle = \frac{1}{L_y}\int_0^{L_y} \mathrm{d}y\ \phi(x, y, s = 0). \tag{2.2}$$

The E×B shearing rate $\omega_{\mathrm{E}\times\mathrm{B}}$ is the radial derivative of the advecting zonal flow velocity[7,26] and quantifies the zonal flow induced shearing of turbulent structures[3,7,4].

Consistent with Ref. 18 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. 18,17.

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

$$\omega_{\mathrm{E\times B}} = \sum_{k_{\mathrm{ZF}}} \widehat{\omega}_{\mathrm{E\times B}}(k_{\mathrm{ZF}}, t) \exp(\mathrm{i} k_{\mathrm{ZF}} x), \tag{2.3}$$

where $\widehat{\omega}_{\mathrm{E\times B}}$ 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 $|\widehat{\omega}_{\mathrm{E\times B}}|_{n_{\mathrm{ZF}}} = 2|\widehat{\omega}_{\mathrm{E\times B}}(k_{\mathrm{ZF}}, t)|$. 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 $|\widehat{\omega}_{\mathrm{E\times B}}|_{n_{\mathrm{ZF}}}$.

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)[17] with adiabatic electron approximation. In agreement with Ref. 18, 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 Tab. 4.1 and for more details about the definition of individual quantities the reader is referred to Refs. 17,18.

	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 paper: 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 $k_x\rho$

3.2 btrzx1 Cluster

The simulation itself will be performed on the `btrzx1`-cluster of the University Bayreuth. This cluster has a wall time from 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`.[23]

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

3.3 Restart Script for Simulation

As stated in Chapter 3.2 the `btrzx1`-cluster has a wall time of 24 hours but since simulations typical run longer than 24 hours one have to restart the simulation after the wall time gets exceeded. Some simulation need even longer so that the simulation itself needs multiple weeks to finish with a result which makes the restart of the simulation a daily task that could be easily forget. So to eliminate this minor inconveniences the following restart script were written.

As script language `python` was chosen over shell because of its easiness to write code and the variety of tools which can interact with the `bash`-shell. `bash` was also considered but after some issue occurred regarding the mechanism to check wether a file exist or not, `python` was selected.

The script was named `slurm_monitor.py` to indicate that this script works with the the `Slurm` resource manager. The source code was pushed to the gkw repository on BitBucket[12] or can be found in Appendix 6.1

3.3.1 How to run Restart Script in Terminal

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

```
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 3.3.2. 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 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:

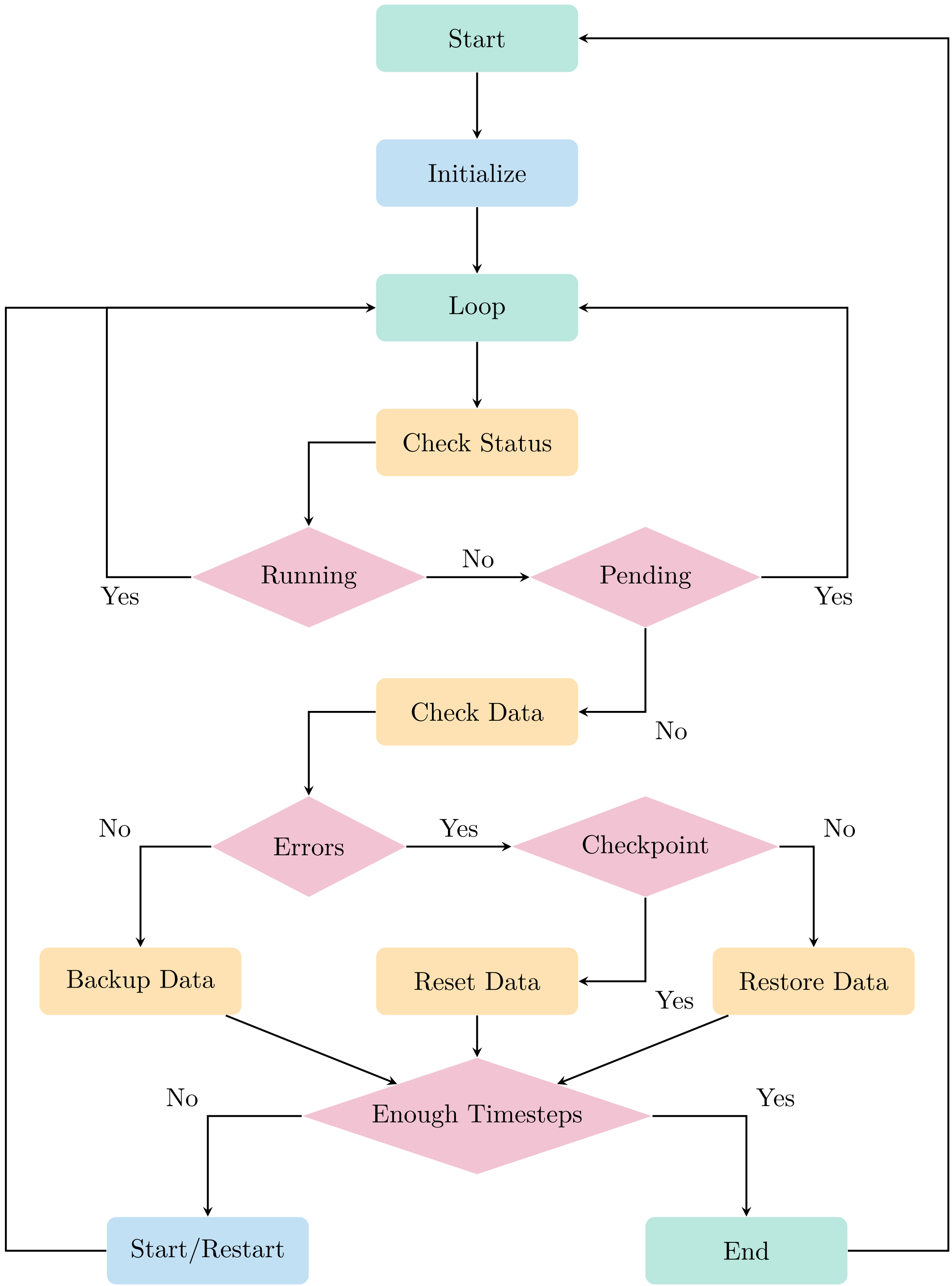
   ```
   screen -r $SESSION
   ```

3.3.2 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:

- The script has no requirements because it runs on the standard `python3` library
- **!Required!** Set job name not longer than 8 characters to identify simulation
  `-j [JOBNAME], --job-name [JOBNAME]`
- Creates jobscript file with defined content
  (look into variable `jobscriptContent` to add more option)
  `--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
  (uses `rsync` command line utility)
  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)

3.3.3 General Structure



3.3.4 Status File created from Restart Script

As stated in Chapter 3.3.2 the restart script is writing a status file with the current status and progress bars of the simulation. The output is formatted like a table and can bei adjust in its appearance with the `--format` parser option [Chapter 3.3.2]. The output types which are written in the status file are listed in Tab. 3.2. To each output prints the restart script the date, time and the duration of the monitoring additionally into the table.

Type	Information
`STARTING`	Start of monitoring process or Start/Restart of simulation
`CONTINUE`	If the simulation has performed at least one run
`CONTROL`	Check of the current time steps of the simulation
`SUCCESS`	Stop of 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 the backup of the data to the predefined location
`RESTORE`	The script reloads the data from the backup folder if errors occur
`RESET`	The simulation gets reset with the use of dump files
`ERROR`	During the monitoring does occur an error 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 writes down the progress of simulation to the end of the status file in form of a progress bar which gets updated dynamically. 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 ist 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 has the advantage that someone can easily access all important information at the glance. To boost productivity the following command could be quite handy:

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

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

3.3.5 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[22] and supports the research of Dominik Müller for his Bachelor Thesis[16] with the `RHMD`-code[15]. The script was named `torque_monitor.py` and can be found in the GitHub Repository of this thesis[11] 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

Results and Discussion 4

The performed simulation for this chapter are documented in Tab. 6.1 with the parameter set, in Tab. 6.2 with the time steps and zonal flow mode number n_{ZF} for each simulation and in Tab. 6.3 with the data location in the GitHub Repository/Folder of the 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 fearing numerical dissipation. Numerical dissipatation can therefore result to no formation of zonal flow stuctures, which cause an 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 knowm minimum number of grid points to verify result in general.

4.1.1 Benchmark

Starting from the “Standard resolution with 6th order (S6)” [Tab. 3.1] the first step is to reproduce the result of Ref. 18 in Section IV. Note that, because of selected circular cencentric 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. 18 for s-α geometry. In Fig. 4.1 the obtained data is simular to the results from Ref. 18 with subdued turbulence after $\sim 3000R/v_{\mathrm{th}}$.

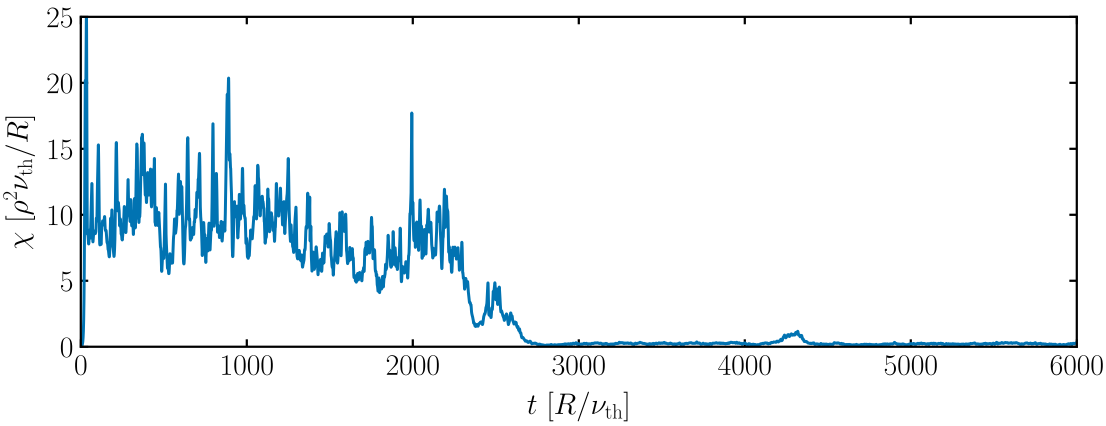


Figure 4.1: Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for benchmark

As next step an approach to the finite heat flux threshold were made to verify the selection of the gradient length R/L_T. As in Ref. 18 in Section V conclude is the finite heat flux threshold approximately located at a grendent length of $R/L_T = 6.3$ for circular geometry [FIG. 4 of Ref. 18]. 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 occur in the whole time domain, which is in agreement with Ref. 18.

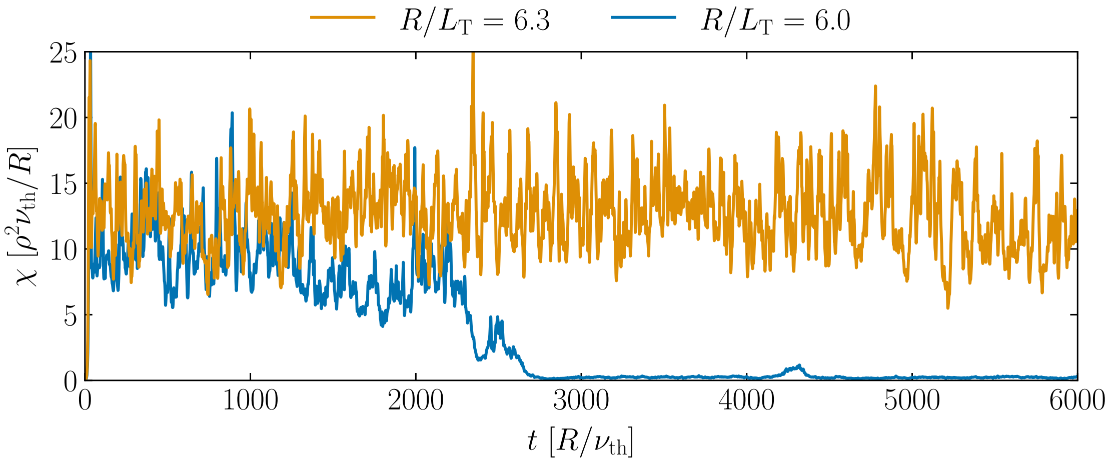


Figure 4.2: Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ and $R/L_T = 6.3$ for benchmark

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 visable 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 as well not acceptable according to criteria **(2)** since the surpressed turbulence state gain instability after $\sim 3000 R/v_{\text{th}}$.
So to conclude only grid points with $N_{\nu_\parallel} = 48, 64$ are satisfying the set criteria. Due to criteria **(1)** the selected resolution will be:

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

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

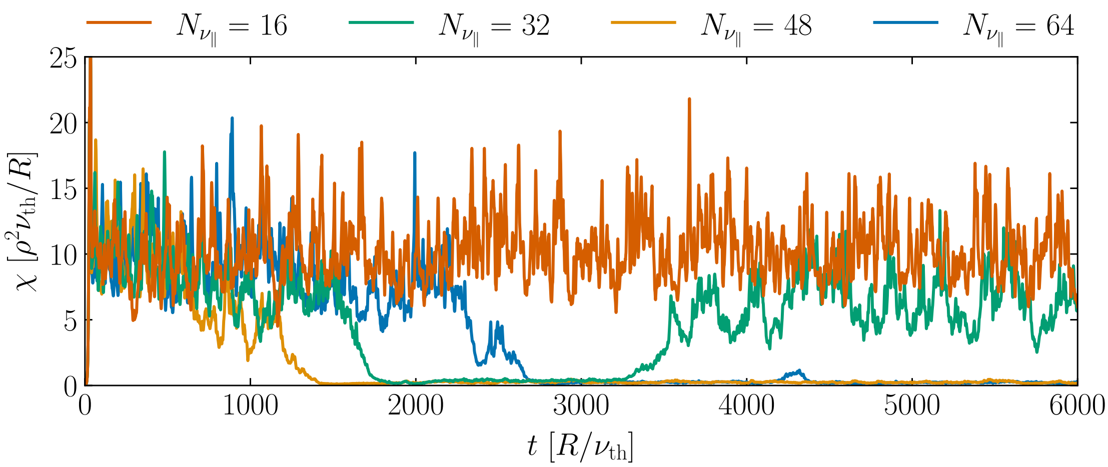


Figure 4.3: Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for reduced parallel velocity grid points $N_{\nu_\parallel}$

4.1.3 Reduction of magnetic moment grid points N_μ

As 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 $\sim 3000 R/v_{\mathrm{th}}$.

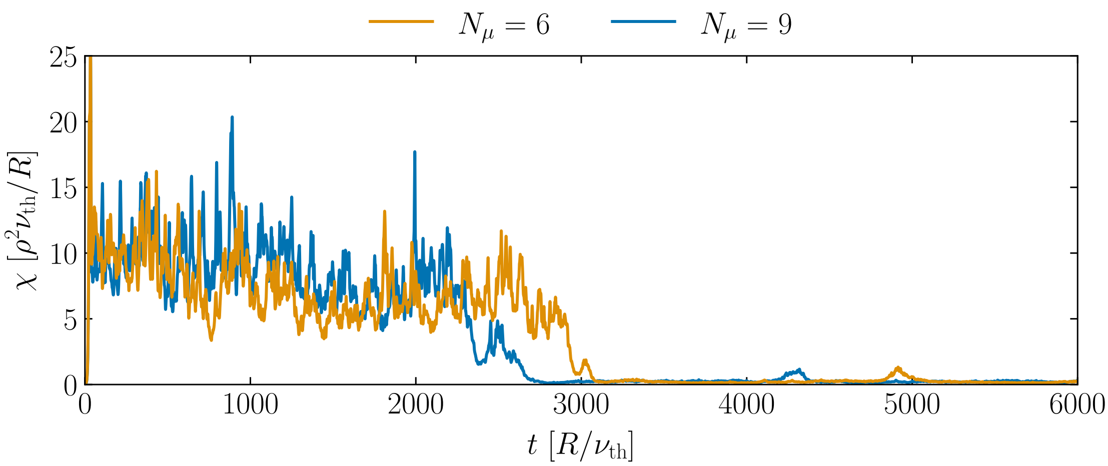


Figure 4.4: Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for reduced magnetic moment grid points N_μ

To conclude a curial result the number of grids point 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 the both criteria in mind, to the following resolution:

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

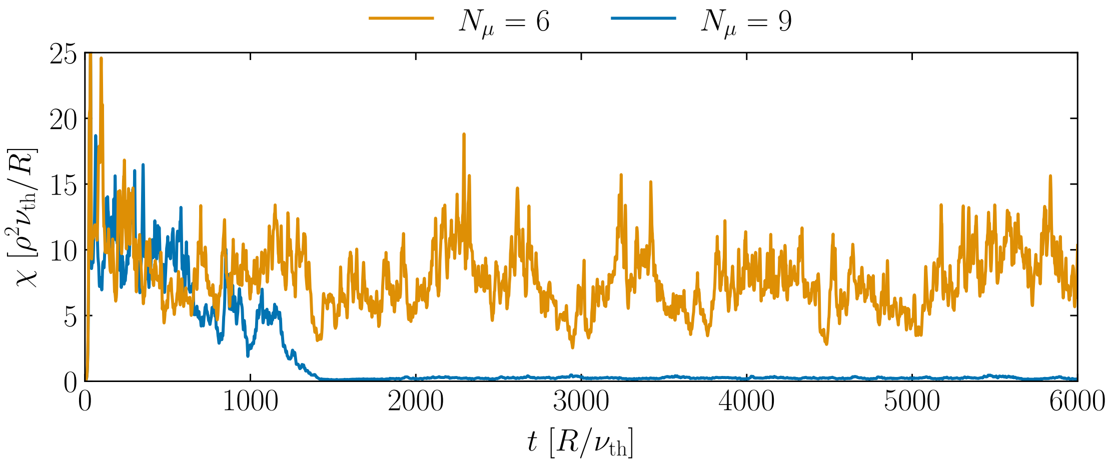


Figure 4.5: Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ and $N_{\nu_\parallel} = 48$ for reduced magnetic moment grid points N_μ

4.1.4 Reduction of magnetic field grid points N_s

In the final step the number of grid points for the magnetic field N_s get reduced with the following parameters:

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

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

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

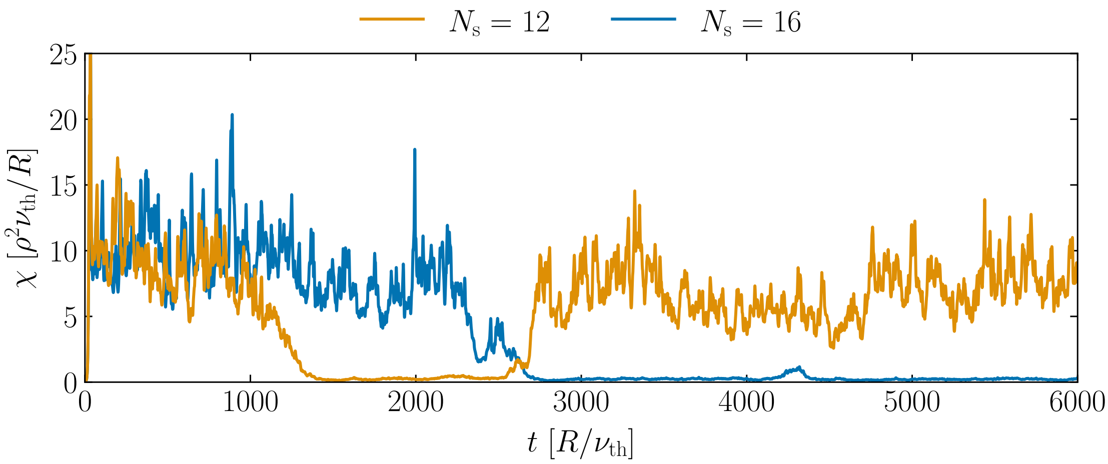


Figure 4.6: Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for reduced magnetic field grid points N_s

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 are displayed in Tab. 4.1 with reduced number of grid points for the parallel velocity $N_{\nu_\parallel}$.

	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	48	9	1	$\lvert\nu_\parallel\rvert$	0.2	0.1	0.1	6	1.4	2.1

Table 4.1: Resolution used in this paper: 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 $k_x\rho$

4.2 Size Convergence of $\mathrm{E}\times\mathrm{B}$ staircase pattern

This chapter is an further iteration of the brief communication published in “Physics of Plasma”. It provides additional plots and paragraphs that were not necessary for the publication. the format was changed as well because the publication was written in REVTEX 4.1 and this bachelor thesis in `scrreprt`. The brief communication can be found in Appendix 6.5 or under Ref. 13.

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[18] 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. 4.10. This type of structure is characterized by intervals of almost constant shear with alternating sign satisfying the Waltz criterion $|\omega_{\mathrm{E\times B}}| \approx \gamma$[27,26] ($\gamma$ is the growth rate of the most unstable linear ITG driven Eigenmode), connected by steep flanks where $\omega_{\mathrm{E\times B}}$ crosses zero. Fig. 4.10 (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, the converged radial size turns out to at least roughly agree with the standard radial box size of Ref. 18.

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. 4.10 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 time scale 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 $\sim 57.20 - 76.27\,\rho$ (i.e., $n_{\mathrm{ZF}} = 4,\ 3$ in the 3×1 case).

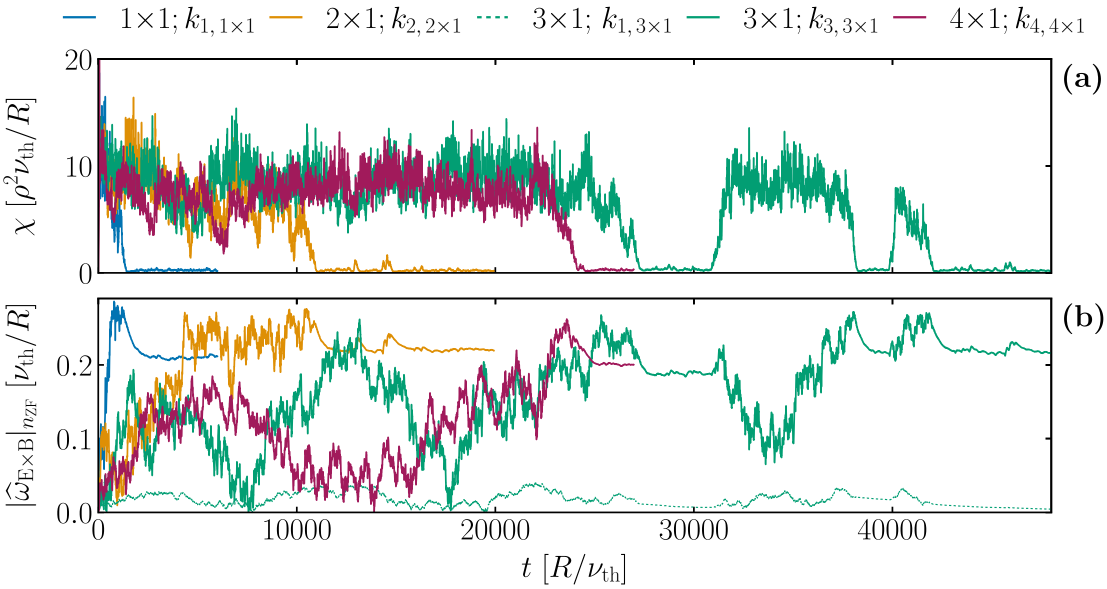


Figure 4.7: **(a)** Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for radial increased box sizes
(b) Time traces of $|\widehat{\omega}_{\mathrm{E\times B}}|_{n_{\mathrm{ZF}}}$ for radial increased box sizes

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:

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

Interestingly, suppression of the turbulence by the emergence of a fully developed staircase pattern always occurs after $\sim 1000\ R/v_{\mathrm{th}}$ [Fig. 4.8 (a)], i.e., significantly faster compared to the 3×1 and 4×1 realizations. As shown in Fig. 4.10 (b) also this test 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.

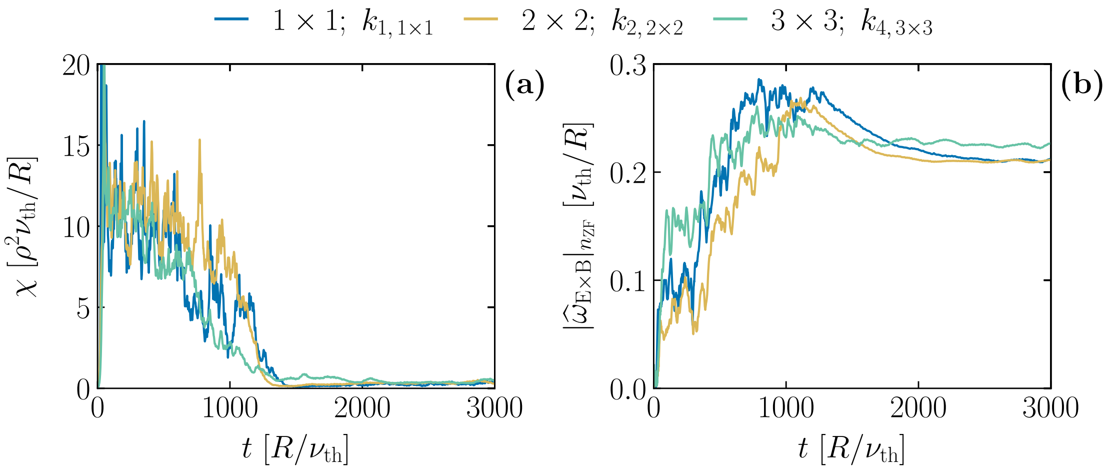


Figure 4.8: **(a)** Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for isotropic increased box sizes
(b) Time traces of $|\widehat{\omega}_{\mathrm{E\times B}}|_{n_{\mathrm{ZF}}}$ for isotropic increased box sizes

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. 4.8 (b), Fig. 4.10 (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 $\lambda \in [57.20,\ 76.27]\,\rho$ as observed in the first test is addressed in the next paragraph.

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. 4.9 (a)]. The convergence of staircase pattern can be seen in Fig. 4.10 (c) and confirms again a size of a typical mesoscale. Fig. 4.10 (c) also confirms that indeed a competition between patterns with two sizes $\lambda \in [57.20,\ 76.27]\, \rho$ 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. 4.10 (c) in the 3×2.5 realization. The staircase structure has a pattern between 3 and 4 repetitions which get represented in the second repetition with no signifciant 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. 4.9 (b)].

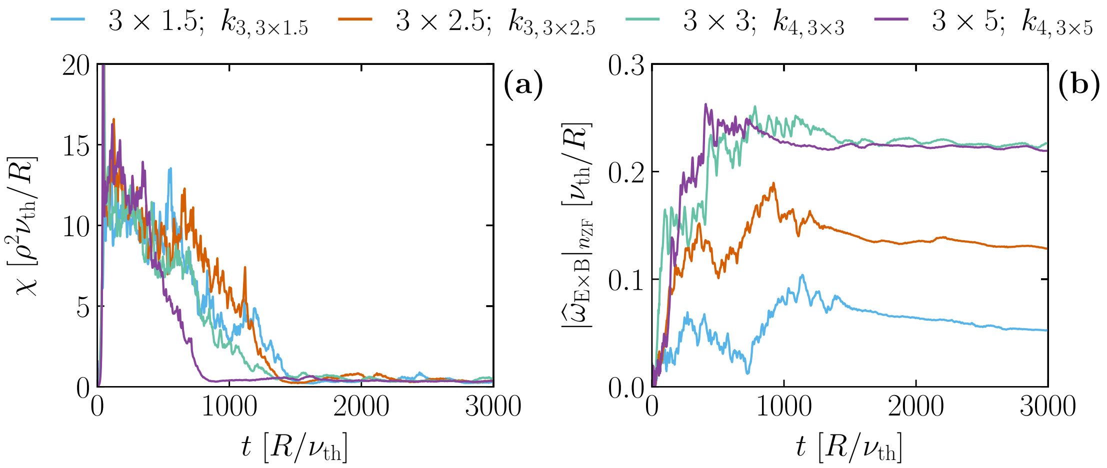


Figure 4.9: **(a)** Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for binormal increased box sizes
(b) Time traces of $|\widehat{\omega}_{\mathrm{E\times B}}|_{n_{\mathrm{ZF}}}$ for binormal increased box sizes

4.2.4 Staircase structures in Comparison

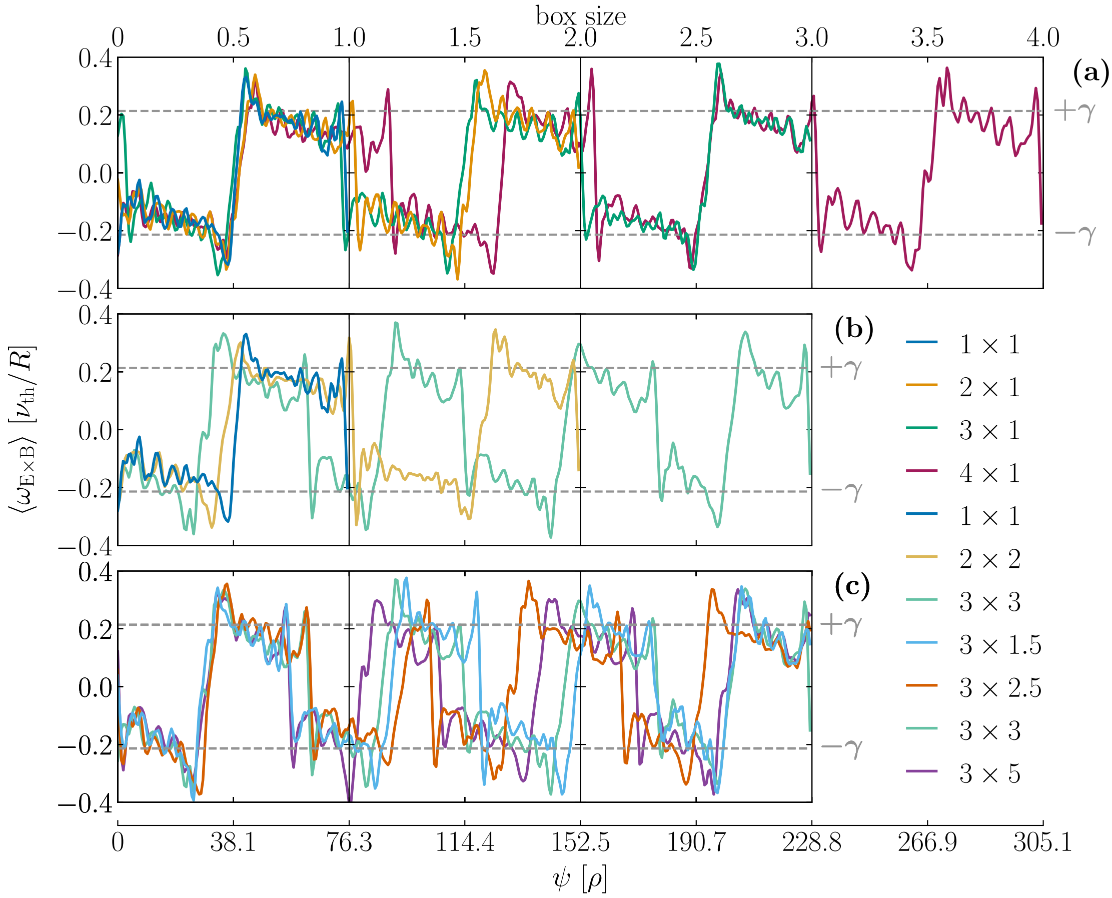


Figure 4.10: Comparison of shearing rate $\omega_{\mathrm{E}\times\mathrm{B}}$ for each box sizes scan averaged over given time interval and the growth rate $\pm\gamma$ of the most unstable linear ITG driven Eigenmode. The staircase structures are radially shifted with respect to each over till alignment 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:	$t_{3\times1.5}$	$\in [2000, 3000]$,	$t_{3\times2.5}$	$\in [2000, 3000]$,
	$t_{3\times3}$	$\in [2000, 3000]$	$t_{3\times5}$	$\in [1000, 3000]$

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[18], 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 $t > 11000\, R/v_{\text{th}}$, while stationary turbulence during the entire simulation time trace of $12000\, R/v_{\text{th}}$ is found for $R/L_T = 6.4$ [Fig. 4.11]. The finite heat flux threshold, hence, is:

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

in accordance to Ref. 18. 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 $n_{\text{ZF}} = 3$ establishes. Again, the $n_{\text{ZF}} = 1$ (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 $\sim 57.20 - 76.27\, \rho$.

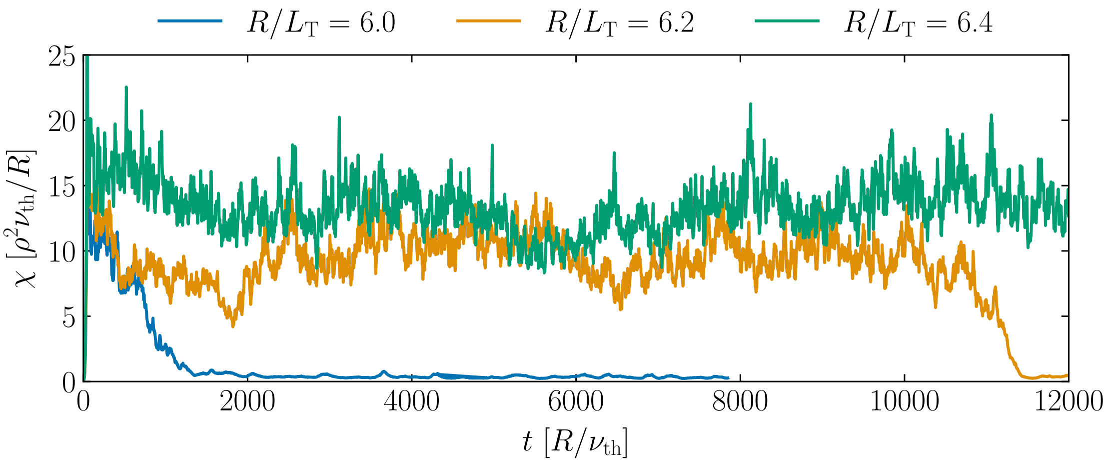


Figure 4.11: Time traces of the heat conduction coefficient χ for different gradient lengths R/L_T

CHAPTER

5 Closure

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 $\sim 57.20 - 76.27\,\rho$ 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 $10\,\rho$[5], 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.[5], is inherent to ITG driven turbulence, since avalanches are spatially organized by the $\mathrm{E} \times \mathrm{B}$ staircase pattern[14,5,20,18].

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("--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()

# 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", "")

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

emailAddress = args.mail
RESTARTMAIL = args.restartmail

backupLocation = args.backup

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
#statusFile = open(statusFilename, "r+")
```

```
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

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

## 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 ./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()
folder = os.getcwd()
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

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

commandJobStarting = "sbatch"

commandBackup  = ["rsync", "-a", "", backupPath]
commandRestore = ["rsync", "-a", "-I", "--exclude=" + statusFilename,
    ↪ backupPath + "/", ""]
```

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

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

def progressbar(required_value, current_value, barsize=42,
                prefix="",
                progress_fill="=", progress_fill_top="",
    ↪ progress_fill_bot="",
                progress_unfill=".",
                progress_bracket=["[","]"]):

    x = int(barsize*current_value/required_value)
    percent = int(100*current_value/required_value)

    try:
        percent_format = "  (" + (int(math.log10(100)) -
    ↪ int(math.log10(percent)))*" " + "{}%)"
    except ValueError:
        percent_format = "  (" + int(math.log10(100))*" " + "{}%)"

    try:
        ratio_format = "  " + (int(math.log10(required_value)) -
    ↪ int(math.log10(current_value)))*" " + "{}/{}"
    except ValueError:
        ratio_format = "  " + int(math.log10(required_value))*" " + "{}/{}"


    bar_format =  "{}  " + progress_bracket[0] + progress_fill_bot + "{}" +
    ↪ progress_fill_top + "{}" + progress_bracket[1] + percent_format +
    ↪ ratio_format
    bar = bar_format.format(prefix, progress_fill*x,
    ↪ progress_unfill*(barsize-x), percent, current_value, required_value)

    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_conter = 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,
    ↪ 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])

try:
    progressbartime_content = [progressbar(required_time, current_time,
↪ progress_fill_top=">",
                                           prefix="RUN " +
↪ str(run_conter) + (2 - int(math.log10(run_conter)))*" " + " ")]
except ValueError:
    progressbartime_content = [progressbar(required_time, current_time,
↪ progress_fill_top=">",
                                           prefix="RUN " +
↪ str(run_conter) + 2*" " + " ")]
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(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(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(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(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_checkpoint.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', 'status.txt']
    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(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(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('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(HDF5_FILENAME)):

    # Find all possible keys items, i.e. both groups and datasets
    f = h5py.File(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(SIM_DIR+'/'+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(filename)):
            os.remove(filename)


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

    if(os.path.isfile(check1Filename) and os.path.isfile(check1Bin)):
        DM1 = True
    if(os.path.isfile(check2Filename) and os.path.isfile(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(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(filename, "r") as file:
        line = file.readlines()[line_index]

        content = line.split(" ")
        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_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)

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

PID = subprocess.getoutput(commandMonitorKill).split("\n")[0]
if kill:
    subprocess.run(["kill", PID])
    quit()

# 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(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(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(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(dataFilename)
                    #f = h5py.File(dataFilename)
                    f.close()

                    # Check if FDS/FDS.dat is updated after run and has
↪ equially time stamp as gkwdata.h5
                    timestamp_data    =
↪ int(os.path.getmtime(dataFilename))
                    timestamp_restart =
↪ int(os.path.getmtime(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,
↪ 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 |
+----------------------------------------------------------------------------+
| STARTING  Start monitoring 3x1.5      2023-01-23  09:58:18  0:00:00:00:00 |
| BACKUP    /scratch/bt712347/backup/   2023-01-23  09:58:18  0:00:00:00:00 |
+----------------------------------------------------------------------------+
| STARTING  Submitted batch job 903536  2023-01-23  09:58:20  0:00:00:00:01 |
| RUNNING   Job is executed             2023-01-23  09:58:50  0:00:00:00:31 |
| BACKUP    /scratch/bt712347/backup/   2023-01-24  09:38:08  0:00:23:39:49 |
| CONTROL   Current Timesteps 2086      2023-01-24  09:38:49  0:00:23:40:30 |
+----------------------------------------------------------------------------+
| STARTING  Submitted batch job 905108  2023-01-24  09:38:49  0:00:23:40:30 |
| RUNNING   Job is executed             2023-01-24  09:39:19  0:00:23:41:01 |
| BACKUP    /scratch/bt712347/backup/   2023-01-25  09:18:41  0:01:23:20:22 |
| CONTROL   Current Timesteps 4179      2023-01-25  09:19:35  0:01:23:21:16 |
+----------------------------------------------------------------------------+
| STARTING  Submitted batch job 906542  2023-01-25  09:19:35  0:01:23:21:16 |
| RUNNING   Job is executed             2023-01-25  09:20:05  0:01:23:21:46 |
| BACKUP    /scratch/bt712347/backup/   2023-01-26  08:59:21  0:02:23:01:03 |
| CONTROL   Current Timesteps 6289      2023-01-26  09:00:38  0:02:23:02:19 |
+----------------------------------------------------------------------------+
| STARTING  Submitted batch job 907237  2023-01-26  09:00:38  0:02:23:02:19 |
| RUNNING   Job is executed             2023-01-26  09:01:08  0:02:23:02:49 |
| BACKUP    /scratch/bt712347/backup/   2023-01-27  08:40:56  0:03:22:42:37 |
| CONTROL   Current Timesteps 8411      2023-01-27  08:42:23  0:03:22:44:04 |
+----------------------------------------------------------------------------+
| STARTING  Submitted batch job 910236  2023-01-27  08:42:23  0:03:22:44:04 |
| RUNNING   Job is executed             2023-01-27  08:42:53  0:03:22:44:34 |
| BACKUP    /scratch/bt712347/backup/   2023-01-28  02:41:23  0:04:16:43:05 |
| CONTROL   Current Timesteps 10000     2023-01-28  02:42:50  0:04:16:44:31 |
| SUCCESS   Stop monitoring 3x1.5       2023-01-28  02:42:50  0:04:16:44:31 |
+----------------------------------------------------------------------------+
| CONTINUE  Continue monitoring 3x1.5   2023-02-09  18:22:46  0:04:16:44:31 |
| BACKUP    /scratch/bt712347/backup/   2023-02-09  18:22:46  0:04:16:44:31 |
| CONTROL   Current Timesteps 10000     2023-02-09  18:22:47  0:04:16:44:31 |
+----------------------------------------------------------------------------+
| STARTING  Submitted batch job 936758  2023-02-09  18:22:47  0:04:16:44:31 |
| WAITING   Job is pending              2023-02-09  18:23:17  0:04:16:45:01 |
+----------------------------------------------------------------------------+
| CONTINUE  Continue monitoring 3x1.5   2023-02-11  00:12:32  0:04:16:45:01 |
| BACKUP    /scratch/bt712347/backup/   2023-02-11  00:12:32  0:04:16:45:01 |
| CONTROL   Current Timesteps 10000     2023-02-11  00:12:34  0:04:16:45:02 |
+----------------------------------------------------------------------------+
| STARTING  Submitted batch job 937665  2023-02-11  00:12:34  0:04:16:45:02 |
| WAITING   Job is pending              2023-02-11  00:13:04  0:04:16:45:32 |
| RUNNING   Job is executed             2023-02-11  01:58:08  0:04:18:30:36 |
| BACKUP    /scratch/bt712347/backup/   2023-02-12  01:38:54  0:05:18:11:22 |
| CONTROL   Current Timesteps 14497     2023-02-12  01:41:22  0:05:18:13:50 |
+----------------------------------------------------------------------------+
```

```
| STARTING  Submitted batch job 937946  2023-02-12  01:41:22  0:05:18:13:50 |
| RUNNING   Job is executed             2023-02-12  01:41:52  0:05:18:14:20 |
| BACKUP    /scratch/bt712347/backup/   2023-02-13  01:22:37  0:06:17:55:06 |
| CONTROL   Current Timesteps 18985     2023-02-13  01:25:40  0:06:17:58:08 |
+---------------------------------------------------------------------------+
| STARTING  Submitted batch job 938797  2023-02-13  01:25:40  0:06:17:58:08 |
| RUNNING   Job is executed             2023-02-13  01:26:10  0:06:17:58:38 |
| BACKUP    /scratch/bt712347/backup/   2023-02-13  06:51:20  0:06:23:23:48 |
| CONTROL   Current Timesteps 20000     2023-02-13  06:54:11  0:06:23:26:40 |
| SUCCESS   Stop monitoring 3x1.5       2023-02-13  06:54:11  0:06:23:26:40 |
+---------------------------------------------------------------------------+
| CONTINUE  Continue monitoring 3x1.5   2023-02-13  15:34:37  0:06:23:26:40 |
| BACKUP    /scratch/bt712347/backup/   2023-02-13  15:34:37  0:06:23:26:40 |
| CONTROL   Current Timesteps 20000     2023-02-13  15:34:38  0:06:23:26:40 |
+---------------------------------------------------------------------------+
| STARTING  Submitted batch job 939062  2023-02-13  15:34:38  0:06:23:26:41 |
| RUNNING   Job is executed             2023-02-13  15:35:08  0:06:23:27:11 |
| BACKUP    /scratch/bt712347/backup/   2023-02-14  11:31:13  1:00:19:23:15 |
| CONTROL   Current Timesteps 20000     2023-02-14  11:38:17  1:00:19:30:19 |
+---------------------------------------------------------------------------+
| STARTING  Submitted batch job 952358  2023-02-14  11:38:17  1:00:19:30:19 |
| RUNNING   Job is executed             2023-02-14  11:38:47  1:00:19:30:49 |
| ERROR     SLURM Job failed to execute 2023-02-14  15:38:58  1:00:23:31:01 |
| RESTORE   /scratch/bt712347/backup/   2023-02-14  15:38:58  1:00:23:31:01 |
| CONTROL   Current Timesteps 20000     2023-02-14  15:44:44  1:00:23:36:46 |
+---------------------------------------------------------------------------+
| STARTING  Submitted batch job 956235  2023-02-14  15:44:44  1:00:23:36:46 |
| RUNNING   Job is executed             2023-02-14  15:45:14  1:00:23:37:17 |
| ERROR     SLURM Job failed to execute 2023-02-15  11:01:35  1:01:18:53:38 |
| RESTORE   /scratch/bt712347/backup/   2023-02-15  11:01:44  1:01:18:53:46 |
| CONTROL   Current Timesteps 20000     2023-02-15  11:06:52  1:01:18:58:54 |
+---------------------------------------------------------------------------+
| STARTING  Submitted batch job 969292  2023-02-15  11:06:52  1:01:18:58:54 |
| RUNNING   Job is executed             2023-02-15  11:07:22  1:01:18:59:24 |
| ERROR     SLURM Job failed to execute 2023-02-16  10:48:07  1:02:18:40:09 |
| RESTORE   /scratch/bt712347/backup/   2023-02-16  10:48:07  1:02:18:40:09 |
| CONTROL   Current Timesteps 20000     2023-02-16  10:51:17  1:02:18:43:20 |
+---------------------------------------------------------------------------+
| STARTING  Submitted batch job 970455  2023-02-16  10:51:17  1:02:18:43:20 |
| WAITING   Job is pending              2023-02-16  10:51:47  1:02:18:43:50 |
| RUNNING   Job is executed             2023-02-16  12:56:50  1:02:20:48:53 |
| BACKUP    /scratch/bt712347/backup/   2023-02-16  16:11:56  1:03:00:03:59 |
+---------------------------------------------------------------------------+



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

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
1	6.0	1x1	12	16	6	0.02	1.4	21	83
2	6.0	1x1	12	32	6	0.02	1.4	21	83
3	6.0	1x1	12	48	9	0.02	1.4	21	83
4	6.0	1x1	12	64	9	0.02	1.4	21	83
5	6.0	1x1	16	16	9	0.02	1.4	21	83
6	6.0	1x1	16	32	9	0.02	1.4	21	83
7	6.0	1x1	16	48	6	0.02	1.4	21	83
8	6.0	1x1	16	48	9	0.02	1.4	21	83
9	6.0	1x1	16	48	9	0.025	1.4	21	83
10	6.0	1x1	16	48	9	0.02	0.7	11	83
11	6.0	1x1	16	48	9	0.02	1.4	21	43
12	6.0	1x1	16	48	9	0.02	1.4	21	63
13	6.0	1x1	16	64	6	0.02	1.4	21	83
14	6.0	1x1	16	64	9	0.02	1.4	21	83
15	6.0	2x1	16	48	9	0.02	1.4	21	83
16	6.0	2x2	16	48	9	0.02	1.4	21	83
17	6.0	3x1	16	48	9	0.02	1.4	21	83
18	6.0	3x1.5	16	48	9	0.02	1.4	21	83
19	6.0	3x1.5	16	48	9	0.02	1.4	21	83
20	6.0	3x2.5	16	48	9	0.02	1.4	21	83
21	6.0	3x3	16	48	9	0.02	1.4	21	83
22	6.0	3x5	16	48	9	0.02	1.4	21	83
23	6.0	4x1	16	48	9	0.02	1.4	21	83
24	6.2	2x2	16	64	9	0.02	1.4	21	83
25	6.2	3x3	16	48	9	0.02	1.4	21	83
26	6.3	1x1	16	64	9	0.02	1.4	21	83
27	6.4	3x3	16	48	9	0.02	1.4	21	83

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	2x1	18000.0	30000		True	2	True
16	6.0	2x2	18000.0	30000		True	2	True
17	6.0	3x1	48000.0	80000		True	3	True
18	6.0	3x1.5	16840.3	28068		True	4	True
19	6.0	3x1.5	14170.0	23618	23618	True	3	True
20	6.0	3x2.5	6000.0	10000		True	3, 4	True
21	6.0	3x3	7847.0	13083		True	4	True
22	6.0	3x5	4769.0	7958		True	4	True
23	6.0	4x1	30000.0	50000	47084	True	4	True
24	6.2	2x2	6000.0	10000		True	2	True
25	6.2	3x3	14682.7	24475	14473-16132	True	3	True
26	6.3	1x1	6000.0	10000		False	0	True
27	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/boxsize2x1/Ns16/Nvpar48/Nmu9
16	./data/S6_rlt6.0/boxsize2x2/Ns16/Nvpar48/Nmu9
17	./data/S6_rlt6.0/boxsize3x1/Ns16/Nvpar48/Nmu9
18	./data/S6_rlt6.0/boxsize3x1.5/Ns16/Nvpar48/Nmu9
19	./data/S6_rlt6.0/boxsize3x1.5/Ns16/Nvpar48/Nmu9/Broken
20	./data/S6_rlt6.0/boxsize3x2.5/Ns16/Nvpar48/Nmu9
21	./data/S6_rlt6.0/boxsize3x3/Ns16/Nvpar48/Nmu9
22	./data/S6_rlt6.0/boxsize3x5/Ns16/Nvpar48/Nmu9
23	./data/S6_rlt6.0/boxsize4x1/Ns16/Nvpar48/Nmu9
24	./data/S6_rlt6.2/boxsize2x2/Ns16/Nvpar64/Nmu9
25	./data/S6_rlt6.2/boxsize3x3/Ns16/Nvpar48/Nmu9
26	./data/S6_rlt6.3/boxsize1x1/Ns16/Nvpar64/Nmu9
27	./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: 11 April 2023)

The radial size convergence of the $\mathrm{E}\times\mathrm{B}$ staircase pattern is adressed in local gradient-driven flux tube simulations of ion temperature gradient (ITG) driven turbulence. Its is shown that a mesoscale pattern size of $\sim 57.20-76.27\,\rho$ 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[2–4] as well as global gradient-driven[5–7] and global flux-driven[1,8–11] 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)[12] 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 Tab. I and for more details about the definition of individual quantities the reader is referred to Ref. 2 and 12.

	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 $(L_x,\, L_y) = (76.27,\, 89.76)\,\rho$ in the radial and binormal direction. Here, x is the radial coordinate that labels the flux surfaces normalized by the thermal Larmor radius ρ, 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[13]. 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 [Tab. 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 by[2,14,15]

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

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, 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[4]

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

The $\mathrm{E}\times\mathrm{B}$ shearing rate $\omega_{\mathrm{E}\times\mathrm{B}}$ is the radial derivative of the advecting zonal flow velocity[16,17] and quantifies the zonal flow induced shearing of turbulent structures[16,18,19].

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 12.

[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 $|\omega_{\mathrm{E}\times\mathrm{B}}| \approx \gamma$[17,20] ($\gamma$ is the growth rate of the most unstable linear ITG driven Eigenmode), connected by steep flanks where $\omega_{\mathrm{E}\times\mathrm{B}}$ crosses zero. Fig. 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, the converged radial size 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 time scale 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 $\sim 57.20-76.27\,\rho$ (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 $N_{\mathrm{R}}\times N_{\mathrm{B}} \in [1\times1,\ 2\times2,\ 3\times3]$. Interestingly, suppression of the turbulence by the emergence of a fully developed staircase pattern 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) also this test 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.

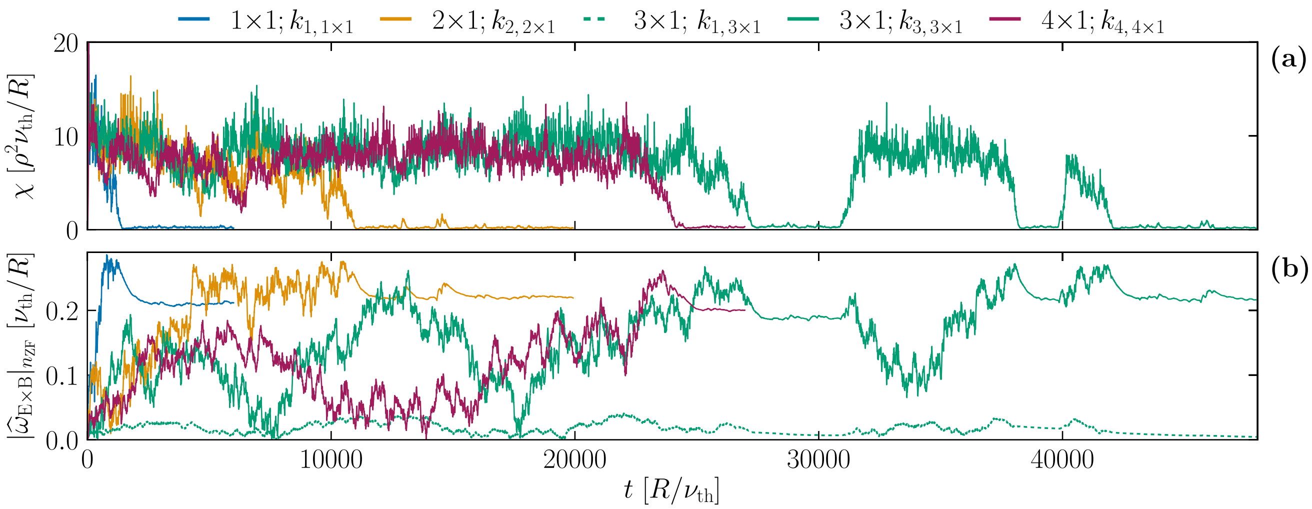


FIG. 1: **(a)** Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for radial increased box sizes
(b) Time traces of $|\widehat{\omega}_{\mathrm{E}\times\mathrm{B}}|_{n_{\mathrm{ZF}}}$ for radial increased box sizes

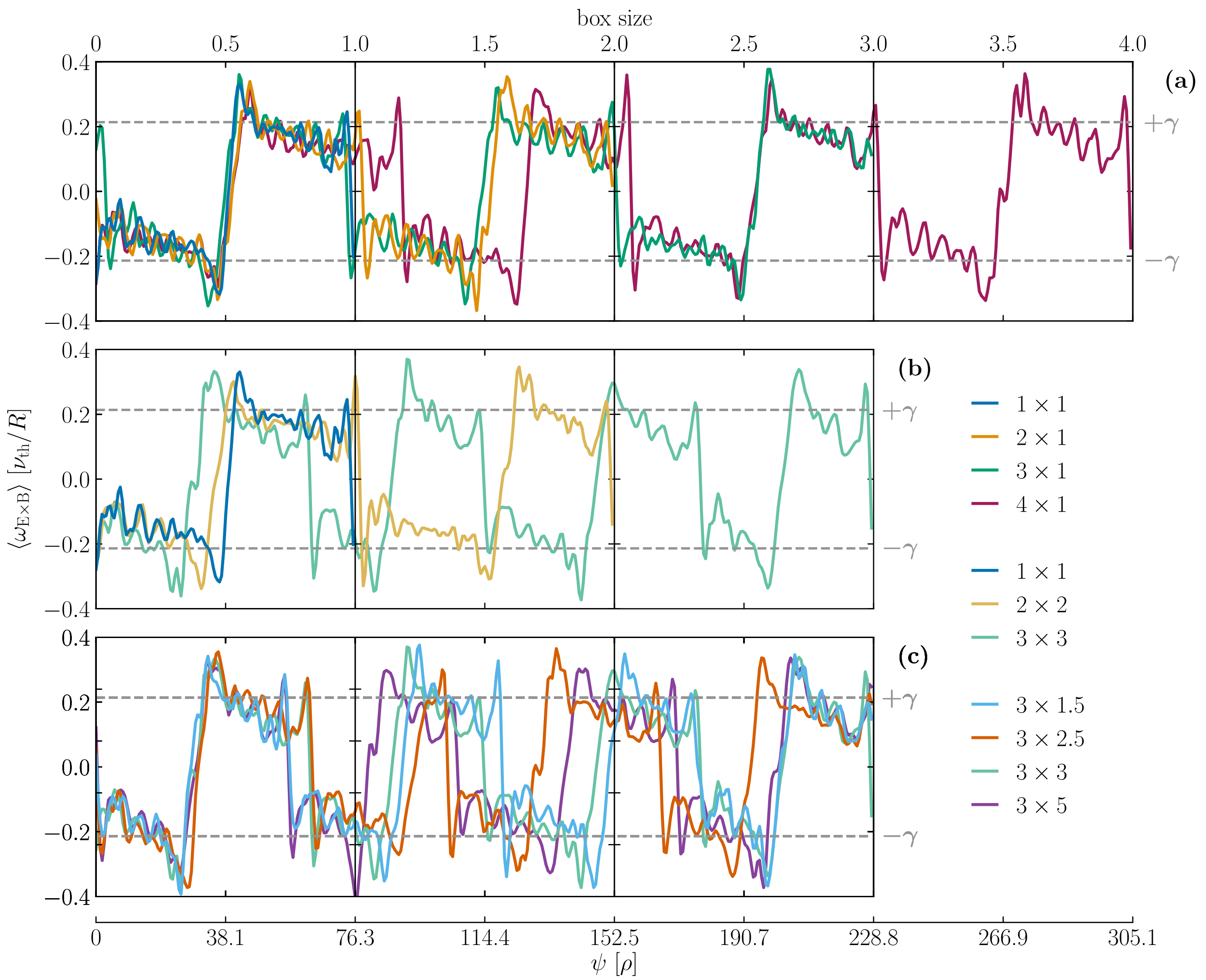


FIG. 2: Comparison of shearing rate $\omega_{\mathrm{E\times B}}$ for each box sizes scan averaged over given time interval and the growth rate $\pm\gamma$ of the most unstable linear ITG driven Eigenmode. The staircase structures are radially shifted with respect to each over till alignment for better 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: $t_{3\times1.5} \in [2000, 3000]$, $t_{3\times2.5} \in [2000, 3000]$, $t_{3\times3} \in [2000, 3000]$ $t_{3\times5} \in [1000, 3000]$

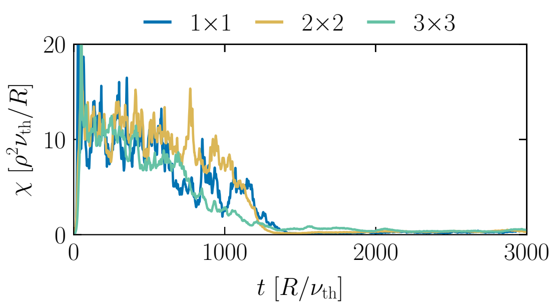


FIG. 3: Time traces of the heat conduction coefficient χ for $R/L_T = 6.0$ for isotropic increased box sizes

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 $\lambda \in [57.20,\ 76.27]\,\rho$ as observed in the first test is addressed in the next paragraph.

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 $\sim 2000\,R/v_\mathrm{th}$ [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 $\lambda\in[57.20,\ 76.27]\,\rho$ 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 3 and 4 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).

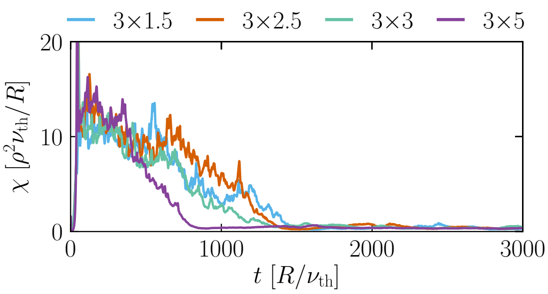


FIG. 4: Time traces of the heat conduction coefficient χ for $R/L_T=6.0$ for binormal increased box sizes

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 $t>11000\,R/v_\mathrm{th}$, while stationary turbulence during the entire simulation time trace of $12000\,R/v_\mathrm{th}$ is found for $R/L_T=6.4$. The finite heat flux threshold, hence, is $R/L_T|_\mathrm{finite}=6.3\pm0.1$ in accordance to 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 $\sim 57.20-76.27\,\rho$.

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 $\sim 57.20-76.27\,\rho$ 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 $10\,\rho$[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[1,2,5,14].

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]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).
[4]F. Rath, A. G. Peeters, and A. Weikl, Phys. Plasmas **28**, 072305 (2021).
[5]B. F. McMillan, S. Jolliet, T. M. Tran, L. Villard, A. Bottino, and P. Angelino, Physics of Plasmas **16**, 022310 (2009), https://doi.org/10.1063/1.3079076.
[6]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).
[7]J. Seo, H. Jhang, and J.-M. Kwon, Physics of Plasmas **29**, 052502 (2022), https://doi.org/10.1063/5.0086587.
[8]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).
[9]W. Wang, Y. Kishimoto, K. Imadera, H. Liu, J. Li, M. Yagi, and Z. Wang, Nuclear Fusion **60**, 066010 (2020).
[10]Y. J. Kim, K. Imadera, Y. Kishimoto, and T. S. Hahm, Journal of the Korean Physical Society **81**, 636 (2022).
[11]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), https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2021.0231.
[12]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).
[13]S. Hamada, Kakuyugo Kenkyu **1**, 542 (1958).
[14]F. Rath, A. G. Peeters, R. Buchholz, S. R. Grosshauser, P. Migliano, A. Weikl, and D. Strintzi, Phys. Plasmas **23**, 052309 (2016).
[15]M. J. Pueschel, M. Kammerer, and F. Jenko, Physics of Plasmas **15**, 102310 (2008).
[16]T. S. Hahm and K. H. Burrell, “Flow shear induced fluctuation suppression in finite aspect ratio shaped tokamak plasma,” Phys. Plasmas **2**, 1648–1651 (1995).
[17]R. E. Waltz, R. L. Dewar, and X. Garbet, Phys. Plasmas **5**, 1784–1792 (1998).
[18]H. Biglari, P. H. Diamond, and P. W. Terry, Phys. Fluids B: Plasma Physics **2**, 1–4 (1990).
[19]K. H. Burrell, Phys. Plasmas **4**, 1499–1518 (1997).
[20]R. E. Waltz, G. D. Kerbel, and J. Milovich, Phys. Plasmas **1**, 2229 (1994).

CHAPTER

7 Bibliography

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

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

[3] 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.

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

[5] 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.

[6] 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 $\mathbf{E} \times \mathbf{B}$ staircase in magnetized plasmas. *Phys. Rev. Lett.* **114**, 085004.

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

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

[9] 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.

[10] 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.

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

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

[13] 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. *Phys. of Plasmas* **7** (3), 969–983.

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

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

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

[17] 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.

[18] 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. Plasmas* **23** (8), 082517.

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

[20] 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. Plasmas* **23** (5), 052309.

[21] 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. Plasmas* **28** (7), 072305.

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

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

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

[25] 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.

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

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

[28] 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.

[29] 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. Plasmas* **24** (10), 102317.