Universidade do Minho

Dynamics of large-scale perturbations of steady convective dynamos in a horizontal plane layer of electrically conducting fluid

Simon Ranjith Jeyabalan

Tese de Doutoramento apresentada à
Faculdade de Ciências da Universidade do Porto,
Universidade de Aveiro e Universidade de Minho,
Matemática Aplicada

2022

Dynamics of large-scale perturbations of steady convective dynamos in a horizontal plane layer of electrically conducting fluid

Simon Ranjith Jeyabalan
Programa Doutoral em Matemática Aplicada
Matemática Aplicada
2022

Orientador
Dr. Roman Chertovskih
Investigador, Research Centre for Systems and Technologies,
Faculdade de Engenharia
Universidade do Porto

Coorientadores
Dr. Sílvio Marques de Almeida Gama
Professor, Departamento de Matematica,
Faculdade de Ciências
Universidade do Porto

Dr. Vladislav Zheligovsky
Principal researcher,
Institute of Earthquake Prediction Theory and Mathematical
Geophysics, Russian Academy of Sciences, Moscow

Acknowledgements

It is the Summer of 2022. Here I would like to acknowledge the people who served as my scientific advisors, my family, my friends, and the people who were my support and encouragement. I am sincerely thankful to all of them but I feel that my words cannot express my gratitude adequately.

In Winter 2017 Professor Sílvio Gama introduced me to Professor Vladislav Zheligovsky and Professor Roman Chertovskih. Immediatedly they presented me with a maths problem that could lead towards a doctoral degree. My understanding was limited then, but my curiosity and the peace that I had inside me directed my paths. It knew it would be challenging because they mentioned it quite frequently in their conversations.

Looking back there were many hurdles along the way. But never did I doubt if all this can come to fruition. I remember, once a good friend of mine told me, "the way you eat an elephant is to take one bite at a time". It is such a true statement. Knowledge gives you confidence. I realized the way forward in this season is to take one small step at a time.

I have admired Professor Vladislav's expertise on fluid dynamics. His guidance during the entire duration of the project was phenomenal. His commitment in imparting knowledge - that which he has meticulously accumulated over four decades - to his students is commendable. I thank him and Professor Olga Podvigina for their support and supervision.

I thank Professor Roman Chertovskih for sharing his expertise on scientific computations. His friendliness, support and assistance is greatly appreciated. His criticism, suggestions or comments, of any computational task that I performed during this period has helped me in my analytical skills.

I am equally thankful to Professor Sílvio Gama for his supervision of the project. His skills on analysis using statistical tools are impressive. Discussions with him were always entertaining - there never had been a dull moment with him.

I'm thankful to my father Ambrose Jeyabalan and elder brother Emmanuel for their love and support. I thank my mother Merlyn and grandmother (my mother's mom), Esther. A special thank you to my grandmother who brought in the tradition of family prayer in our family. She also helped me to use Psalms 23, 91, and 121 from the Bible as a prayer when I was about 8 years old.

I'm grateful to all the personnel in the respective computing centres for their contributions, financial support and the resources:

- Navigator, Laboratory for Advanced Computing, University of Coimbra
- Centro de Matemática da Universidade do Porto, University of Porto

- Research Center for Systems and Technologies (SYSTEC), University of Porto
- Oblivion, University of Evora
- Institute of Earthquake Prediction Theory and Mathematical Geophysics, Moscow, Russian Federation

I want to thank Fundação para a Ciência e a Tecnologia (FCT) for the scholarship (Reference: PD/BD/142889/2018) and for the financial support of the computations under the projects: CPCA/A00/7317/2020 and CPCA/A0/7296/2020.

A heartfelt 'thank you' to God for this chapter in my life: He has been with me and always has been faithful to keep His word to me. Psalms 36:9 says, "For with thee is the fountain of life: in thy light shall we see light". I believe God is the source and origin of life. It is only through the light He imparts can we truly see our lives and the truths of our lives. Our understanding comes from Him.

Abstract

In this thesis, we numerically investigate thermal space-periodic convection of an electrically conducting fluid in a rotating layer, in the possible presence of magnetic field. Thermal convection in the Boussinesq approximation is assumed. The nonlinear convective dynamo is described by the Navier-Stokes (with the buoyancy, Coriolis and Lorentz forces), magnetic induction and heat transfer equations with perfectly electrically conducting stress-free horizontal boundaries maintained at constant temperatures. We assume as initial conditions small or large perturbations of a magnetohydrodynamic (MHD) steady state with square horizontal periodicity cells examined by Chertovskih *et al.*, Physica D, 239, 1188-1209 (2010). Its weakly non-linear perturbations involving spatial scales were considered using the multiscale formalism by Chertovskih and Zheligovsky, Physica D, 313, 99-116 (2015).

One of the objectives of this work is to study perturbations, whose periods in horizontal directions are one to four times larger than that of the MHD state subjected to perturbation. Our computations reveal that the scale ratio 1/4 is still too large for the perturbations to be consistent with the asymptotic behaviour observed *ibid.* Next, fully nonlinear magnetic field generation remains sustained only in the elongated periodicity cells, where one of the horizontal spatial periods has the same period as the state subjected to perturbation, and in the complementary direction it is twice or four times larger.

All the computed attractors are chaotic, except for the attractor in the triple-period elongated fluid cells. Two distinct MHD attractors in the double-period and an unique attractor in quadruple-period fluid cells were identified. In these geometries no hydrodynamic attractors were found with the magnetic field decaying indefinitely.

Estimates of magnetic Reynolds numbers as well as the behaviour of the mean helicity does not distinguish between the generating and the non-generating (triple-period) attractors in elongated cells. In "wide" periodicity cells, where both periods in the horizontal direction are at least twice larger than that of the MHD state subjected to perturbation, the characteristic length scales decrease and become less than one period of the initial state. As a result, the convection is less vigorous and generation of magnetic field ceases gradually.

Computations were also performed to analyze a family of integral quantities $D_{q,s}$ as examined by Donzis *et al.*, Journal of Fluid Mechanics, 732, 316-331 (2013). D_q and $D_{q,2}$ computed for the flow decrease monotonically for all the considered values $q \leq 10$ in all simulations. However, monotonicity is broken at certain times for the magnetic field. The behaviour of $D_{q,s}$ in s is non-monotonic both for the flow and the magnetic field.

Resumo

Nesta tese, investigamos numericamente a convecção térmica, periódica no espaço, de um fluido condutor numa camada rotativa, na presença de um campo magnético. Assumimos a convecção térmica na aproximação de Boussinesq. O dínamo convectivo não-linear é descrito pela equação de Navier-Stokes (com a flutuabilidade, forças de Coriolis e Lorentz), pela equação de indução magnética e pela equação de transferência de calor. As fronteiras horizontais não têm tensões, são isotérmicas e condutoras eléctricas perfeitas. Assumimos como condições iniciais as pequenas ou grandes perturbações de um estado estacionário MHD, com células de periodicidade horizontal quadradas, estudadas por Chertovskih *et al.*, Physica D, 239, 1188-1209 (2010). As suas perturbações fracamente não-lineares, envolvendo escalas espaciais, foram consideradas, usando o formalismo multiescala de Chertovskih e Zheligovsky, Physica D, 313, 99-116 (2015).

Um dos objetivos desta tese é estudar as perturbações, cujos períodos nas direções horizontais são uma a quatro vezes maiores do que a do estado MHD sujeito à perturbação. Os nossos cálculos revelam que a razão de escala 1/4 ainda é demasiadamente grande para que as perturbações sejam consistentes com o comportamento assintótico observado "no mesmo lugar". Em seguida, a geração de campo magnético totalmente não-linear permanece sustentada apenas no alongamento células de periodicidade, onde um dos períodos espaciais horizontais tem o mesmo período que o estado sujeito à perturbação e na direção complementar é duas ou quatro vezes maior.

Todos os atratores calculados são caóticos, exceto o atrator nas células de fluido alongadas de período triplo. Dois atratores MHD distintos com período duplo e um único atrator em células de fluido de período quádruplo foram identificados. Nestas geometrias não foram encontrados atratores hidrodinâmicos com o decaimento do campo magnético ao longo do tempo.

As estimativas dos números de Reynolds magnéticos, bem como o comportamento da helicidade média, não distinguem entre os atratores geradores e não-geradores (período triplo) em células alongadas. Em células de periodicidade "larga", onde ambos os períodos na direção horizontal são pelo menos duas vezes maiores que o do estado MHD submetido a perturbação, as escalas de comprimento características diminuem e tornam-se mais pequenas do que o período do estado inicial. Como consequência, a convecção é menos vigorosa e a geração do campo magnético cessa gradualmente.

Os cálculos também foram realizados para analisar uma família de quantidades integrais $D_{q,s}$ conforme estudados em Donzis *et al.*, Journal of Fluid Mechanics, 732, 316-

331 (2013). As quantidades D_q e $D_{q,2}$, calculadas para o escoamento, diminuem, em todas as simulações, monotonicamente para todos os valores considerados $q \leq 10$. No entanto, a monotonicidade é quebrada em determinados momentos para o campo magnético. O comportamento de $D_{q,s}$ em função s é não-monótono tanto para o escoamento como para o campo magnético.

Contents

Acknowledgements **i**

Abstract **iii**

Resumo **v**

List of Figures **ix**

List of Tables **xiii**

Introduction **1**

1 Statement of the problem **7**
1.1 Free convection 8
1.1.1 Boussinesq approximation 9
1.2 Convection in a plane layer 11
1.2.1 Boundary conditions 13
1.3 Free convection in the presence of a magnetic field 14
1.3.1 Boundary conditions 15
1.4 Convective cells 16
1.5 Summary 17

2 Numerical methods and Software libraries **21**
2.1 Spectral methods 22
2.1.1 Generalities 22
2.1.2 The Galerkin-type method 23
2.1.3 The collocation method 23
2.1.4 The method of least-squares 24
2.1.5 The Fourier method 24
2.1.5.1 Fourier series 25
2.1.5.2 Aliasing 29
2.1.5.3 Aliasing removal 30
2.1.5.4 Sine and cosine Fourier series 31
2.2 Solenoidality in the Fourier space 34

2.3 Equations of hydromagnetic convection in the Fourier space 35
2.4 Time discretization method 36
2.5 Symmetries 40
2.6 Software libraries 41
2.6.1 FFTW 42
2.6.1.1 Features 42
2.6.1.2 Composition and runtime structure 42
2.6.1.3 Plans 43
2.6.1.4 Wisdom 44
2.6.1.5 Distributed memory with MPI 45
2.6.2 P3DFFT 46
2.6.2.1 Features 46
2.6.2.2 Data mapping 47
2.7 Summary 48

3 Numerical Results **51**
3.1 Numerical techniques 52
3.2 Results of simulation 54
3.2.1 Regimes with the periodicites 1x2/2x1 64
3.2.2 Regimes with the periodicites 1x3/3x1 77
3.2.3 Regimes with the periodicities 1x4/4x1 82
3.2.4 Regimes in the (M_1, M_2)-periodicity for $M_i \geq 2$ 87
3.2.5 Videos of the kinetic and magnetic energy evolution 89
3.2.6 Analysis of integral parameters of the regimes 91
3.3 Summary 95

4 Conclusion **97**

Bibliography **99**

Appendix A Software **105**
A.1 Code verification 105
A.1.1 Linear dynamo 105
A.1.2 Non-linear dynamo 106
A.2 Scaling of the parallel code 109
A.3 Code 114
A.3.1 magneto.f90 114

List of Figures

1.1 An illustration of a volume of fluid fixed in space. 7

1.2 An illustration of a fluid between two horizontal planes in Rayleigh Bénard convection. 10

1.3 An illustration of a periodicity box in a horizontal layer. 12

1.4 A sketch of the convective layer with $(M_1, M_2) = (2, 2)$. 16

1.5 Isosurfaces of the kinetic and magnetic energy densities at the levels of 1/2 of the respective maxima of the short-scale convective dynamo regime subject to perturbations (top row). Isolines of the kinetic and magnetic energy densities at the midplane $x_3 = 0$, step 1.5 and 0.3 respectively (bottom row) 18

2.1 An illustration of a 2D block (pencil decomposition) of a 3D grid. 46

3.1 Energy spectrum for the flow at $t = 220$ in run 2 for $M_1 = M_2 = 2$ 53

3.2 Space- and time-averaged kinetic and magnetic energies for the attractors obtained in the runs with the standard ($128M_1 \times 128M_2 \times 97$ harmonics) and alternative ($256 \times 256 \times 257$ harmonics) resolution 60

3.3 Temporal behaviour of the time-dependent estimates of the length scales, ℓ'_1 and ℓ'_3, and of the magnetic Reynolds number, Rm'_1 and Rm'_3, in 1×2.2 63

3.4 Temporal behaviour of the kinetic and magnetic energies and the evolution of the magnetic energy versus kinetic energy in the double-period runs, 1×2.1, 1×2.2, 1×2.3, 2×1.1, 2×1.2, 2×1.3 67

3.5 Isosurfaces of the kinetic and magnetic energy densities at the level of 1/2 of the respective maxima in the run 1×2.3 at time $t = 1650.707$ 68

3.6 Temporal behaviour of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$ and $\mathrm{Re}\,\hat{v}^1_{222}$ at three time intervals: $0 \le t \le 320$ (phase T of the initial transient processes, chaotic behaviour and approach of the quasiperiodic metastable state), $800 \le t \le 830$ (phase G in the vicinity of the quasiperiodic hydrodynamic metastable state), $1520 \le t \le 2020$ (departure from phase G, phase S of the saturated chaotic regime and approach of the chaotic phase D), and $2000 \le t \le 2200$ (phase D of magnetic field decay), in the run 2×1.2 70

3.7 Poincaré mapping of the plane ($\mathrm{Re}\,\hat{v}^1_{111}$, $\mathrm{Re}\,\hat{v}^2_{111}$) defined by the condition $\mathrm{Re}\,\hat{v}^3_{222} = 0$ on the time intervals $300 \le t \le 1500$ (phase G), $1600 \le t \le 2015$ (phase S and its small neighbourhood), and $2000 \le t \le 2200$ (phase D), in the run 2×1.2 71
3.8 Temporal power spectra of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$ and $\mathrm{Re}\,\hat{v}^1_{222}$ at the time intervals, $285 \le t \le 1555$ (phase G), $1580 \le t \le 1940$ (phase S), and $2000 \le t \le 2200$ (phase D), in the run 2×1.2 72
3.9 Temporal behaviour of the kinetic and magnetic energies in the double-period run 2×1.2 for $1540 \le t \le 1960$, phase S. The evolution of the magnetic energy versus kinetic energy in this run at the same time interval 73
3.10 Temporal behaviour of the mean helicity with the kinetic energy in the runs 1×2.2, 2×1.2 and 2×1.1 75
3.11 Temporal behaviour of the Fourier coefficients $\mathrm{Re}\,\hat{b}^1_{111}$ and $\mathrm{Im}\,\hat{b}^1_{222}$ in the run 2×1.1 76
3.12 Temporal behaviour of the kinetic and magnetic energies in triple-period runs and projection of the trajectories on the plane of magnetic and kinetic energies: runs, 3×1.1 and 3×1.2 for the standard resolution; 3×1.2 for the alternative resolution 78
3.13 Isosurfaces of the kinetic and magnetic energy densities in the run 3×1.2 (the standard resolution) at times $t = 380.671$ and $t = 566.671$ at the levels of 15%, 30% and 50% of the respective maxima of the densities 79
3.14 Isosurfaces of the zero-value flow components v^1, v^2, v^3 in the run 3×1.2 (the standard resolution) at times $t = 380.671$ and $t = 566.671$ 80
3.15 Temporal behaviour of the local in time estimates of the magnetic Reynolds number, Rm'_1 and Rm'_3 and of the kinetic helicity together with the kinetic energy in the run 3×1.2 81
3.16 Temporal behaviour of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$, and $\mathrm{Re}\,\hat{v}^1_{222}$, for $0 \le t \le 600$, in the run 3×1.2 81
3.17 Temporal power spectra of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$ and $\mathrm{Re}\,\hat{v}^1_{222}$ at the interval $50 \le t \le 533$, and at the interval $75 \le t \le 975$ in the run 3×1.2 83
3.18 Poincaré sections of the plane ($\mathrm{Re}\,\hat{v}^1_{111}$, $\mathrm{Re}\,\hat{v}^2_{111}$) defined by the condition $\mathrm{Re}\,\hat{v}^3_{222} = 0$ for the hydrodynamic attractors in the runs 1×3.1 (constructed for the time interval $100 \le t \le 487$), and 3×1.3 (for $300 \le t \le 550$) 83
3.19 Temporal behaviour of the kinetic and magnetic energies, and projection of the trajectories on the plane of magnetic and kinetic energies in the quadruple-period runs: 4×1.2 and 4×1.1 for the standard resolution; 4×1.1 for the alternative resolution 84
3.20 Temporal behaviour of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$, and $\mathrm{Re}\,\hat{v}^1_{222}$, in the run 4×1.1 . 85
3.21 Temporal power spectra of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$ and $\mathrm{Re}\,\hat{v}^1_{222}$ at the time intervals $15 \le t \le 120$, $132 \le t \le 212$, and $223 \le t \le 242$, in the run 4×1.1 for the standard resolution 86

3.22 Temporal behaviour of the local in time estimates of the magnetic Reynolds number, Rm'_1 and Rm'_3, and of the kinetic helicity together with the kinetic energy in the run 4×1.1 87
3.23 Isosurfaces of the kinetic and magnetic energy densities at the levels of 50% and 35% of the respective maxima of the densities in the run 4×1.1 at time $t = 222.626$ 87
3.24 Temporal behaviour of the kinetic and magnetic energies in the double-period runs 4×4.2 and 2×4.2. 88
3.25 Temporal behaviour of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$ and $\mathrm{Re}\,\hat{v}^1_{222}$ in the run 2×4.2 . . 88
3.26 Temporal power spectra of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$ and $\mathrm{Re}\,\hat{v}^1_{222}$ at the time interval $40 \le t \le 381.21$ in the run 2×4.2 89
3.27 Poincaré section of the plane ($\mathrm{Re}\,\hat{v}^1_{111}$, $\mathrm{Re}\,\hat{v}^2_{111}$) defined by the condition $\mathrm{Re}\,\hat{v}^3_{222} = 0$ for the runs 2×4.2, constructed for the time interval $40 \le t \le 381.21$. 90
3.28 Time evolution of the quantities $D^{\mathbf{v}}_q$, $D^{\mathbf{v}}_{q,s}$, $D^{\mathbf{b}}_q$ and $D^{\mathbf{b}}_{q,s}$ for $s = 2$ and q varying in the range $2 \le q \le 10$, and for $q = 2$ and s in the range $1 < s \le 10$ in the run 4×1.1 94

A.1 Evolution of magnetic energy for a fluid at rest and $e^{-2*0.617t}$ 107
A.2 Evolution of magnetic energy of a divergence-free velocity field and $e^{2\times 0.09696\,t}$ 107
A.3 Evolution of magnetic energy of a non-divergence-free velocity field and $e^{2\times 0.12599\,t}$ 107
A.4 The evolution of kinectic E_v and magnetic energy E_b of the MHD system for the parameters used by Podviginia [2006] 108
A.5 Elapsed time of the parallel code for 25 time steps for $512 \times 512 \times 97$ 111
A.6 Speed-ups for 25 time steps for the resolution $512 \times 512 \times 97$. 112
A.7 Scaling of P3DFFT with forward and backward sine Fourier transforms for 10 iterations. 112
A.8 Speed-ups of forward and backward sine transforms for 10 iterations 113

List of Tables

2.1 A variant of the classical Runge-Kutta method for weights and coefficients proposed by Cox and Matthews [2002]. 38
2.2 Mapping of data array onto processor grid and memory layout. 48

3.1 Minimum energy spectrum decay (orders of magnitude) in the simulated regimes of spatial periods $M_1\,L$ and $M_2\,L$ in x_1 and x_2, respectively 55
3.2 Attractors and metastable states in simulations in the (M_1, M_2)-periodicity boxes 56
3.3 Magnetic Reynolds number estimates in runs for perturbations of moderate scale separation (in the order of increasing Rm_4) 62
3.4 Magnetic Reynolds number estimates in various phases of the regimes observed in the runs with the periodicities 1×2/2×1 65

Introduction

The atmosphere surrounding the cosmic bodies may appear void but it is a dynamic environment. It has invisible matter and is dominated by forces. Most planets in the Solar System have magnetic fields around them and this region of outer space is referred to as the magnetosphere. They deflect the highly charged particles called cosmic rays from the Sun or coming from the interstellar space - they also trap these particles and this produces the luminous auroras. So they are assumed to protect a planets' surface from harmful radiations.

The magnetosphere surrounding each planet in our Solar System is different: Venus and Mars do not have a magnetosphere. The Earth's magnetosphere is believed to be formed by the electrical currents in a constantly moving molten metal core on the inside. Its shape resembles an ice cream cone with rounded front and a long trailing tail that faces away from the Sun - a consequence of an interaction of the terrestrial magnetic field and the solar winds. Mercury has about one percent of the Earth's magnetic field. Jupiter, on the other hand, has the strongest and biggest magnetic field in our Solar System and this is only second to the Sun. In its interior it does not have a molten metal core; instead, it has compressed liquid metallic hydrogen. Evaporating oxygen and water molecules from Saturn's rings transform the shape of its magnetosphere. Uranus has a weak magnetic field and it varies across its surface. This is because their magnetosphere is offset from their rotational axis. Neptune's magnetic field is similar to Uranus and auroras can appear across the planet - not just close to the poles as on the Earth.

Thus, it is interesting to understand these environments in our Solar System. In particular, the Earth's.

The Earth's internal structure is layered. It has an inner core, an outer core, the mantle and the crust. The outer core has a molten substance and is assumed to be in a state of thermal convection with a probable belief that its heat source could be coming from the radioactive decay of isotopes of uranium, thorium and potassium. This convective motion of the conducting fluid in the presence of a small magnetic field induces an electromotive force generating electric currents. The associated magnetic field in return is believed to sustain the initial magnetic field. This mechanism is called convective dynamo. It was originally proposed by Larmor in the beginning of the previous century [Larmor, 2013].

The physical phenomena in the convective dynamos in planets and stars can be described by the following governing equations:

- the Navier-Stokes equation representing the fluid-flow,

- the magnetic induction equation, describing the evolution of the magnetic field in the electrically conducting fluid, and
- the heat transfer/conduction equation, describing the evolution of temperature in the system.

The magnetic induction equation is a linear (in magnetic field) parabolic partial differential equation. Magnetic field affects the fluid flow by means of the Lorentz force, which is quadratic in magnetic field. Hence if the magnetic field is small then its influence on the flow can be neglected. Consequently, with a given fluid flow, the evolution of a small initial magnetic field (at least on short time intervals when the field is growing) can be studied by solving the magnetic induction equation alone. This problem is called the kinematic dynamo problem [Moffat, 1978]. If the magnetic field does not decay for longer times, then we say that the flow acts as a kinematic dynamo for the corresponding value of magnetic diffusivity, η, defining the electromagnetic properties of the fluid.

If the flow is steady (i.e., the flow velocity does not depend on time), then the kinematic dynamo problem reduces to an analysis of the spectrum of the magnetic induction operator. Suppose the volume occupied by the conducting fluid is finite with some regular boundary conditions for the magnetic field. If the dominant (i.e., with the maximal real part) eigenvalue of the magnetic induction operator has a positive real part, then the flow is a kinematic dynamo for the employed value of η.

During the initial development of the kinematic dynamo theory, the so-called antidynamo theorems were introduced. They prove that flows with certain characteristics cannot act as dynamos: an axisymmetric flow cannot generate an axisymmetric magnetic field with the same axis of symmetry [Cowling 1934, 1957], flows with a toroidal velocity [Bullard *et al.*, 1954, Elsasser 1946] and plane-parallel flows (flow that is orthogonal to a certain fixed direction at any point in the fluid volume) [Zeldovich, 1957, Zeldovich *et al.*, 1980]. The latter two are referred to as the Elsasser and Zeldovich antidynamo theorems.

The dynamo theory was developed during the latter part of the 20th century. An important concept in this theory is the notion of magnetic α-effect suggested by E. Parker [1955, 1979]. His assumption is that due to the interactions of small-scale fluctuations of the flow (known as "cyclonic events"), the magnetic field in the conducting medium in the presence of a negligible magnetic diffusion generates an electromotive force parallel to the original magnetic field, which in turn can amplify and sustain the original magnetic field. This idea is in the heart of the theory of mean-field electrodynamics [Steenbeck *et al.*, 1966, Krause and Rädler, 1980]. The reader may refer to Rädler [2007] for its development. Similar phenomena were also presented in the theory of hydrodynamic [Dubrulle and Frisch, 1991](the anisotropic kinetic α-effect) and magnetohydrodynamic (MHD)[Zheligovsky, 2003] (the combined MHD α-effect) stability (see also [Zheligovsky, 2011]).

The emergence of the magnetic α-effect has also been observed in experiments. In the Karlsruhe experiment [Rädler *et al.*, 1998, Stieglitz and Müller, 2001, Müller and Stieglitz, 2002, Tilgner and Busse, 2002], a spiral flow of molten sodium arranged in pipes, whose diameter was smaller than the radius of the cylindrical volume, generated magnetic field;

The molten metal moved in opposite direction in the adjacent pipes. A similar flow was first considered by Roberts [1970] and then further investigated by Plunian and Rädler [2002], Rüdiger, Feudel and Seehafer [1998], Tilgner [1997]. All experiments exhibited a strong α-effect.

In the mean-field electrodynamics, it is postulated that the electromotive force is related to the mean magnetic field by a coefficient of proportionality, traditionally denoted by α. However, such a relation is yet to be fully justified [Chertovskih, 2015].

The concept of the magnetic α-effect is closely related to the concepts of scale separation and averaging over small spatial scales. In order to examine the influence of these effects, rigorous asymptotic methods can be used, provided there is significant scale separation; the ratio, ε, between the characteristic small and large spatial scales is a small parameter of the stability problem. By small scale fields, we refer to fields that depend only on fast spatial variables, $\mathbf{x} = (x_1, x_2, x_3)$, and time, t, and by large scale fields, we refer to fields that depend on slow spatial variables, $\mathbf{X} = \varepsilon(x_1, x_2)$ and on slow time, $T = \varepsilon^s t$, where $s \geq 1$.

Two approaches can be followed: on the one hand, we can study perturbations that depend on the fast and slow spatial variables, by expanding them in power series in ε and deriving a hierarchy of equations for the terms involved in the expansions. The mean parts of the terms satisfy partial differential equations in the slow variables that are derived as solvability conditions for the partial differential equations in the fast variables for the fluctuating parts. The equations for the averaged terms involve tensors of the respective α-effect or eddy diffusivity. In principle, they can be solved in any region of slow variables. This approach was applied to the kinematic dynamo in Rädler [2007], Rasskazov *et al.* [2018], Kosobokov [1995], Zheligovsky [2003], the hydrodynamic stability problem in Dubrulle and Frisch [1991] and the MHD stability problem in Zheligovsky [2003].

On the other hand, when addressing a large-scale linear stability problem of periodically-replicated small-scale regimes, we can focus on the mean fields that feature spatial periodicity in the slow variables. By linearity, the solution (a large-scale magnetic or stability mode, when the eigenvalue problem for a stability operator is considered) takes the form of a Bloch field $\mathrm{e}^{i\mathbf{q}\cdot\mathbf{x}}\mathbf{f}(\mathbf{x})$, where the wave vector $\mathbf{q}$ is small, $|\mathbf{q}| = \varepsilon$. Thus the amplitude-modulating exponential $\mathrm{e}^{i\mathbf{q}\cdot\mathbf{x}}$ can be negated and we end up with a parabolic (when the evolutionary equation is considered) or elliptic (in the case of an eigenvalue problem) equation depending only on fast variables. The remaining "carrier" factor $\mathbf{f}(\mathbf{x})$ (unlike in the AM radio signal transmission is not harmonic) can now be expanded in the asymptotic power series ε and this yields a solution equivalent to the one constructed by the general method for the same boundary conditions. Small-ε asymptotics of kinematic magnetic modes of the Bloch type was explored in Roberts [1970, 1972], Vishnik [1987]. In Zheligovsky and Chertovskih [2020, 2022], the size of the wave vector $\mathbf{q}$ of the modulating harmonics $e^{i\mathbf{q}\cdot\mathbf{x}}$ was not limited to smaller values; the goal of this work was to investigate which $\mathbf{q}$ maximizes the efficiency of the dynamo, i.e., for which $\mathbf{q}$ the dominant magnetic mode benefits the maximum growth rate. It is concluded that, at least for the range of magnetic molecular diffusivities that were considered, the most favorable for magnetic field generation are wave vectors involving moderate scale separation ($|\mathbf{q}| \sim 1/2$).

The dependence of rotation for magnetic field generation is an intriguing question in itself. We observe rotation in cosmic bodies and its effects on its surroundings. Several studies were performed to understand this relation between magnetic field and rotation. In Gertsenshtein *et al.* [2007], Gertsenshtein and Chertovskih [2008], Chertovskih *et al.* [2010], regimes of nonlinear magnetic dynamo powered by convection of electrically conducting fluid in a plane layer rotating above the vertical axis with square periodicity boxes were studied, for the Taylor number varying from 0 to 2000 with the other parameters fixed. The regimes are separated by short sequences of bifurcations from the onset of convection and have a simple spatial structure with the fluid motion in the form of distorted rolls. In Chertovskih *et al.* [2010], five branches of hydrodynamically stable amagnetic regimes were found: two-dimensional rolls of three types, travelling waves and three-dimensional "wavy" rolls. All regimes, except for a family of rolls, may act as kinematic magnetic dynamos and give rise to non-linear MHD steady states (13 branches) and time-periodic regimes (8 branches). Bifurcations in which these branches emerge and disappear were identified. The emerging complex regimes were also reported: two-frequency quasiperiodic and chaotic regimes, and a finite Feigenbaum period-doubling sequence of bifurcations of a torus superceded by a chaotic regime and by a torus with 1/3 of the cascade frequency. The symmetries of the regimes were listed *ibid.*

These regimes are stable to short-scale perturbations, i.e., perturbations of the same periodicity in horizontal directions as the regimes themselves. Large-scale weakly nonlinear perturbations of hydromagnetic convective regimes in a rotating layer were considered in Zheligovsky [2009] and it was unprecedented. The perturbations were assumed to depend on the slow horizontal and fast spatial variables. The asymptotic techniques which had earlier revealed the α-effect and eddy diffusivity in linear stability problems were used to derive the amplitude equations governing the perturbations of free convective dynamos (i.e., no external forcing influences the flow). In this case the kernel of linearisation involves two zero-mean neutral modes because of the spatial invariance of the regimes under perturbation. These results were applied in Chertovskih and Zheligovsky [2015] in order to investigate the stability of steady states from the branch $\mathrm{S}_8^{\mathrm{R}1}$ identified in Chertovskih *et al.* [2010] to weakly nonlinear large-scale perturbations. Numerical results from the system of amplitude equations revealed that a large-scale perturbation either converges to a small-scale (periodicity of the unperturbed steady state) neutral stability mode of constant amplitude or it blows up at a finite slow time.

The stability of the regimes found in Chertovskih *et al.* [2010] to perturbations involving moderate scale separation is yet to be investigated in the fully nonlinear setup. When considering a perturbation, initially proportional to $\mathrm{e}^{i\mathbf{q}\cdot\mathbf{x}}\mathbf{f}(\mathbf{x})$, it is impossible to exclude or negate the amplitude modulating harmonics in the equations governing the perturbed regime due to their nonlinearity. Consequently, the problem can be addressed only by direct numerical simulations with an adequate spatial resolution in the computations; this implies that in practice we can consider only perturbations and the emerging regimes, whose spatial periods are several times larger than the periods of the regime under perturbation.

When convective motions and magnetic field are permitted to have large spatial periods – here, in the horizontal directions – the resultant characteristic length scales of the fields is

expected to grow and give rise to higher effective kinematic and magnetic Reynolds number favourable for magnetic field generation. An objective of this project is to verify these expectations. Our quest also involves answering the question, whether the dynamo regimes under consideration are stable with respect to moderately scale-separated perturbations. If they are found to be unstable, is the evolution of perturbations involving the moderate scale-separation comparable with the prediction in Chertovskih and Zheligovsky [2015] for the evolution of weakly nonlinear perturbations featuring high scale separation, or, in line with the results of Zheligovsky and Chertovskih [2020, 2022], or do the moderately scale-separated perturbations follow their own paths in development?

In Donzis *et al.* [2013], a family of integral quantities D_q was studied numerically for four numerical solutions to the Navier-Stokes equation. All the four flows had been used previously for the study of turbulence. It was found that, for all of them, D_q increases in q for the considered values $q \leq 18$ – a result, that is yet to be understood analytically. In section 3.2.6, we numerically investigate these quantities and introduce similar ones, $D_{q,s}$.

This thesis is organized in a way that in the next two chapters we present the mathematical statement of the problem at hand and the numerical methods used to solve it. This is then followed by analysis of results from the computations and the conclusion.

In the Appendix, we present the parallel software code that was developed for our study and the tests that were performed to validate it.

Chapter 1

Statement of the problem

Fluids are made up of atoms. These atoms have kinetic energy. As they interact with one another molecular collisions and macroscopic motions occur. Mean free path (mfp) is the average distance by an atom between collisions. If this characteristic length scale between molecular collisions, denoted by l_{mfp}, is greater than the characteristic length scale of macroscopic motions, L_{hydro}, then a fluid may be regarded as a continuous field:

$$L_{\text{hydro}} \gg l_{\text{mpf}} \ .$$

In this case the motion of a macroscopic field can be described by partial differential equations for physical quantities such as density, energy and momentum. The evolution of such quantities describe the behaviour of a fluid.

To illustrate this, let us consider the concept of conservation of mass. Consider a volume C of fluid fixed in space and let S be the boundary of C. Let ρ be the density of a macroscopic fluid particle, $\mathbf{v}$ be its velocity. Let $d\mathbf{s}$ be an element of the surface pointing in the outward normal direction and, $|d\mathbf{s}|$ be its area.

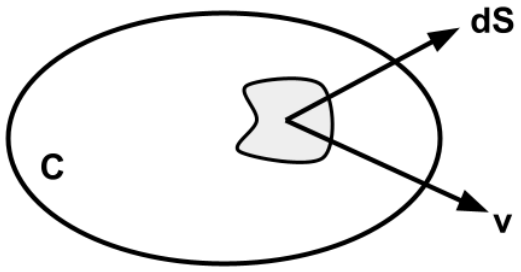


Figure 1.1 An illustration of a volume of fluid fixed in space.

The outward mass flux through the element $d\mathbf{s}$ is $\rho\mathbf{v} \cdot d\mathbf{s}$. Therefore, the rate of mass loss from C is $\int_s \rho\mathbf{v} \cdot d\mathbf{s}$. The total mass in C is $\int_C \rho \, dV$. Thus, the rate of mass loss can be written as

$$-\frac{d}{dt}\int_C \rho \, dV = -\int_C \frac{\partial \rho}{\partial t} \, dV = \int_S \rho\mathbf{v} \cdot d\mathbf{s} = \int_C \nabla \cdot (\rho\mathbf{v}) \, dV \ .$$

By considering an arbitrary infinitesimal volume of fluid C, the volume integrals can be eliminated and we obtain

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{v}) \,. \tag{1.1}$$

Equation (1.1) is called the equation of continuity and we see that the net divergence creates a corresponding change in density. Similar equations can be derived for energy and momentum.

Besides, the motion of atoms in a fluid also enable transfer of heat and mass. This phenomenon is called convection. If the transfer is caused by an external motion then it is referred to as forced convection. Thermal convection involves heat transfer and is called free if the fluid motion is caused by an external force such as gravity. The density variations because of temperature differences lead to instabilities, which are triggered by the buoyancy forces.

The behaviour of a convecting fluid is determined by its hydrodynamic variables: fluid velocity $\mathbf{v}(\mathbf{x}, t)$, density of mass $\rho(\mathbf{x}, t)$, pressure $p(\mathbf{x}, t)$ and temperature $T(\mathbf{x}, t)$. The equations that govern a hydrodynamic system are the Navier-Stokes equation (the equation of motion), the continuity equation, the energy equation (the heat transfer equation) and the equation of state. These equations are entirely general and can be simplified based on the physical properties of a system.

In this chapter we give a brief introduction to fluid mechanics and the equations of magnetohydrodynamics.

We use the following notation to refer to the vector and scalar quantities. A vector variable is represented by a bold lower case letter as in $\mathbf{a}$ and a scalar variable by a non-bold lower case letter as in a. In its explicit form, in the three-dimensional Euclidean space, a vector at a position, $\mathbf{x} = (x_1, x_2, x_3) = \sum_{i=1}^{3} \mathbf{e}_i x_i$, at an instant of time t, is represented by $\mathbf{a} = (a_1, a_2, a_3)$. Here, $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ is the canonical basis in the three-dimensional Euclidean space.

The real and imaginary parts of a complex number z are denoted by $\mathrm{Re}(z)$ and $\mathrm{Im}(z)$, respectively.

Standard differential operator notation is used for scalar and vector functions:

- The gradient of a scalar function a is denoted by ∇a, and divergence of a vector field $\mathbf{a}$ by $\nabla \cdot \mathbf{a}$.
- The Laplacian is represented by ∇^2 .
- The rotational, or curl, of a vector field $\mathbf{a}$ is indicated by $\nabla \times \mathbf{a}$.

1.1 Free convection

Free thermal convection involves the transfer of heat from one region to another with a fluid motion induced by the gravitational field. The basis of Boussinesq approximation is, (small) variations in density are due to (small) variations in temperature and buoyancy drives the fluid motion.

1.1.1 Boussinesq approximation

In order to present the governing equations of a convective system under Boussinesq approximation, we consider that gravitation is the only external force acting on the fluid and assume the following:

- Density is a linear function of temperature,

$$\rho = \rho_0(1 - \alpha(T - T_0)), \tag{1.2}$$

  where T_0 is a chosen mean temperature, ρ_0 is the density at T_0 and α is the thermal expansion coefficient. Its dependence on pressure is ignored, implying that the fluid is incompressible.

- The variations in density are ignored except for the term which represents the external force. In this case it is the buoyancy force. Thus, we represent the external force as

$$\rho_0(1 - \alpha(T - T_0))\mathbf{g}\,, \tag{1.3}$$

  where $\mathbf{g} = -g\,\mathbf{e}_3$ is the acceleration due to gravity.

- All parameters representing the physical properties of the fluid such as ν (kinematic viscosity), κ (thermal diffusivity), α are treated as constants.

With these assumptions, the effects of temperature due to viscous dissipation can be neglected [Chandrasekhar, 1961] and we introduce:
the Navier-Stokes equation,

$$\frac{\partial \mathbf{v}}{\partial t} = \nu\nabla^2\mathbf{v} - (\mathbf{v}\cdot\nabla)\mathbf{v} - \mathbf{g}\alpha T' - \frac{1}{\rho_0}\nabla p', \tag{1.4}$$

with $T' = T - T_0$ and p' as the perturbation of the hydrostatic pressure at T_0.
the heat transfer equation,

$$\frac{\partial T'}{\partial t} = \kappa\nabla^2 T' - (v\cdot\nabla)T', \tag{1.5}$$

and the solenoidal equation,

$$\nabla\cdot\mathbf{v} = 0. \tag{1.6}$$

In equations (1.4) and (1.5), we replace the non-dimensional quantities with their corresponding dimensional quantities - we drop the primes,

- $\mathbf{x} = d\mathbf{x}'$, where d is the characteristic length scale,
- $\mathbf{v} = \dfrac{\kappa}{d}\mathbf{v}'$,
- $p = \dfrac{\rho_0\kappa^2}{d^2}p'$,
- $T = \delta T T'$, where δT is the characteristic temperature difference,

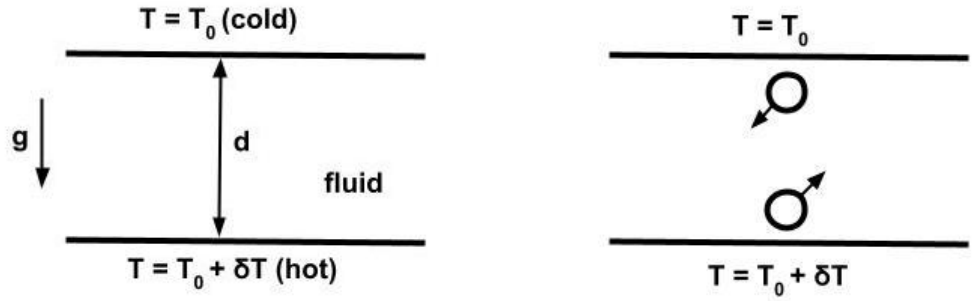
- $\tau_v = d^2/\kappa$, the diffusion time for temperature difference,

and consider $\mathbf{g} = -g\,\mathbf{e}_3 = (0, 0, -g)$ in order to obtain the dimensional form of the governing equations of convection:

$$\frac{\partial \mathbf{v}}{\partial t} = \nu \nabla^2 \mathbf{v} - (\mathbf{v} \cdot \nabla)\mathbf{v} + g\alpha T - \frac{1}{\rho_0}\nabla p, \tag{1.7}$$

$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T - (\mathbf{v} \cdot \nabla)T. \tag{1.8}$$

In Rayleigh-Bénard convection we study the flows caused by the rising hot fluid and the falling cold fluid. If the temperature gradient is small then no convection takes place. This is because of the stabilizing effects of viscous friction.



(a) (b)

Figure 1.2 An illustration of a fluid between two horizontal planes in Rayleigh Bénard convection.

For example, consider a small displacement of a cold blob downwards and a hot blob upwards as in Fig. 1.2(b). If there are no damping mechanisms such as heat diffusion or viscous friction, then the buoyancy force would allow the hot blob to continue rising and the cold blob to continue falling. In other words, convection occurs when the buoyancy force exceeds the dissipative effects of the viscous drag and heat diffusion.

Now, let us consider the effect of rotation on a convective system under Boussinesq approximation.

Consider a convecting fluid rotating about the vertical axis with a constant angular velocity $\mathbf{\Omega} = \Omega \mathbf{e}_3$. In order to describe the motion in the fluid we consider a frame of reference which is rotating about the same vertical axis and with the same angular velocity as the fluid. In this case two additional forces act on the rotating fluid: the Coriolis and the centrifugal [Chandrasekhar, 1961]. Thus, the Navier-Stokes equation (1.7) can be described in the dimensional form as

$$\frac{\partial \mathbf{v}}{\partial t} = \nu \nabla^2 \mathbf{v} - (\mathbf{v} \cdot \nabla)\mathbf{v} + g\alpha T - \frac{1}{\rho_0}\nabla p + 2\mathbf{v} \times \mathbf{\Omega} + \frac{1}{2}\nabla |\mathbf{\Omega} \times \mathbf{x}|^2. \tag{1.9}$$

The term $2\mathbf{v}\times\mathbf{\Omega}$ represents the Coriolis acceleration and the term $\frac{1}{2}\nabla|\mathbf{\Omega}\times\mathbf{x}|^2$ represents the centrifugal force. The heat transfer equation (1.8) and the solenoidal condition (1.6) remain the same.

The non-dimensional equations can be deduced by replacing the dimensional parameters.

So, the governing (non-dimensional) equations of convection in the presence of rotation are

$$\frac{\partial\mathbf{v}}{\partial t} = P\nabla^2\mathbf{v} - (\mathbf{v}\cdot\nabla)\mathbf{v} + PRT\mathbf{e}_3 + P\sqrt{Ta}\mathbf{v}\times\mathbf{e}_3 - \nabla\Big(p - \frac{1}{8}TaP^2|\mathbf{e}_3\times\mathbf{x}|^2\Big), \tag{1.10}$$

$$\frac{\partial T}{\partial t} = \nabla^2 T - (\mathbf{v}\cdot\nabla)T, \tag{1.11}$$

where

$$P = \frac{\nu}{\kappa}\ , \quad R = \frac{\alpha g\delta T d^3}{\nu\kappa}\ , \quad Ta = \frac{4\Omega^2 d^4}{\nu^2}\ . \tag{1.12}$$

The non-dimensional parameters are the Prandtl number, P (indicating the ratio of the fluid viscosity to thermal diffusivity), the Rayleigh number, R (characterising the magnitude of the thermal buoyancy forces) and the Taylor number, Ta (representing the angular velocity of the fluid rotation).

By using the vector identity

$$(\mathbf{v}\cdot\nabla)\mathbf{v} = \frac{1}{2}\nabla|\mathbf{v}|^2 - \mathbf{v}\times(\nabla\times\mathbf{v})\ ,$$

and by denoting the modified pressure $\tilde{p}$ as

$$\tilde{p} = p - \frac{1}{8}TaP^2|\mathbf{e}_3\times\mathbf{x}|^2 + \frac{\mathbf{v}^2}{2}\ , \tag{1.13}$$

we have

$$\frac{\partial\mathbf{v}}{\partial t} = P\nabla^2\mathbf{v} + \mathbf{v}\times(\nabla\times\mathbf{v}) + PRT\mathbf{e}_3 + P\sqrt{Ta}\mathbf{v}\times\mathbf{e}_3 - \nabla\tilde{p}\ . \tag{1.14}$$

Under the Boussinesq approximation, the non-dimensional equations (1.6), (1.11), (1.14) describe the dynamics of the convective system.

1.2 Convection in a plane layer

A simple convective system can be described as follows. Consider a fluid confined between two infinite horizontal planes. Let the fluid be heated uniformly from below and the temperatures at the bottom and top planes be T_1 and T_2 respectively. The planes are maintained at constant temperatures. Let the fluid layer rotate about the vertical axis x_3 with a constant angular velocity $\mathbf{\Omega}$, the direction of gravity coincide with $-\mathbf{e}_3$. The vertical size of the fluid layer is supposed to be 1.

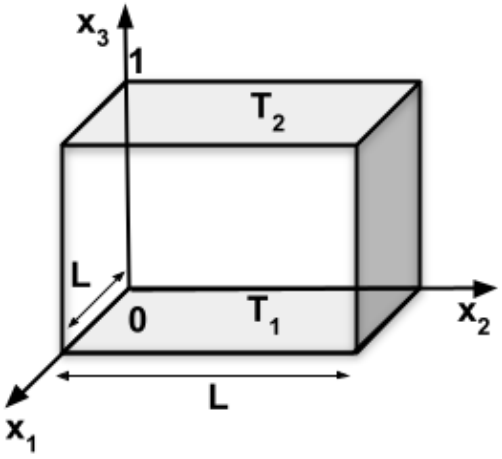


Figure 1.3 An illustration of a periodicity box in a horizontal layer.

The spatial domain considered here is a box (see Figure 1.3): $\mathscr{C} = [0, L] \times [0, L] \times [0, 1]$, where L is the horizontal size of the convective cell measured in units of layer width, and which is called as the aspect ratio.

Let $x_3 = 0$ and $x_3 = 1$ be the horizontal boundaries of the fluid layer.

$$T(\mathbf{x}, t)|_{x_3=0} = T_1, \quad T(\mathbf{x}, t)|_{x_3=1} = T_2, \quad T_1 > T_2.$$

We define a temperature profile,

$$\theta(\mathbf{x}, t) = T(\mathbf{x}, t) - (T_1 + (T_2 - T_1)x_3), \tag{1.15}$$

which represents the deviation of the temperature field during convection - from a steady state where the fluid flow is at rest and heat is only transferred by diffusion.

Now by using (1.15) in the Navier-Stokes (1.14) and in the heat transfer equation (1.11) we obtain (tilde is dropped in the pressure term),

$$\frac{\partial \mathbf{v}}{\partial t} = P\nabla^2\mathbf{v} + \mathbf{v} \times (\nabla \times \mathbf{v}) + PR\theta\mathbf{e}_3 + P\sqrt{Ta}\mathbf{v} \times \mathbf{e}_3 - \nabla p, \tag{1.16}$$

$$\frac{\partial \theta}{\partial t} = \nabla^2\theta - (\mathbf{v} \cdot \nabla)\theta + v_3, \tag{1.17}$$

representing a convective flow in a rotating plane layer.

To represent or model the fluid flow in the terrestrial objects, a spherical shell with fluid confined between two nested spheres seems a reasonable choice. But a plane layer with fluid enclosed between two infinite planes simplifies the complexity of the problem - both mathematically and computationally. A plane layer can be regarded as a segment of a spherical shell.

In the next section, we present the boundary conditions considered for v and θ in this work.

1.2.1 Boundary conditions

Suppose a fluid is confined between the horizontal planes, $x_3 = 0$ and $x_3 = 1$, which are maintained at constant temperatures. The vertical component of the velocity field is assumed to be zero, which indicates that the boundary either is impermeable or non-deformable by the fluid. In this case we consider the following conditions at the boundaries:

$$\theta\,|_{x_3=0,1} = 0 \quad \text{and} \quad v_3\,|_{x_3=0,1} = 0\,. \tag{1.18}$$

Rigid and free boundaries are the customary choices for the horizontal boundary conditions. On a rigid boundary, no-slip is considered. That is, all three components of the velocity vanish. On a free boundary the tangential stress vanishes, which suggests

$$S_{x_1x_3} = S_{x_2x_3} = 0\,. \tag{1.19}$$

Here, $S_{x_ix_j}$ is the stress acting in the direction of x_j per unit area on an element of the boundary normal to x_i. A basic assumption in fluid dynamics is that the stress, $S_{x_ix_j}$, and the rate of increase of strain, $\epsilon_{x_ix_j}$, given by

$$\epsilon_{x_ix_j} = \frac{1}{2}\Big(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i}\Big)\,, \tag{1.20}$$

are linearly related [Chandrasekhar, 1961]. For an isotropic fluid [Frisch, 1995],

$$S_{x_ix_j} = -s\delta_{x_ix_j} + 2\vartheta\epsilon_{x_ix_j} + \lambda\delta_{x_ix_j}\epsilon_{x_kx_k}\,, \tag{1.21}$$

where s, λ, and ϑ are scalar functions of x_i. Since the isotropic tensor, $-s\,\delta_{x_ix_j}$, has no transverse component, the conditions in (1.19) are equivalent to the vanishing of the viscous stress components $s_{x_1x_3}$ and $s_{x_2x_3}$ [Chandrasekhar, 1961]:

$$s_{x_1x_3} = \vartheta\Big(\frac{\partial v_1}{\partial x_3} + \frac{\partial v_3}{\partial x_1}\Big) \quad \text{and} \quad s_{x_2x_3} = \vartheta\Big(\frac{\partial v_2}{\partial x_3} + \frac{\partial v_3}{\partial x_2}\Big)\,, \tag{1.22}$$

where ϑ is the coefficient of viscosity. Also, since $v_3 = 0$ on the boundary, it follows from (1.22),

$$\frac{\partial v_1}{\partial x_3}\bigg|_{x_3=0,1} = \frac{\partial v_2}{\partial x_3}\bigg|_{x_3=0,1} = 0\,, \tag{1.23}$$

for a free boundary.

So, we assume that the horizontal boundaries are stress free. The boundary conditions (1.23) for the horizontal components are referred to as the Neumann boundary conditions and for the vertical components (1.18) are referred to as the Dirichlet boundary conditions. For θ, however, an uniform boundary condition is considered.

1.3 Free convection in the presence of a magnetic field

The subject of hydromagnetics is concerned with how a magnetic field can affect a fluid behaviour. Its presence along with the effects of rotation do impart certain properties to the fluid. As an example, consider a fluid which has the property of electrical conduction. Let also suppose that magnetic field is present. These two contribute to two kinds of effects: First, electric currents are produced by the motion of electrically conducting fluid as it traverses across the magnetic lines of force and the associated magnetic field affects the existing fields. Second, the fluid elements with the electric currents traversing the magnetic lines of force contribute to additional forces acting on the fluid elements [Chandrasekhar, 1961]. This twofold interaction between the fluid and the magnetic field is striking.

The basic equations representing these interactions are written using the Maxwell's equations with some suitable modifications. One such approximation is to ignore the displacement currents and the currents associated with a charge separation in the Maxwell's equations. Under these considerations the Maxwell's equations are given by

$$\nabla \cdot \mathbf{b} = 0\,, \tag{1.24}$$

$$\nabla \times \mathbf{b} = 4\pi \mathbf{J}\,, \tag{1.25}$$

$$\nabla \times \boldsymbol{\epsilon} = -\mu \frac{\partial \mathbf{b}}{\partial t}\,, \tag{1.26}$$

where $\boldsymbol{\epsilon}$ and $\mathbf{b}$ are the electric and magnetic fields, $\mathbf{J}$ is the current density, and μ is the magnetic permeability.

The equation of current density is given by,

$$\mathbf{J} = \sigma(\boldsymbol{\epsilon} + \mu \mathbf{v} \times \mathbf{b})\,. \tag{1.27}$$

In the above representation, σ is the electrical conductivity. The equations (1.24)–(1.27) are the basic equations of the electromagnetic field for hydromagnetics. These incorporate the effect of fluid motions on the electromagnetic field.

On the other hand, the effect of the field on the motion is from the force experienced by the fluid elements by virtue of the currents they carry across the magnetic field. This is the Lorentz force, given by

$$\mathcal{L} = \mu \mathbf{J} \times \mathbf{b}\,. \tag{1.28}$$

We shall now deduce the equation of the magnetic field. According to equation (1.27),

$$\boldsymbol{\epsilon} = \frac{1}{\sigma}\mathbf{J} - \mu \mathbf{v} \times \mathbf{b}\,,$$

and by making use of (1.25), we have

$$\boldsymbol{\epsilon} = \frac{1}{4\pi\sigma}\nabla \mathbf{b} - \mu \mathbf{v} \times \mathbf{b}\,.$$

Also by inserting the above expression for $\boldsymbol{\epsilon}$ in (1.26), we obtain

$$\frac{\partial \mathbf{b}}{\partial t} - \nabla \times (\mathbf{v} \times \mathbf{b}) = -\nabla \times (\eta \nabla \times \mathbf{b}), \tag{1.29}$$

where $\eta = \dfrac{1}{4\pi\mu\sigma}$, is referred to as the magnetic diffusivity. Equation (1.29) is a general equation and it differs from the usual definition by a factor of $1/4\pi$. It is also noted that this equation is not restricted to incompressible fluids.

The non-dimensional magnetic induction equation is obtained by replacing the dimensional parameters for length, time and velocity and by choosing

$$\mathbf{b} = \frac{\kappa\sqrt{\mu}}{d}\,\mathbf{b}'.$$

Thus the final magnetic induction equation is given by

$$\frac{\partial \mathbf{b}}{\partial t} = \nabla \times (\mathbf{v} \times \mathbf{b}) + \frac{P}{P_m}\nabla^2\mathbf{b}. \tag{1.30}$$

Here, $P_m = \dfrac{\nu}{\eta}$ is the magnetic Prandtl number that measures the magnetic properties of the fluid. The advective term in (1.30) can be expressed as

$$\nabla \times (\mathbf{v} \times \mathbf{b}) = (\mathbf{b} \cdot \nabla)\mathbf{v} - (\mathbf{v} \cdot \nabla)\mathbf{b}. \tag{1.31}$$

Here, $(\mathbf{b} \cdot \nabla)\mathbf{v}$ describes the distortion of the magnetic field by the fluid flow, and $(\mathbf{v} \cdot \nabla)\mathbf{b}$ can be interpreted as the transport of the magnetic field by the fluid flow.

By substituting (1.27) in (1.28), the Lorentz force can be expressed as

$$\mathcal{L} = \frac{\mu}{4\pi}(\nabla \times \mathbf{b}) \times \mathbf{b} = -\frac{\mu}{4\pi}\mathbf{b} \times (\nabla \times \mathbf{b}), \tag{1.32}$$

and by adding it to the Navier-Stokes equation (1.16), we obtain

$$\frac{\partial \mathbf{v}}{\partial t} = P\nabla^2\mathbf{v} + \mathbf{v} \times (\nabla \times \mathbf{v}) + PR\theta\mathbf{e}_3 + P\sqrt{Ta}\mathbf{v} \times \mathbf{e}_3 - \mathbf{b} \times (\nabla \times \mathbf{b}) - \nabla p. \tag{1.33}$$

In the absence of magnetic field the convective system is governed by the equations (1.16) and (1.17) together with (1.6).

1.3.1 Boundary conditions

For the magnetic field, the boundary conditions depend on the electrical properties of the medium adjoining the fluid. We shall consider two cases.

If the medium adjoining the fluid is electrically non-conducting then no currents can traverse the boundary. So it is necessary that $J_3 = 0$. Moreover, the field, $\mathbf{b}$, outside the layer must be appropriate to a vacuum and be derivable from a potential [Chandrasekhar, 1961]. Thus,

$$\mathbf{b} = \nabla\psi, \quad \text{where} \quad \nabla^2\psi = 0;$$

and on the interface between the fluid and the medium the field **b** must be continuous.

If the medium adjoining the fluid is a perfect conductor then no magnetic field can cross the boundary. Hence, we must require

$$b_3 = 0 \quad \text{and} \quad \epsilon_1 = \epsilon_2 = 0, \tag{1.34}$$

on a plane boundary adjoining a perfect conductor. This is the case of a rigid boundary and there will be no motions at the boundary. The conditions (1.34) are equivalent to

$$b_3 = 0 \quad \text{and} \quad J_1 = J_2 = 0. \tag{1.35}$$

We consider perfectly electrically conducting boundaries for the magnetic field, **b** . Therefore, the boundary conditions are

$$\left.\frac{\partial b_1}{\partial x_3}\right|_{x_3=0,1} = \left.\frac{\partial b_2}{\partial x_3}\right|_{x_3=0,1} = b_3\Big|_{x_3=0,1} = 0\,. \tag{1.36}$$

1.4 Convective cells

In this study, we consider a convective cell (see Figure 1.3) that has periods L in the horizontal directions x_1 and x_2 and construct convective cells with periods M_1L and M_2L, where M_1 and M_2 are positive integers. Therefore, the solutions of a MHD system have periods M_1L and M_2L in x_1 and x_2, respectively, i.e.,

$$\mathbf{v}(\mathbf{x}) = \mathbf{v}(x_1 + m_1M_1L, x_2 + m_2M_2L, x_3), \tag{1.37}$$

$$\mathbf{b}(\mathbf{x}) = \mathbf{b}(x_1 + m_1M_1L, x_2 + m_2M_2L, x_3), \tag{1.38}$$

$$\theta(\mathbf{x}) = \theta(x_1 + m_1M_1L, x_2 + m_2M_2L, x_3), \tag{1.39}$$

for all integers m_1 and m_2, $(x_1, x_2) \in \mathbb{R}^2$, and $x_3 \in [0, 1]$.

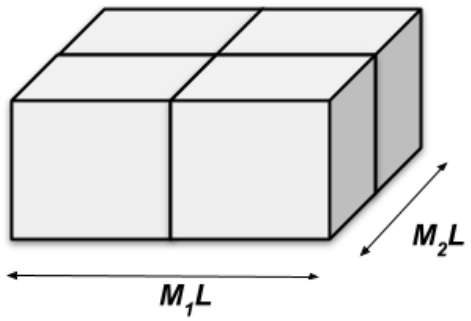


Figure 1.4 A sketch of the convective layer with $(M_1, M_2) = (2, 2)$.

All pair of integers (M_1, M_2) in the range $1 \le M_1, M_2 \le 4$ are considered and this corresponds to the scale ratio in the interval $0.25 \le \varepsilon \le 1$. We define

$$\varepsilon = \frac{L}{\max(M_1L, M_2L)} = \frac{1}{\max(M_1, M_2)}\ ,$$

as the scale separation parameter; l is the horizontal width of the one-box convective cell, and L is the horizontal size (relative the layer width) of the periodicity cell of the unperturbed convective dynamo regime whose value equals $2\sqrt{2}$, corresponding to the horizontal wave number $k = \pi/\sqrt{2}$. On increasing the Rayleigh number in a non-rotating layer, this hydrodynamic mode is the first to become unstable when convective flow sets in.

We regard the geometries of the (M_1, M_2)-periodicity and (M_2, M_1)-periodicity boxes as identical, but the computations in such geometries are distinct. This is because of the significant anisotropic structures of the regime in the horizontal directions subject to perturbations and the lack of the symmetry of rotation by $\pi/2$ in the arrangement of horizontal directions x_1 and x_2. In Figure 1.5, the distorted roll structures of the flow and the associated magnetic structures of the small-scale periodically replicated regime are shown. Thus, small perturbations result in different initial conditions and may reveal different attractors in the same dynamical system.

We employ the same dimensionless parameters as in Podvigina [2006] and Chertovskih [2010]:

$$P = 1, \quad P_m = 8, \quad R = 2300, \quad L = 2\sqrt{2}.$$

and study perturbations of the steady-state dynamo from the branch $\mathrm{S}_8^{\mathrm{R1}}$ [Chertovskih, 2010] for $Ta = 675$. Since the Rayleigh number is close to the critical value for the onset of convection, convective attractors have a simple roll structure. Similarly, the chosen magnetic Prandtl number is close to its critical value for the onset of kinematic magnetic field generation by convective flows and therefore flows in the convective MHD attactors are not significantly altered by the Lorentz force. In [Podvigina 2006, 2008], the dependence of the Rayleigh and Prandtl numbers on magnetic field generation by convection was examined. It was found that the critical P_m in general decreases on increasing R over the critical value, but over a certain threshold of R, this behaviour is non-monotonic.

Besides we consider that the mean velocity, the mean magnetic field and the mean temperature profile vanish:

$$\langle \mathbf{v} \rangle = \langle \mathbf{b} \rangle = \langle \theta \rangle = 0. \tag{1.40}$$

Here,

$$\langle f \rangle = \frac{1}{M_1\, M_2\, L^2} \int_{\mathscr{C}} f(\mathbf{x})\, d\mathbf{x},$$

denotes the spatial mean over the (M_1, M_2)-periodicity box. Spatial mean is an essential condition in convective flows. It is imposed at the initial stage and it holds true over time.

1.5 Summary

In this chapter, we discussed the following:

- the Boussinesq approximation and the governing equations of convection in the absence and presence of rotation;

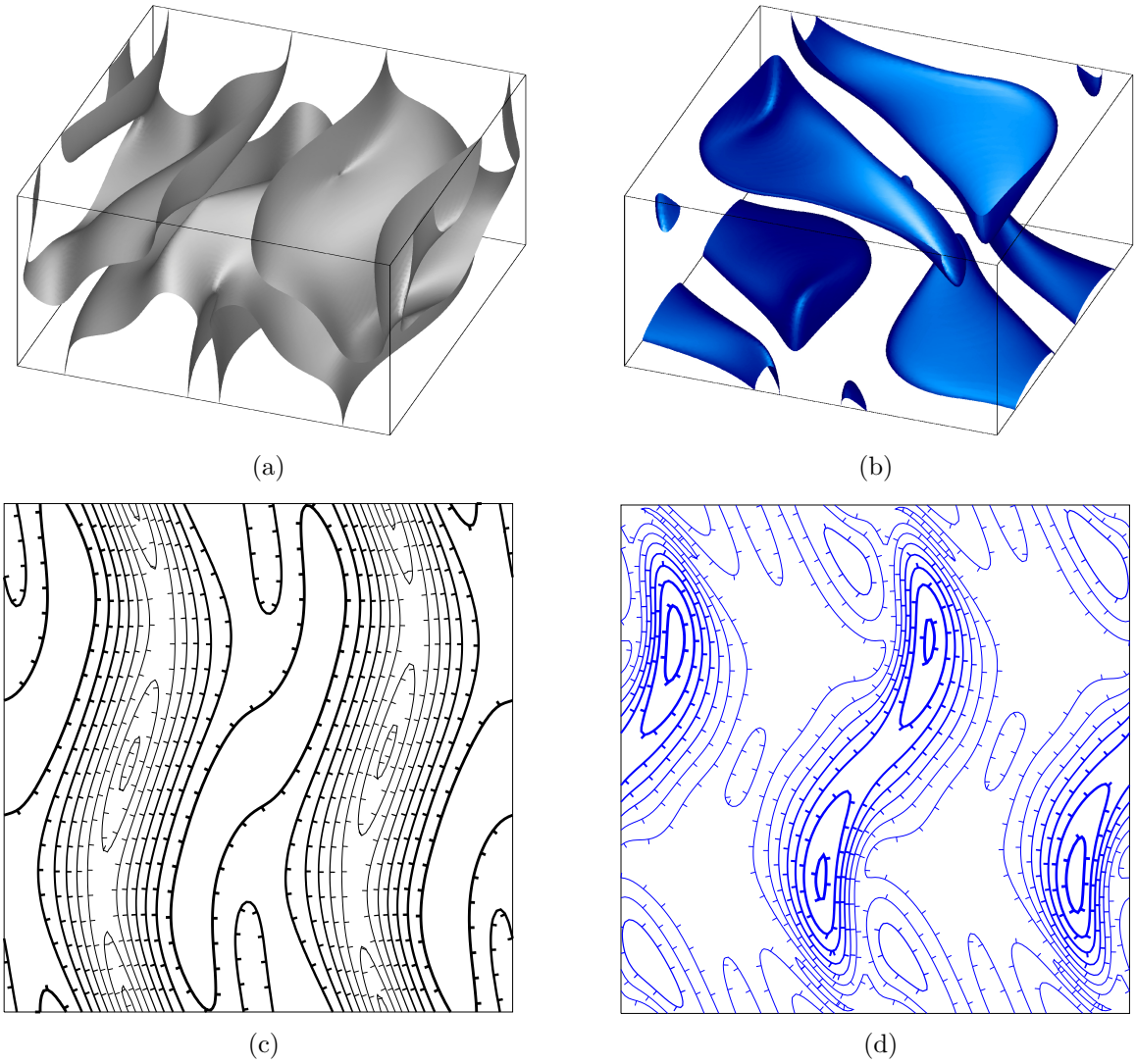


Figure 1.5 Isosurfaces of the kinetic (a) and magnetic (b) energy densities at the levels of 1/2 of the respective maxima of the short-scale convective dynamo regime subject to perturbations. Isolines of the kinetic (c) and magnetic (d) energy densities at the midplane $x_3 = 0$, step 1.5 and 0.3 respectively. In (c) and (d) the line width increases along the value of the energy density.

- the Rayleigh-Bénard convection with an example;
- the stress-free perfectly electrically conducting horizontal boundaries for the fluid-flow, $\mathbf{v}$ and the magnetic field, $\mathbf{b}$. However an uniform boundary for θ is considered;
- a convective flow in a plane layer;
- how to construct a convective layer from a square periodicity cell and the dimensionless parameters considered in our computations.

In the following chapter, we will look at some numerical methods that can be used to solve the convective hydromagnetic equations and the software tools used for the development of a parallel code in Fortran.

Chapter 2

Numerical methods and Software libraries

The space and time-dependent problems that model real-life scenarios are usually expressed in terms of *partial differential equations* (PDEs). Their geometries are complex and the vast majority cannot be solved by analytical methods. Instead, an approximation of the PDEs is constructed using different discretization methods. These methods approximate the PDEs with numerical model equations and the solutions to these model equations, in turn, are an approximation of the real solution to the PDEs.

The most frequent discretization methods are the finite difference, finite element, finite volume and spectral methods.

Finite difference is one of the oldest and simplest methods to solve differential equations. Its principle is to approximate the differential operator by replacing the derivatives in the equation using differential quotients. The approximations of the solution are computed in space using a finite part of a Taylor series. The error between the numerical and the exact solution is the error committed in the approximation of the derivatives.

Finite volume is another popular numerical method for approximations. It is used to solve elliptic, parabolic or hyperbolic PDEs. Its domain is divided into a finite number of elements which is either structured or unstructured. Subsequently each element is used to build cells (or control volumes). Thus a dual mesh is formed. Integration of the PDE is perfomed over each control volume and each results in a balance equation. The set of balance equations is then approximated by numerical integration. Since this method is based on the integral form of the balance equations, more accuracy/stability can be obtained for larger derivatives (sharp gradients) in a domain [Neill, 2018].

In the finite element method, a dependent variable in a PDE, say u, is approximated by u^N using a linear combination of basis functions

$$u \approx u^N, \quad u^N = \sum_i u_i \, \psi_i \, .$$

Here, ψ_i denote the basis functions and u_i denote the coefficients that approximate u with u^N. The method consists of assuming piecewise continuous functions for the basis and determining the coefficients in a manner that reduces the error.

Spectral methods also belong to a class of weighted residual methods where approximations are represented by a truncated series expansion [Peyret, 2002]. These approximations are defined for a given function or for a solution of a differential equation. In such cases, the error or the discrepancy which should be exactly zero is forced to zero in an approximate sense through the below scalar product

$$(u, v)_w = \int_a^b u(x)\, v(x)\, w(x)\, dx\,, \tag{2.1}$$

where $u(\cdot)$ and $v(\cdot)$ are functions defined on the interval $[a, b]$ and $w(\cdot)$ is a weight function. The given function is represented as a linear combination of simpler basis functions.

In this chapter, we present the concept of the spectral methods, its various formulations, the Fourier series in discrete form, their sine and cosine versions, a phenomenon called aliasing, which arises in the Fourier computations and aliasing removal. With the aforesaid, we construct the three dimensional Fourier representations of $(\mathbf{v}, \mathbf{b}, \theta)$. We also brief about a neat procedure on how to produce a smooth solenoidal field.

In the latter part of this chapter, the computing tools such as the time discretization method, ETD3RK, which is used for computations, symmetries for analyzing the solutions of a dynamical system, software libraries, FFTW and P3DFFT, which are used in the numerical simulations are discussed.

2.1 Spectral methods

2.1.1 Generalities

Consider a real-valued function $u(\cdot)$ defined on the interval $[a,\, b]$. Let $\{\varphi_k(\cdot)\}_{k=0}^{\infty}$ be a family of orthogonal functions on $[a,\, b]$ relative to the weight function, $w(\cdot) \geq 0$ such that

$$(\varphi_k, \varphi_l)_w = \int_a^b \varphi_k(x)\, \varphi_l(x)\, w(x)\, dx = c_k\, \delta_{k,l}, \tag{2.2}$$

where c_k is a constant and $\delta_{k,l}$ is the Kronecker delta given by

$$\delta_{k,l} = \begin{cases} 1 & \text{if} \quad k = l, \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

Our aim is to determine a linear combination,

$$u^N(x) = \sum_{k=0}^{N} \hat{u}_k\, \varphi_k(x), \quad a \leq x \leq b, \tag{2.4}$$

of the first $N+1$ basic functions, $\{\varphi_k\}_{k=0}^{\infty}$, that best approximate $u(x)$ on $[a, b]$. We emphasize as *best* because an approximation method is defined by the choice of basis functions, the manner in which the coefficients are determined and the weight.

Following the notation presented in Peyret [2002], we define the residual as

$$R_N(x) = u(x) - u^N(x). \tag{2.5}$$

We recall some of the well-known spectral methods in the below section. For more information on these methods we refer the reader to Peyret [2002].

2.1.2 The Galerkin-type method

This method is characterized by the fact that the residual $R_N(\cdot)$ is orthogonal to $\{\varphi_i(\cdot)\}_{i=0}^{N}$. That is,

$$\begin{aligned}
(R_N, \varphi_i)_w = 0, \quad i = 0, 1, \cdots, N \quad &\Longleftrightarrow \\
\int_a^b \left(u(x) - u^N(x)\right) \varphi_i(x)\, w(x)\, dx = 0, \quad i = 0, 1, \cdots, N \quad &\Longleftrightarrow \\
\int_a^b \left(u(x) - \sum_{k=0}^{N} \hat{u}_k\, \varphi_k(x)\right) \varphi_i(x)\, w(x)\, dx = 0, \quad i = 0, 1, \cdots, N \quad &\Longleftrightarrow \\
\sum_{k=0}^{N} \hat{u}_k \int_a^b \varphi_k(x)\, \varphi_i(x)\, w(x)\, dx = \int_a^b u(x)\, \varphi_i(x)\, w(x)\, dx, \quad i = 0, 1, \cdots, N.&
\end{aligned}$$

By using the orthogonality relation (2.2), the above equality reduces to

$$\hat{u}_i \int_a^b \varphi_i^2(x)\, w(x)\, dx = \int_a^b u(x)\, \varphi_i(x)\, w(x)\, dx\,,$$

$$\hat{u}_i = \frac{\int_a^b u(x)\, \varphi_i(x)\, w(x)\, dx}{\int_a^b \varphi_i^2(x)\, w(x)\, dx} = \frac{(u,\, \varphi_i)_w}{(\varphi_i, \varphi_i)_w}\,. \tag{2.6}$$

Thus, the coefficients $\hat{u}_i$ are the coefficients of u with respect to the orthogonal base $\{\varphi_k\}_{k=0}^{\infty}$.

2.1.3 The collocation method

In the collocation method, the residual $R_N(x)$ is set equal to zero at $N+1$ fixed points (collocation points), $x_i \in [a,\, b]$, $i = 0, 1, \cdots, N$, so that

$$\begin{aligned}
u^N(x_i) = u(x_i), \quad i = 0, 1, \cdots, N \quad &\Longleftrightarrow \\
\sum_{k=0}^{N} \hat{u}_k\, \varphi_k(x_i) = u(x_i), \quad i = 0, 1, \cdots, N&
\end{aligned} \tag{2.7}$$

which is an algebraic system of $N+1$ equations with $N+1$ unknown coefficients, $\hat{u}_k$. The existence of an unique solution to this system is equivalent to $\det\{\varphi_k(x_i)\} \neq 0$.

In the collocation method, the orthogonality relation expressed in (2.2) is replaced by an equivalent relation in discrete form

$$\sum_{i=0}^{N} \varphi_k(x_i)\,\varphi_l(x_i)\,w(x_i) = c_k\,\delta_{k,l}. \tag{2.8}$$

The coefficients, $\hat{u}_k$, can also be determined explicitly without solving the system (2.7) by using the above orthogonality relation (2.8): we multiply both sides of (2.7) by $\varphi_l(x_i)\,w(x_i)$ and sum over i to obtain

$$\sum_{k=0}^{N} \hat{u}_k \underbrace{\sum_{i=0}^{N} \varphi_k(x_i)\,\varphi_l(x_i)\,w(x_i)}_{c_k\,\delta_{k,l}} = \sum_{i=0}^{N} u(x_i)\varphi_l(x_i)\,w(x_i) \quad \Longleftrightarrow$$

$$\hat{u}_l \sum_{i=0}^{N} \varphi_l^2(x_i)\,w(x_i) = \sum_{i=0}^{N} u(x_i)\varphi_l(x_i)\,w(x_i) \quad \Longleftrightarrow$$

$$\hat{u}_l = \frac{\sum_{i=0}^{N} u(x_i)\varphi_l(x_i)\,w(x_i)}{\sum_{i=0}^{N} \varphi_l^2(x_i)\,w(x_i)}\,, \quad l = 0, 1, 2, \ldots, N. \tag{2.9}$$

2.1.4 The method of least-squares

In the method of least-squares, the coefficients $\hat{u}_k$ are determined by minimizing the scalar product $J = (R_N, R_N)_w$. It is achieved by setting the partial derivatives of J to zero with respect to the coefficients $\hat{u}_k$. In other words, J attains a minimum, if

$$\frac{\partial J}{\partial \hat{u}_k} = 0, \quad k = 0, 1, \cdots, N\,.$$

If the residual R_N is continuous, then

$$J = \int_a^b \left(u(x) - \sum_{k=0}^{N} \hat{u}_k\,\varphi_k(x) \right)^2 w(x)\,dx\,, \tag{2.10}$$

and its partial derivative

$$\frac{\partial J}{\partial \hat{u}_k} = -2 \int_a^b \left(u(x) - \sum_{k=0}^{N} \hat{u}_k\,\varphi_k(x) \right) w(x)\,dx. \tag{2.11}$$

Thus, in the continuous case, it is a Galerkin-type method. If R_N is discrete, then this method can be tagged as a collocation type method.

2.1.5 The Fourier method

One of the familiar spectral methods is the Fourier method. It is prevalent in periodic problems. Spatial periodicity appears in a variety of physical applications either in one or two directions and so Fourier series can be used for spatial approximations in such cases. Evidently the basis functions used are trigonometric functions.

To assume periodicity is necessary for a Fourier series approximation because if it is not considered then the Gibbs oscillations contaminate the solution. Though efficient methods to remove the Gibbs phenomenon exist Fourier series should be restricted to the representation of smooth, periodic functions [Peyret, 2002].

For non-periodic problems, orthogonal polynomials such as Chebyshev or Legendre, also constitute a proper basis. The Chebyshev series expansion, in particular, is exempt from Gibbs phenomenon at the boundaries. We refer the reader to Peyret [2002] for a general view of Chebyshev polynomials, to Gottlieb and Orszag [1977] and Canuto *et al.* [2006] for properties, methods, and use of Legendre polynomials.

With that all being said, Fourier approximations do come into picture for non-periodic problems provided the basis functions satisfy the boundary conditions. We discuss this in section 2.1.5.4.

Before we recount the Fourier method let us consider some facts which are vital to its application.

If $u(\cdot)$ is a function defined on $[0, 2\pi]$ and is absolutely integrable, then

$$\int_0^{2\pi} |u(x)|\, dx < +\infty. \tag{2.12}$$

Let $f : [a, b] \to \mathbb{R}$ be a function and we consider $a = x_1 \leq x_2 \leq \ldots \leq x_n = b$. We say that the total variation of f on the interval $[a, b]$ is the quantity

$$V = \sup_P \sum_{i=1}^{n} |f(x_i) - f(x_{i-1})| \tag{2.13}$$

where the supremum runs over the set of all partitions $P = \{x_0, x_1, \ldots, x_n\}$.

2.1.5.1 Fourier series

Consider a function $u : [0, 2\pi] \to \mathbb{C}$ approximated by an expansion of the form

$$\sum_{k=-\infty}^{\infty} \tilde{u}_k\, e^{\alpha i k x}, \tag{2.14}$$

where $i^2 = -1$ and $\{e^{\alpha i k x}\}_k$ as the basis functions. The orthogonality property of the complex exponential functions is given by

$$\int_0^{2\pi} e^{i\,k\,x} e^{-i\,l\,x} dx = \begin{cases} 2\pi, & \text{if } k = l \\ 0, & \text{if } \ k \neq l \end{cases} \tag{2.15}$$

The coefficients, $\tilde{u}_k$, are referred to as Fourier coefficients or Fourier modes in the Fourier method and are computed as below.

The residual, $R_N = u - u^N$ is orthogonal to $e^{\alpha ikx}$, and the weight function, $w(x) = 1$:

$$\begin{aligned}(u(x) - u^N(x), e^{\alpha ikx}) = 0 \iff \\ \int_0^{2\pi} (u(x) - u^N(x))\, e^{-\alpha ikx}\, dx = 0 \iff \\ \int_0^{2\pi} \left(u(x) - \sum_{l\in\mathbb{Z}} \tilde{u}_l\, e^{\alpha ilx} \right)\, e^{-\alpha ikx}\, dx = 0\,.\end{aligned}$$

Using the orthogonality relation (2.15) and rearranging the terms, we have

$$\tilde{u}_k = \frac{1}{2\pi} \int_0^{2\pi} u(x)\, e^{-\alpha ikx}\, dx. \tag{2.16}$$

Thus under certain conditions of regularity we have the Fourier transform (2.14) and its inverse (2.16).

The truncated Fourier series approximation of the function $u(\cdot)$ is given by

$$u^N(x) = \sum_{k=-N/2}^{N/2} \tilde{u}_k\, e^{\alpha ikx}, \tag{2.17}$$

where N is a positive integer and $k \in \mathbb{Z}$ represents the wave numbers.

Suppose we know the values of the function u only at certain fixed points, $x_j = \frac{2\pi}{N} j$, $j \in \{0, 1, 2, \cdots, N-1\}$. These points are known as the collocation points. Since u is periodic, we have $u(x_0) = u(x_N)$, and this leaves us with N data points for computation. The truncated Fourier series at these points can be represented as

$$\begin{aligned}u^N(x_j) &= \sum_{k=-K}^{K} \tilde{u}_k\, e^{\alpha ikx_j} \\ &= \sum_{k=-K}^{K} \tilde{u}_k\, e^{ik2\pi j/N}, \quad j = 0, 1, \cdots, N-1.\end{aligned} \tag{2.18}$$

Apparently, we have N values for u^N and $2K + 1$ values for the coefficients $\tilde{u}_k$.

Let us now denote the periodicity of $u(\cdot)$ by L and assume K to be an even natural number (the case $K = 2^n, n \in \mathbb{N}$, corresponds to the fastest FFT algorithm [Cooley and Tukey, 1965]. Refer to section 2.6.1 for more details). Then the error of approximation between the infinite series (2.14) and the truncated series (2.17) is bounded by the magnitude of the truncated modes. That is,

$$\int_0^L \left(u(x) - u^N(x) \right)^2 dx = L \sum_{|k|>N/2} |\, \tilde{u}_k\, |^2\,. \tag{2.19}$$

If the Fourier coefficients $\tilde{u}_k$ in (2.17) can be approximated by the trapezoidal rule in an uniform grid $x_j = \frac{L}{N}j$, $j = 0, 1, \ldots, N-1$, then by using the below formula for quadrature,

$$\frac{1}{L}\int_0^L f(x)\,dx = \frac{1}{N}\sum_{j=0}^{N-1} f(x_j), \tag{2.20}$$

we have

$$\tilde{u}_k \approx \hat{u}_k = \frac{1}{N}\sum_{j=0}^{N-1} u(x_j)\, e^{\alpha i k x_j} = \frac{1}{N}\sum_{j=0}^{N-1} u_j\, e^{-2\pi i k j/N}, \quad u_j = u(x_j). \tag{2.21}$$

This quadrature (2.20) is exact for trigonometric polynomials, $\varphi_l(x) = e^{\alpha i l x}$, $|l| < N$, $l \in \mathbb{Z}$. However, for $e^{\pm\alpha i N x}$, these values are not equal.

$$\frac{1}{L}\int_0^L e^{\pm\alpha i N x} dx = 0 \quad \neq \quad \frac{1}{N}\sum_{j=0}^{N-1} e^{\pm\alpha i N x_j} = 1.$$

Therefore the Fourier coefficients for the wave number $k = \pm N/2$, which corresponds to the Nyquist frequency,

$$\tilde{u}_{\pm\frac{N}{2}} = \frac{1}{L}\int_0^L u(x)\, e^{\pm\alpha i \frac{N}{2} x} dx$$

cannot be computed accurately by (2.21) and we exclude $\hat{u}_{\pm\frac{N}{2}}$ in the computations by setting them to zero. For a sufficiently large number of Fourier modes, the approximation error depends only on the regularity of the function [Peyret, 2002].

Eventually, with these above changes the discrete complex Fourier transforms (direct and inverse) of u can be represented using the equalities below:

$$\hat{u}_k = \frac{1}{N}\sum_{j=0}^{N-1} u_j e^{-2\pi i k j/N}, \quad u_j = \sum_{k=-N/2+1}^{N/2-1} \hat{u}_k e^{2\pi i k j/N}. \tag{2.22}$$

A direct implementation of either one of the above sums requires $O(N^2)$ operations. If the number of grid points can be split, $N = p \times q$, then the transformation can be split into two separate transformations. By performing this operation recursively we obtain an algorithm called Fast Fourier Transform (FFT) [Brigham, 1974, Boyd, 2001, Press *et al.*, 2002]. This algorithm requires just $O(N \log_2 N)$ operations.

Further, computing non-linearities in the Fourier space corresponds to evaluating convolutions which are expressed as sums of Fourier coefficients over all pairs of wave numbers. Again a direct summation is inefficient here. Nonetheless FFT provides a way out: FFT traverses across to the physical space, performs the computations and then switches back to the Fourier space. This procedure is referred to as the pseudo-spectral method.

If $u(x)$ is a real-valued function, then the two Fourier coefficients, with an opposite value of k are complex conjugates.

$$\tilde{u}_k = (\tilde{u}_{-k})^*, \quad \forall k, \tag{2.23}$$

where * indicates the complex conjugate. In practical calculations, only about half of the Fourier coefficients are needed to reconstruct a real-valued function.

The real form of Fourier series of $u(x)$ is given by

$$\tilde{a}_0 + \sum_{k=1}^{\infty} \Big(\tilde{a}_k \cos(\alpha k x) + \tilde{b}_k \sin(\alpha k x) \Big) \tag{2.24}$$

where the Fourier coefficients $\tilde{a}_k$, $\tilde{b}_k \in \mathbb{R}$ can be determined by their respective formulas

$$\tilde{a}_0 = \frac{1}{L} \int_0^L u(x)\, dx, \quad \tilde{a}_k = \frac{2}{L} \int_0^L u(x) \cos(\alpha k x)\, dx, \quad \tilde{b}_k = \frac{2}{L} \int_0^L u(x) \sin(\alpha k x)\, dx, \quad k > 0.$$

and

$$\tilde{a}_0 = \tilde{u}_0, \quad \tilde{a}_k = 2\,\mathrm{Re}(\tilde{u}_k), \quad \tilde{b}_k = 2\,\mathrm{Im}(\tilde{u}_k), \quad k > 0.$$

The derivatives and integrals of the discrete Fourier series are expressed by elementary arithmetic operations. By differentiating equation (2.14) with respect to x, which is a spatial variable, we have,

$$\frac{du}{dx} = \sum_{k=-\infty}^{\infty} (\alpha i k)\, \tilde{u}_k\, e^{\alpha i k x}. \tag{2.25}$$

Similarly, integrating (2.14) yields

$$\int_0^L u(x)\, dx = L\, \tilde{u}_0\,. \tag{2.26}$$

Therefore, a derivative in the physical space corresponds to a product in the Fourier space.

Some results on convergence from Peyret [2002] are presented below:

- If the function $u(\cdot)$ is periodic, continuous on $[0, L]$ and bounded, then its Fourier series (2.14) converges uniformly.

- For a function $u(\cdot)$ periodic, continuous on $[0, 2\pi]$ and its derivatives $u^{(p)}(\cdot)$ up to order $m-1$ and with the m^{th} derivative absolutely integrable, we have

$$\tilde{u}_k = O(|k|^{-m}) \quad \text{as} \quad k \longrightarrow \infty.$$

- If the total variation of the m^{th} derivative is bounded, then

$$\tilde{u}_k = O(|k|^{-m-1}) \quad \text{as} \quad k \longrightarrow \infty.$$

- There is a fast convergence of $\tilde{u}_k$ towards zero if u is regular.

- The rate of convergence of the Fourier approximation is only algebraic in the presence of singularities.

- Let f be an absolutely integrable function. Then, the series (2.24) converges at all points in the interval $[a, b]$ if f has bounded total variation with the following properties:
  - $f(x) = \frac{1}{2}\left[f(x^+) + f(x^-)\right]$
  - $f(x + 2\pi) = f(x); \quad x \notin [0, 2\pi]$
  - $f(0) = f(2\pi) = \frac{1}{2}\left[f(0^+) + f(2\pi^-)\right]$.

2.1.5.2 Aliasing

As it can be noticed in the representations (2.22), some Fourier modes are not identified on the grid x_j,

$$e^{\alpha i(k+Nl)x_j} = e^{\alpha ikx_j}, \quad \forall k, l \in \mathbb{Z} \tag{2.27}$$

and this pollutes some of the Fourier coefficients, a phenomenon called aliasing.

$$\hat{u}_k = \sum_{l=-\infty}^{\infty} \tilde{u}_{k+Nl} = \tilde{u}_k + \sum_{\substack{l=-\infty \\ l\neq 0}}^{\infty} \tilde{u}_{k+Nl}. \tag{2.28}$$

The second sum is the discretisation error of (2.16), and it is appropriately called the aliasing error.

The use of a pseudo-spectral method to a non-linear equation requires some consideration as the aliasing error can become significantly bigger with the computations performing relatively slower. To illustrate this, consider the discrete Fourier transforms of the functions $u(x)$ and $v(x)$ at the grid points x_j,

$$u_j = \sum_{k=-N/2+1}^{N/2-1} \hat{u}_k e^{\alpha ikx_j}, \quad v_j = \sum_{k=-N/2+1}^{N/2-1} \hat{v}_k e^{\alpha ikj}, \quad j = 0, \ldots, N-1. \tag{2.29}$$

The discrete Fourier coeffcients of the product $s(x) = u(x)v(x)$ is a convolution sum,

$$\check{s}_k = \sum_{\substack{m+n=k \\ |m|,|n|\leq N/2-1}} \hat{u}_m \hat{v}_n, \quad |k| \leq N-2. \tag{2.30}$$

A direct computation of the above equality requires $O(N^2)$ operations. We have an expression of the form

$$\hat{s}_k = \check{s}_k + \sum_{\substack{m,n=-N/2+1 \\ m+n=k+N}}^{N/2-1} \hat{u}_m \hat{v}_n + \sum_{\substack{m,n=-N/2+1 \\ m+n=k-N}}^{N/2-1} \hat{u}_m \hat{v}_n \tag{2.31}$$

where the second and the third sum correspond to alias terms. A faster alternative would be to use the FFT where the coefficients,

$$\hat{s}_k = \frac{1}{N} \sum_{j=0}^{N-1} u_j v_j e^{-\alpha ikx_j}, \quad |k| \leq \frac{N}{2} - 1. \tag{2.32}$$

can be computed with a complexity of just $O(N \log_2 N)$ operations.

The pseudospectral technique for (2.31) involves the following steps:

1. Compute u_j, v_j, $j = 0, \cdots, N-1$ by FFT from the knowledge of $\hat{u}_k$ and $\hat{v}_k$, $k = -N/2+1, \cdots, N/2-1$.

2. Form the product $s_j = u_j\, v_j$.

3. Compute the Fourier coefficients $\check{s}_k$ by FFT from the values $\check{s}_j$.

Though less operations are involved in the above computations, the aliasing error still remains in (2.31) and the procedure to remove it will be discussed in the next section.

2.1.5.3 Aliasing removal

In order to remove the aliasing errors a simple technique can be implemented. It was considered by Orszag in 1971. This technique is called the "3/2 rule". It is simple and can be easily incorporated into the computations. Besides it is not an expensive operation in terms of computing time and the amount of memory. The principle is to extend the number of points on the grid, x_j, for the involved quantities such that the alias terms are not present in the pseudospectral procedure.

The algorithm is as follows:

- Extend the spectra $\hat{u}_k$, $\hat{v}_k$ in $\hat{u}_k^{'}$, $\hat{v}_k^{'}$ according to

$$\hat{u}_k^{'} = \begin{cases} \hat{u}_k & \text{if} \quad |k| \le K \\ 0 & \text{if} \quad K < |k| < K^{'} \end{cases} \qquad \hat{v}_k^{'} = \begin{cases} \hat{v}_k & \text{if} \quad |k| \le K \\ 0 & \text{if} \quad K < |k| < K^{'} \end{cases} \tag{2.33}$$

- Calculate by FFT, u_j and v_j in the new physical grid $x_j^{'} = \dfrac{L}{M} j$, $j = 1, \cdots, M = 2K^{'}+1$:

$$u(x_j^{'}) = \sum_{k=-K^{'}}^{K^{'}} \hat{u}_k^{'}\, e^{\alpha i k x_j^{'}}, \quad v(x_j^{'}) = \sum_{k=-K^{'}}^{K^{'}} \hat{v}_k^{'}\, e^{\alpha i k x_j^{'}} \tag{2.34}$$

- Multiply $u(x_j^{'})$ and $v(x_j^{'})$:

$$s(x_j^{'}) = u(x_j^{'})\, v(x_j^{'}), \quad j = 1, \cdots, M.$$

- Calculate by FFT the Fourier coefficients of $s(x_j^{'})$

$$\hat{s}_k^{'} = \frac{1}{M} \sum_{j=0}^{M-1} s(x_j^{'})\, e^{-\alpha i k x_j^{'}}, \quad |k| \le K^{'}. \tag{2.35}$$

This latter quantity is free of aliasing of the convolution sum (2.30). Further by considering the product $\hat{s}_k^{'}$, we have

$$\hat{s}_k^{'} = \sum_{\substack{p,q \\ p+q=k}} \hat{u}_p^{'}\hat{v}_q^{'} + \sum_{\substack{p,q \\ p+q=k+M}} \hat{u}_p^{'}\hat{v}_q^{'} + \sum_{\substack{p,q \\ p+q=k-M}} \hat{u}_p^{'}\hat{v}_q^{'}, \quad k = -K, \cdots, K \tag{2.36}$$

where $-K^{'} \leq p, q \leq K^{'}$, $\hat{u}_p^{'}$ and $\hat{v}_q^{'}$ are zero for $|p| > K$ and $|q| > K$. Observing that $q = 2K^{'} + 1 + k - p$ with $K \leq p, q \leq K$, the smaller value of q is $2K^{'} + 1 - 2K$. Therefore, the second and the third sum in (2.36) are zero if

$$2K^{'} + 1 - 2K > K, \tag{2.37}$$

which yields $2K^{'} > 3K - 1$. Thus, in order to eliminate the aliasing errors we may take $K^{'} \geq 3K/2$.

The effect of aliasing appears generally in the quality of the numerical results: oscillations and temporal instabilities. However by increasing the resolution it is found that aliasing effect can be negligible [Peyret, 2002].

We refer to Canuto *et al.*, [1988] for an extensive discussion on the phenomenon of aliasing.

2.1.5.4 Sine and cosine Fourier series

Non-periodic functions can also be represented as a Fourier series with the basis functions satisfying the boundary conditions. Some examples of these basis functions are sine, cosine, Chebyshev polynomials etc.

Consider the Fourier transforms of the sine and cosine function that satisfy the Dirichlet and Neumann boundary conditions respectively.

If a function $u : [0, 1] \to \mathbb{R}$ satisfies the homogeneous boundary conditions, $u(0) = u(1) = 0$, then $u(x)$ can be represented as a sine Fourier series

$$u(x) = \sum_{k=0}^{\infty} \tilde{u}_k \sin(\pi k x), \quad \tilde{u}_k = 2\int_0^1 u(x)\sin(\pi k x)\, dx. \tag{2.38}$$

Suppose u is defined on the $N + 1$ grid points,

$$x_j = \frac{j}{N}, \quad u_j = u(x_j), \quad j = 0, 1, \ldots, N. \tag{2.39}$$

Then we can deduce the discrete sine Fourier transform by applying the quadrature (2.20) for the integral in (2.38).

$$\hat{u}_k = \frac{2}{N}\sum_{j=1}^{N-1} u_j \sin\left(\frac{\pi k j}{N}\right), \quad k = 1, \ldots, N-1. \tag{2.40}$$

Thereafter the inverse transform is given by

$$u_j = \sum_{k=1}^{N-1} \hat{u}_k \sin\left(\frac{\pi k j}{N}\right), \quad j = 1, \ldots, N-1. \tag{2.41}$$

Similarly, with the Neumann boundary conditions:

$$\left.\frac{du}{dx}\right|_{x=0} = \left.\frac{du}{dx}\right|_{x=1} = 0 \tag{2.42}$$

the cosine Fourier series can be represented as

$$u(x) = \frac{\tilde{u}_0}{2} + \sum_{k=1}^{\infty} \tilde{u}_k \cos(\pi k x)\,, \quad \tilde{u}_k = 2\int_0^1 u(x)\cos(\pi k x)\,dx\,. \tag{2.43}$$

Its corresponding discrete cosine Fourier transform and inverse are

$$\hat{u}_k = \frac{2}{N}\left(\frac{u_0}{2} + \sum_{j=1}^{N-1} u_j \cos\left(\frac{\pi k j}{N}\right) + \frac{u^N}{2}(-1)^k\right), \quad k = 0, \ldots, N, \tag{2.44}$$

$$u_j = \frac{\hat{u}_0}{2} + \sum_{k=1}^{N-1} \hat{u}_k \cos\left(\frac{\pi k j}{N}\right) + \frac{\hat{u}^N}{2}(-1)^j, \quad j = 0, \ldots, N. \tag{2.45}$$

With these notions, let us construct the three-dimensional Fourier expansions for temperature, velocity and magnetic field in a horizontal plane layer.

We assume that the temperature $\theta(x,t)$ satisfies the Dirichlet boundary condition on the horizontal boundaries, that is, $\theta = 0$. So, the sine Fourier transform (2.41) can be used and θ can be expanded in x_3 as:

$$\theta = \sum_{k_3=0}^{N/2} \hat{\theta}_{k_3}(x_1, x_2, t)\sin(\pi k_3 x_3)\,. \tag{2.46}$$

This is a truncated expression at $k_3 = N/2$. Its Fourier coefficients $\hat{\theta}(x_1, x_2, t) \in \mathbb{R}$ can be computed using (2.40).

The temperature is also assumed to have a period of length L in the horizontal directions, therefore (2.46) can be further expressed as

$$\begin{aligned}\theta &= \sum_{k_3=0}^{N/2}\left(\sum_{k_2=-N/2+1}^{N/2-1} \hat{\theta}_{k_2,k_3}(x_1,t)\, e^{\alpha_2 i k_2 x_2}\right)\sin(\pi k_3 x_3) \\ &= \sum_{k_2=-N/2+1}^{N/2-1}\sum_{k_3=0}^{N/2} \hat{\theta}_{k_2,k_3}(x_1,t)\sin(\pi k_3 x_3)\, e^{\alpha_2 i k_2 x_2},\end{aligned}$$

where the Fourier coefficients $\hat{\theta}_{k_2,k_3}(x_1,t) \in \mathbb{C}$ can be computed from $\hat{\theta}_{k_3}(x_1,x_2,t)$. We indicate that since $\hat{\theta}(x_1,x_2,t)$ are the coefficients of a real-valued function, only half of $\hat{\theta}_{k_2,k_3}(x_1,t)$ are stored and the remaining coefficients are given by (2.23).

The final Fourier expansion of θ satisfying the Dirichlet boundary condition with periodicity in x_1 and x_2 is given by

$$\theta = \sum_{k_1=-N/2+1}^{N/2-1} \sum_{k_2=-N/2+1}^{N/2-1} \sum_{k_3=0}^{N/2} \hat{\theta}_{k_1,k_2,k_3}(t) \sin(\pi k_3 x_3)\, e^{i(\alpha_1 k_1 x_1 + \alpha_2 k_2 x_2)}\,. \tag{2.47}$$

In a similar manner, the velocity field, $\mathbf{v}(\mathbf{x},t) = (v_1(\mathbf{x},t), v_2(\mathbf{x},t), v_3(\mathbf{x},t))$ of the fluid flow in a plane layer satisfying stress-free boundary conditions on the horizontal planes and periodic in the the horizontal directions is represented as

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \sum_{k_1=-N/2+1}^{N/2-1} \sum_{k_2=-N/2+1}^{N/2-1} \sum_{k_3=0}^{N/2} \begin{pmatrix} \hat{v}^1_{k_1,k_2,k_3}(t) \cos(\pi k_3 x_3) \\ \hat{v}^2_{k_1,k_2,k_3}(t) \cos(\pi k_3 x_3) \\ \hat{v}^3_{k_1,k_2,k_3}(t) \sin(\pi k_3 x_3) \end{pmatrix} e^{i(\alpha_1 k_1 x_1 + \alpha_2 k_2 x_2)}. \tag{2.48}$$

The magnetic field $\mathbf{b}(\mathbf{x},t) = (b_1(\mathbf{x},t), b_2(\mathbf{x},t), b_3(\mathbf{x},t))$ in a plane layer with perfectly conducting horizontal boundaries and periodic in x_1 and x_2 directions can be written as

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \sum_{k_1=-N/2+1}^{N/2-1} \sum_{k_2=-N/2+1}^{N/2-1} \sum_{k_3=0}^{N/2} \begin{pmatrix} \hat{b}^1_{k_1,k_2,k_3}(t) \cos(\pi k_3 x_3) \\ \hat{b}^2_{k_1,k_2,k_3}(t) \cos(\pi k_3 x_3) \\ \hat{b}^3_{k_1,k_2,k_3}(t) \sin(\pi k_3 x_3) \end{pmatrix} e^{i(\alpha_1 k_1 x_1 + \alpha_2 k_2 x_2)}. \tag{2.49}$$

With these three dimensional Fourier representations (2.47)–(2.49), we search for solutions $(\mathbf{v}, \mathbf{b}, \theta)$ of the convective hydromagnetic problem. Subsequently, we represent the Fourier coefficients and the wave vectors using the below vector notation:

$$\hat{\mathbf{f}}_{\mathbf{k}} = (\hat{f}^1_{\mathbf{k}}, \hat{f}^2_{\mathbf{k}}, \hat{f}^3_{\mathbf{k}}), \quad \mathbf{k} = (k_1, k_2, k_3), \quad \tilde{\mathbf{k}} = (i\alpha_1 k_1, i\alpha_2 k_2, \pi k_3)\,. \tag{2.50}$$

The linear operations and their Fourier expressions with the corresponding basis functions for a field $\mathbf{f}$ expanded as in (2.48) are listed below.

- $\nabla \cdot \mathbf{f} = \sum_{\mathbf{k}} (i\alpha_1 k_1 \hat{f}^1_{\mathbf{k}} + i\alpha_2 k_2 \hat{f}^2_{\mathbf{k}} + \pi k_3 \hat{f}^3_{\mathbf{k}}) \cos(\pi k_3 x_3)\, e^{i(\alpha_1 k_1 x_1 + \alpha_2 k_2 x_2)}$
- $\nabla^2 \mathbf{f} = \sum_{\mathbf{k}} -(\alpha_1^2 k_1^2 + \alpha_2^2 k_2^2 + \pi^2 k_3^2) \begin{pmatrix} \hat{f}^1_{\mathbf{k}} \cos(\pi k_3 x_3) \\ \hat{f}^2_{\mathbf{k}} \cos(\pi k_3 x_3) \\ \hat{f}^3_{\mathbf{k}} \sin(\pi k_3 x_3) \end{pmatrix} e^{i(\alpha_1 k_1 x_1 + \alpha_2 k_2 x_2)}$
- $\nabla \times \mathbf{f} = \sum_{\mathbf{k}} \begin{pmatrix} (i\alpha_2 k_2 \hat{f}^3_{\mathbf{k}} + \pi k_3 \hat{f}^2_{\mathbf{k}}) \sin(\pi k_3 x_3) \\ (-\pi k_3 \hat{f}^1_{\mathbf{k}} - i\alpha_1 k_1 \hat{f}^3_{\mathbf{k}}) \sin(\pi k_3 x_3) \\ (i\alpha_1 k_1 \hat{f}^2_{\mathbf{k}} - i\alpha_2 k_2 \hat{f}^1_{\mathbf{k}}) \cos(\pi k_3 x_3) \end{pmatrix} e^{i(\alpha_1 k_1 x_1 + \alpha_2 k_2 x_2)}$
- $\mathbf{f} \times \mathbf{e}_3 = \sum_{\mathbf{k}} \begin{pmatrix} \hat{f}^2_{\mathbf{k}} \cos(\pi k_3 x_3) \\ -\hat{f}^1_{\mathbf{k}} \cos(\pi k_3 x_3) \\ 0 \end{pmatrix} e^{i(\alpha_1 k_1 x_1 + \alpha_2 k_2 x_2)}$

- $\dfrac{\partial \mathbf{f}}{\partial t} = \sum_{\mathbf{k}} \dfrac{d\hat{\mathbf{f}}_{\mathbf{k}}}{dt} \begin{pmatrix} \cos(\pi k_3 x_3) \\ \cos(\pi k_3 x_3) \\ \sin(\pi k_3 x_3) \end{pmatrix} e^{i(\alpha_1 k_1 x_1 + \alpha_2 k_2 x_2)}$

2.2 Solenoidality in the Fourier space

Here, we demonstrate how to project a vector field onto a space of solenoidal field. This is decisive in our study of the convective hydromagnetic problem because we need the solenoidality condition to hold true for all $t \geq 0$.

To begin with, the mean values of the flow, magnetic field and temperature, $\hat{\mathbf{v}}_{\mathbf{k}}$, $\hat{\mathbf{b}}_{\mathbf{k}}$ and $\hat{\theta}_{\mathbf{k}}$ for $\mathbf{k} = 0$ are conserved in time. We also consider that the fluid flow, $\mathbf{v}$, and the magnetic field, $\mathbf{b}$, are divergence-free. (i.e. $\nabla \cdot \mathbf{v} = \nabla \cdot \mathbf{b} = 0$). If this holds for an initial velocity, then

$$\tilde{\mathbf{k}} \cdot \hat{\mathbf{v}}_k(0) = 0\,, \quad \forall\, \mathbf{k}\,.$$

and since the fields are divergence-free:

$$\nabla \cdot \mathbf{v}(\mathbf{x}, t) = \nabla \cdot \mathbf{b}(\mathbf{x}, t) = 0, \quad \forall t \geq 0.$$

But because of numerical errors these conditions can be violated. So, the computed fields will have to be projected onto a space of solenoidal field in order to remain divergence-free.

Let us consider a vector field $\mathbf{a}(\mathbf{x}, t)$ satisfying the periodic boundary conditions in horizontal directions and

$$\left.\frac{\partial a_1}{\partial x_3}\right|_{x_3=0,1} = \left.\frac{\partial a_2}{\partial x_3}\right|_{x_3=0,1} = a_3|_{x_3=0,1} = 0\,.$$

Any smooth vector field $\mathbf{a}$ can be decomposed into a sum of a solenoidal (divergence-free field) and a gradient (curl-free field) part as below:

$$\mathbf{a} = \mathbf{a}_{\text{sol}} + \mathbf{a}_{\text{grad}}\,, \tag{2.51}$$

where $\nabla \cdot \mathbf{a}_{\text{sol}} = 0$, and $\mathbf{a}_{\text{grad}} = \nabla\phi$, with $\phi(\mathbf{x}, t)$ is a scalar function. The equation (2.51) is referred to as the Helmhotz decomposition. By applying divergence to (2.51), we obtain the Poisson equation

$$\nabla \cdot \mathbf{a} = \nabla^2 \phi, \quad \left.\frac{\partial \phi}{\partial x_3}\right|_{x_3=0,1} = a_3|_{x_3=0,1} = 0. \tag{2.52}$$

In the Fourier space, (2.52) corresponds to (see (2.50) for notation):

$$\tilde{\mathbf{k}} \cdot \hat{\mathbf{a}}_{\mathbf{k}} = -|\tilde{\mathbf{k}}|^2\, \hat{\phi}_{\mathbf{k}}, \quad \forall\, \mathbf{k}, \tag{2.53}$$

which can be rearranged to

$$\hat{\phi}_{\mathbf{k}} = -\frac{\tilde{\mathbf{k}} \cdot \hat{a}_{\mathbf{k}}}{|\tilde{\mathbf{k}}|^2}, \quad \forall\, \mathbf{k} \neq 0. \tag{2.54}$$

Computing the Fourier coefficients of $\nabla\phi$, we have

$$\widehat{\nabla\phi}_{\mathbf{k}} = \begin{pmatrix} i\alpha k_1 \\ i\alpha k_2 \\ -\pi k_3 \end{pmatrix} \hat{\phi}_{\mathbf{k}}. \tag{2.55}$$

So, from (2.51), we deduce that the vector field, $\mathbf{a(x},t)$, is divergence-free by performing a projection of the form $\hat{\mathbf{a}}_{\mathbf{k}} - \widehat{\nabla\phi}_{\mathbf{k}}$. We note that for $\mathbf{k} = \mathbf{0}$, the condition (2.53) is automatically satisfied and $\hat{\phi}_0 = 0$ for a mean-free vector field $\mathbf{a}$. We denote the projection of a vector field onto a space of solenoidal field by using an idempotent operator $\mathcal{P}$:

$$\mathcal{P}(\mathbf{a}) = \mathbf{a}_{\text{sol}}\,.$$

The above trick is neat and we point out that such a manoevre is required in order to suppress the accumulation of numerical errors due to finite precision and avoid violation of solenoidality conditions as we integrate our system forward in time. By applying $\mathcal{P}$, the pressure gradient, ∇p, is also eliminated in the Navier-Stokes equation (1.33).

2.3 Equations of hydromagnetic convection in the Fourier space

After substituting the Fourier coefficients (2.47), (2.48) and (2.49) into the Navier-Stokes equation and performing the projection onto the space of solenoidal fields with the use of $\mathcal{P}$, we equate them to the corresponding basis functions, yielding

$$\begin{aligned}\frac{d\hat{\mathbf{v}}_{\mathbf{k}}}{dt} = \mathcal{P}\left(-P\left(\alpha_1^2 k_1^2 + \alpha_2^2 k_2^2 + \pi^2 k_3^2\right)\begin{pmatrix}\hat{v}_{\mathbf{k}}^1 \\ \hat{v}_{\mathbf{k}}^2 \\ \hat{v}_{\mathbf{k}}^3\end{pmatrix} + P\sqrt{Ta}\begin{pmatrix}\hat{v}_{\mathbf{k}}^2 \\ -\hat{v}_{\mathbf{k}}^1 \\ 0\end{pmatrix} + PR\begin{pmatrix}0 \\ 0 \\ \hat{\theta}_{\mathbf{k}}\end{pmatrix}\right) \\ + \mathcal{P}\left(\widehat{\mathbf{v}\times(\nabla\times\mathbf{v})}_{\mathbf{k}} - \widehat{\mathbf{b}\times(\nabla\times\mathbf{b})}_{\mathbf{k}}\right).\end{aligned} \tag{2.56}$$

Similarly, the magnetic induction equation is given by

$$\frac{d\hat{\mathbf{b}}_{\mathbf{k}}}{dt} = -\frac{P}{P_m}\left(\alpha_1^2 k_1^2 + \alpha_2^2 k_2^2 + \pi^2 k_3^2\right)\begin{pmatrix}\hat{b}_{\mathbf{k}}^1 \\ \hat{b}_{\mathbf{k}}^2 \\ \hat{b}_{\mathbf{k}}^3\end{pmatrix} + \widehat{\nabla\times(\mathbf{v}\times\mathbf{b})}_{\mathbf{k}}\,. \tag{2.57}$$

The heat transfer equation can also be represented as

$$\frac{d\hat{\theta}_{\mathbf{k}}}{dt} = -\left(\alpha_1^2 k_1^2 + \alpha_2^2 k_2^2 + \pi^2 k_3^2\right)\begin{pmatrix}\hat{\theta}_{\mathbf{k}}^1 \\ \hat{\theta}_{\mathbf{k}}^2 \\ \hat{\theta}_{\mathbf{k}}^3\end{pmatrix} - \widehat{(\mathbf{v}\cdot\nabla)\theta}_{\mathbf{k}} + \hat{v}_{\mathbf{k}}^3\,. \tag{2.58}$$

The equations (2.56), (2.57) and (2.58) represent the equations of hydromagnetic convection in a rotating plane layer in the Fourier space. The non-linear terms are computed using

the pseudospectral technique using three-dimensional FFTs. The computation for each of the non-linear terms is as follows:

- $\mathbf{v} \times (\nabla \times \mathbf{v})$ requires six FFTs (three direct and three inverse):

$$\hat{\mathbf{v}}_\mathbf{k} \longrightarrow \nabla \times \hat{\mathbf{v}}_\mathbf{k} \xrightarrow{\text{FFT}^{-1}} \nabla \times \mathbf{v} \longrightarrow \mathbf{v} \times (\nabla \times \mathbf{v}) \xrightarrow{\text{FFT}} \widehat{\mathbf{v} \times (\nabla \times \mathbf{v})}_\mathbf{k}$$

- $\mathbf{b} \times (\nabla \times \mathbf{b})$ requires six FFTs (three direct and three inverse):

$$\hat{\mathbf{b}}_\mathbf{k} \longrightarrow \nabla \times \hat{\mathbf{b}}_\mathbf{k} \xrightarrow{\text{FFT}^{-1}} \nabla \times \mathbf{b} \longrightarrow \mathbf{b} \times (\nabla \times \mathbf{b}) \xrightarrow{\text{FFT}} \widehat{\mathbf{b} \times (\nabla \times \mathbf{b})}_\mathbf{k}$$

So, in total, twelve FFTs are needed for the non-linear terms in the Navier-Stokes equation (2.56). For the heat transfer equation (2.58), we require four FFTs (three inverse and one direct):

$$\hat{\theta}_\mathbf{k} \longrightarrow \nabla\hat{\theta}_\mathbf{k} \xrightarrow{\text{FFT}^{-1}} \nabla\theta \longrightarrow (\mathbf{v} \cdot \nabla)\theta \xrightarrow{\text{FFT}} \widehat{(\mathbf{v} \cdot \nabla)\theta}_\mathbf{k}$$

and, for the magnetic induction equation (2.57), we need three FFTs (direct):

$$\mathbf{v} \times \mathbf{b} \xrightarrow{\text{FFT}} \widehat{\mathbf{v} \times \mathbf{b}}_\mathbf{k}.$$

2.4 Time discretization method

Before we discuss the time stepping method used to solve the system of ODEs (2.56), (2.57) and (2.58), it is essential to explain what a stiff problem is and how to overcome the stiffness.

A stiff problem is a problem whose solution is characterized by time scales of different orders of magnitude [Peyret, 2002]. In other words, solutions evolve on two time scales. As an example, the differential equations of chemical kinetics describe reactions that evolve on different time scales – imagine the difference in time scales of corrosion and explosion [Iserles, 2009]. Stiffness is also observed in problems such as weather prediction, reactor kinetics, mathematical biology and electronics etc. They are known to have a phenomenon that exhibits variation at significantly different time scales.

In comparison, non-stiff systems can be resolved efficiently using schemes from Runge-Kutta (RK) and Adams-Bashfort [Conte, 2017] either with constant or adaptive time stepping.

In our computations, we have employed "exponential time differencing (ETD)", a numerical method that allows for significant reduction of the time step and avoids instability in the solution. The method involves the integration of the governing equations accompanied by an approximation of an integral that is comprised of non-linear terms. That is, the equations (2.56), (2.57) and (2.58) can be represented in the vector form

$$\dot{u} = c\,u + F(u,t)\,, \tag{2.59}$$

where c is a constant vector and $F(u,t)$ represents the nonlinear terms.

Let us derive the third-order scheme of the exponential time differencing method by considering the heat transfer equation:

$$\frac{d\hat{\theta}_{\mathbf{k}}}{dt} = c\,\hat{\theta}_{\mathbf{k}} - \widehat{(\mathbf{v}\cdot\nabla)\theta}_{\mathbf{k}} + \hat{v}^3_{\mathbf{k}}. \tag{2.60}$$

The constant, c, in (2.60) takes the value, $-|\tilde{\mathbf{k}}|^2$. For the Navier-Stokes and the magnetic induction equation, c equals $-\dfrac{P}{P_m}\,|\mathbf{k}|^2$ and $-P\,|\mathbf{k}|^2$, respectively. The rest of the procedure remains the same in their respective scheme derivation.

By multiplying (2.60) with the integrating factor, e^{-ct}, and then integrating it over a single time step, $t = t_n$ to $t = t_{n+1} = t_n + h$ yields

$$\begin{aligned}
\int_{t_n}^{t_{n+1}} \Big(\frac{d\hat{\theta}_{\mathbf{k}}}{dt}e^{-ct} - c\,\hat{\theta}_{\mathbf{k}}\,e^{-ct}\Big)\,dt &= \int_{t_n}^{t_{n+1}} e^{-ct}(\,\widehat{(\mathbf{v}\cdot\nabla)\theta}_{\mathbf{k}} + v^3_{\mathbf{k}})\,dt \\
\hat{\theta}_{\mathbf{k}}(t_{n+1})\,e^{-ct_{n+1}} - \hat{\theta}_{\mathbf{k}}(t_n)\,e^{-ct_n} &= \int_{t_n}^{t_{n+1}} e^{-ct}F(\hat{\theta}_{\mathbf{k}},t)\,dt \qquad (F(\hat{\theta}_{\mathbf{k}},t) = \widehat{(\mathbf{v}\cdot\nabla)\theta}_{\mathbf{k}} + v^3_{\mathbf{k}}) \\
\hat{\theta}_{\mathbf{k}}(t_{n+1})\,e^{-ch} - \hat{\theta}_{\mathbf{k}}(t_n) &= \int_{t_n}^{t_{n+1}} e^{-c(t-t_n)}F(\hat{\theta}_{\mathbf{k}},t)\,dt.
\end{aligned}$$

With a change of variables, where $\tau = t - t_n$, the above expression can be written as

$$\hat{\theta}_{\mathbf{k}}(t_{n+1}) = \hat{\theta}_{\mathbf{k}}(t_n)\,e^{ch} + e^{ch}\int_0^h e^{-c\tau}F(\hat{\theta}_{\mathbf{k}}(t_n+\tau),t_n+\tau)\,d\tau. \tag{2.61}$$

This is an exact formula. Our next task is to derive approximations to the integral in (2.61). An approach would be to replace the integral with a quadrature. Thus, (2.61) becomes

$$\hat{\theta}_{\mathbf{k}}(t_{n+1}) = \hat{\theta}_{\mathbf{k}}(t_n)\,e^{c\,h} + h\sum_{j=1}^{m} b_j\,F(\hat{\theta}_{\mathbf{k}}(t_n+k_jh),\,t_n+k_jh), \quad n=0,1,\ldots, \tag{2.62}$$

with an approximation of $\hat{\theta}_{\mathbf{k}}$ at the points $t_n + k_1h, t_n + k_2h, \ldots, t_n + k_mh$, where m defines the order of the approximation. The $k = (k_j)$ are the nodes and $b = (b_j)$ are the weights,

$$b_j(c\,h) = \int_0^1 e^{(1-\tau)c\,h}l_j(\tau)\,d\tau\,, \tag{2.63}$$

with $j = 1, 2, \ldots, m$. The l_j's are the familiar Lagrange interpolation polynomials

$$l_j(\tau) = \prod_{\substack{p=1\\p\neq j}}^{m}\frac{\tau-k_p}{k_j-k_p}, \quad j=1,\cdots,m. \tag{2.64}$$

The weights b_j are linear combinations of the entire functions [Hochbruck and Ostermann, 2010],

$$\varphi_j(c\,h) = \int_0^1 e^{(1-\tau)c\,h}\frac{\tau^{j-1}}{(j-1)!}\,d\tau, \quad j\geq 1. \tag{2.65}$$

Table 2.1 A variant of the classical Runge-Kutta method for weights and coefficients proposed by Cox and Matthews [2002].

$k_1 = 0$			
$k_2 = \frac{1}{2}$	$a_{2,1} = \frac{1}{2}\varphi_1$		
$k_3 = 1$	$a_{3,1} = -\varphi_1$	$a_{3,2} = 2\varphi_1$	
	$b_1 = 4\varphi_3 - 3\varphi_2 + \varphi_1$	$b_2 = -8\varphi_3 + 4\varphi_2$	$b_3 = 4\varphi_3 - \varphi_2$

(since we assume: Let X be a Banach space with norm $\|\cdot\|$. A is a linear operator on X and that $(-A)$ is the infinitesimal generator of a strongly continuous semigroup $\exp^{-tA}$ on X.)

The functions (2.65) satisfy $\varphi_j(0) = 1/j!$ and the recurrence relation

$$\varphi_{j+1}(c\,h) = \frac{\varphi_j(c\,h) - \varphi_j(0)}{ch}, \quad \varphi_0(c\,h) = e^{c\,h}. \tag{2.66}$$

Expanding (2.62) in Taylor's series about the same point $(\hat{\theta}_{\mathbf{k}}(t_n + k_j h),\, t_n + k_j h)$ and comparing the powers of h we determine the weights and coefficients. Based on the approximations of the integrand in (2.61), different explicit and implicit schemes of ETD methods can be obtained. Here, however, we construct the third order ETD RK scheme just as the classical third-order RK method [Iserles, 2009]. An advantage of the Runge-Kutta methods is that they have smaller error constants and larger stability regions. In contrast, multistep methods require previous evaluations of the nonlinear terms F [Cox and Matthews, 2002]. These methods can be inconvenient to use when initially only one value is available.

The third-order RK scheme comprises of three stages and we compute the value of $\hat{\theta}_{\mathbf{k}}$ at the points $t_n + k_j h$, with $k_j \in \left\{0, \frac{1}{2}, 1\right\}$, $j = 1, 2, 3$. We denote the approximation of $F(\hat{\theta}_{\mathbf{k}}(t_n))$ as F_n .

First, let $k_1 = 0$. Since the approximation is already provided by the former step of the numerical method, we have $\hat{\theta}_{\mathbf{k}}(t_n + k_1 h) = \hat{\theta}_{\mathbf{k}}(t_n)$.

For $k_2 = \frac{1}{2}$, we choose an approximation of F and the simplest of approximations is a constant, $F = F(\hat{\theta}_{\mathbf{k}}(t_n), t_n) + O(h)$. We evaluate from $t = t_n$ to $t = t_n + h/2$:

$$\int_0^{h/2} e^{-c\tau} F_n \, d\tau = F_n \left[\frac{e^{-c\tau}}{-c}\right]_0^{h/2} = \frac{F_n}{-c}\left(e^{-ch/2} - 1\right) = \frac{F_n}{c}\left(\frac{e^{ch/2} - 1}{e^{ch/2}}\right).$$

Substituting the above approximation in (2.61) we compute the value of $\hat{\theta}_{\mathbf{k}}$ at $t_n + h/2$, denoted by ς_1,

$$\varsigma_1 = \hat{\theta}_{\mathbf{k}}(t_n)\, e^{ch/2} + (e^{ch/2} - 1)\, F(\hat{\theta}_{\mathbf{k}}(t_n), t_n)/c\,. \tag{2.67}$$

Similarly, at $k_3 = 1$, we consider

$$F = 2F(\varsigma_1,\, t_n + h/2) - F(\hat{\theta}_{\mathbf{k}}(t_n), t_n) + O(h^2)\,, \tag{2.68}$$

on the interval $t_n \le t \le t_{n+1} = t_n + h$ and we substitute it in (2.61). We obtain:

$$\begin{aligned}\int_0^h e^{-c\tau} F(\hat{\theta}_{\mathbf{k}}(t_n+\tau), t_n+\tau)\, d\tau &= \int_0^h (2F(\varsigma_1,\, t_n + h/2) - F(\hat{\theta}_{\mathbf{k}}(t_n), t_n))\, e^{-c\tau}\, d\tau \\ &= (2F(\varsigma_1,\, t_n + h/2) - F(\hat{\theta}_{\mathbf{k}}(t_n), t_n)) \left[\frac{e^{-c\tau}}{-c}\right]_{\tau=0}^{h} \\ &= \frac{2F(\varsigma_1,\, t_n + h/2) - F(\hat{\theta}_{\mathbf{k}}(t_n), t_n)}{c} \frac{(e^{ch}-1)}{e^{ch}}.\end{aligned}$$

Thus, the approximate value of $\hat{\theta}_{\mathbf{k}}$ at $t_n + h$ is given by

$$\varsigma_2 = \hat{\theta}_{\mathbf{k}}(t_n)\, e^{ch} + (e^{ch}-1)\,(2F(\varsigma_1, t_n + h/2) - F(\hat{\theta}_{\mathbf{k}}(t_n), t_n))/c\ . \tag{2.69}$$

By inductively defining φ_j from (2.66) with $\varphi_0 = e^{ch}$,

$$\varphi_1(ch) = \frac{e^{ch}-1}{ch}, \quad \varphi_2(ch) = \frac{e^{ch}-1-ch}{c^2h^2}, \quad \varphi_3(ch) = \frac{e^{ch}-1-ch-\frac{1}{2}c^2h^2}{c^3h^3},$$

we compute the weights b_j :

$$b_1 = 4\varphi_3 - 3\varphi_2 + \varphi_1 \tag{2.70}$$

$$\begin{aligned}&= 4\left(\frac{e^{ch}-1-ch-\frac{1}{2}c^2h^2}{c^3h^3}\right) - 3\left(\frac{e^{ch}-1-ch}{c^2h^2}\right) + \frac{e^{ch}-1}{ch} \\ &= \frac{4e^{ch}-4-4ch-3che^{ch}+3ch+c^2h^2e^{ch}}{c^3h^3} \\ &= \frac{-4-ch+e^{ch}(4-3ch+c^2h^2)}{c^3h^3}\,,\end{aligned} \tag{2.71}$$

$$b_2 = -8\varphi_3 + 4\varphi_2 \tag{2.72}$$

$$\begin{aligned}&= -8\left(\frac{e^{ch}-1-ch-\frac{1}{2}c^2h^2}{c^3h^3}\right) + 4\left(\frac{e^{ch}-1-ch}{c^2h^2}\right) \\ &= \frac{-2e^{ch}+2+2ch+che^{ch}-ch}{c^3h^3} \\ &= \frac{4[2+ch+e^{ch}(-2+ch)]}{c^3h^3}\,,\end{aligned} \tag{2.73}$$

$$b_3 = 4\varphi_3 - \varphi_2 \tag{2.74}$$

$$\begin{aligned}&= 4\left(\frac{e^{ch}-1-ch-\frac{1}{2}c^2h^2}{c^3h^3}\right) - \frac{e^{ch}-1-ch}{c^2h^2} \\ &= \frac{4e^{ch}-4-4ch-2c^2h^2-che^{ch}+ch+c^2h^2}{c^3h^3} \\ &= \frac{-4-3ch-c^2h^2+e^{ch}(4-ch)}{c^3h^3}\,.\end{aligned} \tag{2.75}$$

By using (2.71), (2.73) and (2.75) in (2.62), we have the final quadrature formula through the points t_n, $t_n + h/2$ and $t_n + h$,

$$\begin{aligned}\hat{\theta}_{\mathbf{k}}(t_{n+1}) = \hat{\theta}_{\mathbf{k}}(t_n)\, e^{ch} &+ \{F(\hat{\theta}_{\mathbf{k}}(t_n), t_n)[-4 - hc + e^{ch}(4 - 3hc + h^2c^2)] \\ &+ 4F(\varsigma_1, t_n + h/2)[2 + hc + e^{ch}(-2 + hc)] \\ &+ F(\varsigma_2, t_n + h)[-4 - 3hc - h^2c^2 + e^{ch}(4 - hc)]\}/(h^2c^3)\,. \qquad (2.76)\end{aligned}$$

Equations (2.67), (2.69) and (2.76) constitute a third-order scheme, ETD3RK [Cox and Matthews, 2002].

We note that there are two practical points that the user must bear in mind when applying the ETD methods to PDEs. First, there are modes or harmonics with $c = 0$. In this case, the explicit formulae for the coefficients of $F\left(\hat{\theta}_{\mathbf{k}}(t_n),\, t_n\right), F\left(\hat{\theta}_{\mathbf{k}}(t_{n-1}), t_{n-1}\right)$ cannot be used directly, because they involve division by zero. Instead, their limiting form, as $c \to 0$, must be used: in our computations, we consider zero-mean for velocity, magnetic field and temperature profile. Their Fourier coefficients corresponding to the aforesaid mode are zero and are excluded in the computations. Second, it is necessary to take precaution while evaluating the constant vector c for modes where $|ch|$ is small. This is to avoid the rounding errors arising from the large amount of cancellation in the coefficients. In some cases, when $|ch| \ll 1$, the Taylor series can be used for c rather than the explicit formulae. We use quadruple precision while computing such quantities in the program code and minimize the rounding errors.

Integration in time of (2.56), (2.57) and (2.58) were executed with time steps $h = 0.0005$, 0.00075, and 0.001. Smaller steps were used when the initial magnetic energy exceeded the kinetic energy and for the largest period factors M_1 and M_2. Transient regimes in computations required smaller time steps as to larger time steps for the saturated regimes.

2.5 Symmetries

Analysis of symmetries is an important tool in the investigation of invariant structures in the phase space of a dynamical system. Spatial and temporal translations, the Galilean transformations, parity invariance, rotations and scalings are the symmetries of the Navier-Stokes equation [Frisch 1995]. In this section, we list only the spatial symmetries of the rotating convective dynamo identified in the simulations and are mentioned in [Podvigina, 2006].

Some definitions for symmetries:

- A spatial transformation Γ of the equation

$$\frac{\partial \mathbf{y}}{\partial t} = \mathcal{F}\mathbf{y}, \quad \text{with} \quad \mathbf{y} = \mathbf{y}(\mathbf{x}, t) \in \mathbb{R}^3, \quad \mathcal{F} : \mathbb{R}^3 \to \mathbb{R}^3\,, \qquad (2.77)$$

  is a symmetry group if for a solution $\mathbf{y}$ and a symmetry $\gamma \in \Gamma$, the transformation $\gamma\mathbf{y}$ is also a solution of the equation. An equivalent definition is the following: $\mathcal{F}\gamma\mathbf{y} = \gamma\mathcal{F}\mathbf{y}$ for $\forall\gamma \in \Gamma$, $\forall\mathbf{y} \in \mathbb{R}^3$.

- An isotropy subgroup of a solution $\mathbf{y}$ to (2.77) is a subgroup $\Gamma_y \subseteq \Gamma$ given by

$$\Gamma_y = \{\gamma \in \Gamma | \gamma y = y\}\,. \tag{2.78}$$

A symmetry group of the Boussinesq convection in a plane layer together with the Dirichlet and Neumann boundary conditions is $\mathbf{Z}_4 \ltimes (\mathbf{T}_1 \times \mathbf{T}_2) \times \mathbf{Z}_2 \times \mathbf{Z}_2$ for the (M_1, M_2)-periodicity box, $\mathscr{C}$, with its horizontal face, $M_1 = M_2$, or $\mathbf{Z}_2 \ltimes (\mathbf{T}_1 \times \mathbf{T}_2) \times \mathbf{Z}_2 \times \mathbf{Z}_2$ otherwise. Here, the subgroup $\mathbf{Z}_4$ comprises of rotations by $\pi/2$ about the vertical axis:

$$\begin{aligned} s_1 &: (x_1, x_2, x_3) \mapsto (x_2, -x_1, x_3)\,, \\ s_2 &: (x_1, x_2, x_3) \mapsto (-x_1, -x_2, x_3)\,, \\ s_3 &: (x_1, x_2, x_3) \mapsto (-x_2, x_1, x_3)\,, \end{aligned}$$

and the identity $s_0 = \mathcal{I}$. If the horizontal face of $\mathscr{C}$ is not square, the rotations by π are the only possible scenarios, i.e. the rotations do not involve s_1 and s_2.

The subgroups $\mathbf{T}_1$ and $\mathbf{T}_2$ consist of translations in the x_1 and x_2 directions, respectively:

$$\begin{aligned} \gamma_{\alpha_1}^{x_1} &: (x_1, x_2, x_3) \mapsto (x_1 + \alpha_1, x_2, x_3)\,, \\ \gamma_{\alpha_2}^{x_2} &: (x_1, x_2, x_3) \mapsto (x_1, x_2 + \alpha_2, x_3)\,, \end{aligned}$$

where $0 \leq \alpha_i < M_i\, L\, (\gamma_{M_1 L}^{x_1} = \gamma_{M_2 L}^{x_2} = \mathcal{I})$. The final subgroup, $\mathbf{Z}_2$, is generated by reflection about the horizontal midplane:

$$r : (x_1, x_2, x_3) \mapsto (x_1, x_2, 1 - x_3),$$

and, in the presence of magnetic field, the reversing symmetry, q, given by

$$q : (\mathbf{v}, \theta, \mathbf{b}) \mapsto (\mathbf{v}, \theta, \text{-}\mathbf{b})\,.$$

We note that the rotations s_i and the reflection r about the horizontal midplane affect the spatial coordinates and the components of the vector fields. If periodicity is not imposed then the symmetry group of the Boussinesq convection with the boundary conditions is $\mathbf{SO}(2) \times \mathbf{R}^2 \times \mathbf{Z}^2$ which includes rotations, plane translations and reflections about the midplane.

The symmetries reported in Table 3.2 were detected numerically. A field $\mathbf{f}$ is considered to possess a symmetry σ if the norm of discrepancy $\mathbf{f} - \sigma\,\mathbf{f}$, in the Lebesgue space $L_2(\mathscr{C})$, is smaller than the threshold value of 10^{-6}.

2.6 Software libraries

In this section we deliberate the computing tools used in this project. FFTW and P3DFFT are the frontrunners and this discussion is focussed on their features, functionalities, algorithms and data layout both in terms of local memory and the processor layout.

2.6.1 FFTW

FFTW (Fast Fourier Transform in the West), is a software library for computing discrete Fourier transforms. It was developed by Matteo Frigo and Steven G. Johnson [1998]. According to them, the number of floating-point operations performed by an FFT algorithm in modern-day processors is far less important and that the interactions with the processor pipeline and the memory hierarchy has a larger impact on performance. So, they built an adaptive program called FFTW, which tunes the computations automatically for a particular hardware.

2.6.1.1 Features

Some important features of FFTW are:

- It is portable to any platform with a C compiler.
- Its algorithms use $O(N \log N)$ operations for computing the transforms. The input data size N can be arbitrary and it also includes prime numbers.
- FFTW's library supports Fortran and C interfaces.
- Transforms for Discrete Fourier Sine and Cosine are available.
- Parallel multi-threaded and distributed memory transforms with MPI are provided - starting from version 3.3.

2.6.1.2 Composition and runtime structure

FFTW is composed of the following elements. In fact they are its core engine.

Executor. An executor consists of highly optimized composable blocks of C code called codelets. A codelet is a specialized piece of code that computes part of the transform. The computation of the transform is accomplished by an executor.

Plan. A plan is a special data structure that specifies the combination of codelets to be applied by the executor. It can be reused as many times as needed.

Planner. A planner determines the plan using a dynamic programming algorithm at run time (before the computations begin). It tries to minimize the execution time of the computations by measuring the run time of different plans and selects the fastest for the current hardware.

A code generator. It is a program that generates codelets automatically with the help of a special-purpose compiler. It is written in the Caml Light dialect of the functional language ML (CAML). It produces a representation of a codelet in the form of an abstract syntax tree (AST), and then optimizes it. A code generator is useful because the speed of an executor depends on the efficiency of the codelets but writing and optimizing them can be tedious and error-prone. It is also simple to experiment with different algorithms or coding strategies.

Some algorithms that the code generator uses are the Cooley-Tukey algorithm, the prime-factor algorithm, Rader's algorithm (for prime sizes), and a split-radix algorithm. It can also produce new algorithms that are yet to be understood [Frigo and Johnson, 1998].

The executor implements the Cooley-Tukey algorithm as follows. If a data array of size N is factored into $N = N_1 N_2$ then subsequently, the algorithm recursively computes N_1 transforms of size N_2. It then multiplies the results by certain constants traditionally called twiddle factors, and finally computes N_2 transforms of size N_1.

Codelets are of two types:

- Non-twiddle or normal codelets. They compute transform of small sizes. They are the base case in recursion.
- Twiddle codelets. These codelets combine small transforms and compute a bigger transform. They are similar to normal codelets but in addition their input is multipied by twiddle factors. These are used for the internal levels of the recursion.

The following is an example of a high-level description of a possible plan for a transform of length $N = 128$.

```
FFT(128) =
DIVIDE-AND-CONQUER(128, 4)
DIVIDE-AND-CONQUER(32, 8)
SOLVE(4)
```

Initially the executor computes 4 transforms of size 32 recursively. It then uses the twiddle codelet of size 4 to combine the results of the sub problems. In a similar manner, the problems of size 32 are divided into 8, each of size 4, which are then solved using a normal codelet. The results are combined using a size-8 twiddle codelet.

To summarize, the computation of transform involves two phases. First, a planner determines a plan. Second the executor executes the plan on the input data array. The executor takes in an array to be transformed, an optimal plan, which is a data structure on how to factorize N, and a right composition of codelets to be used.

Array sizes which can be factored into small primes (2,3, 5 and 7) are efficient. Otherwise a slower general-purpose routine is used. For efficient transforms of other sizes, the code generator can be used. For example, transforms of size $513 = 19 \times 3^3$ can be supported in an efficient way with a customized FFTW.

The planner can also collect data about the machines, store them and reuse later. This is useful if numerical simulations are to be executed for a longer duration.

2.6.1.3 Plans

FFTW's strategy to find an optimal plan is simple. The planner should measure the execution time of many plans and select the best for the underlying hardware. Practically to try all plans is not possible due to the finite number of combinations involved. Instead the planner confines its search space using a dynamic programming algorithm.

This selected optimal plan for a size N is still assumed to be optimal for a sub-structure as well. That is, when size N is used as a subproblem of a larger transform. This assumption

is false because the states of the cache are different in both the cases [Frigo and Johnson, 1998].

Below are the available FFTW planner flags that can be chosen by the user.

- FFTW_ESTIMATE. With this flag a sub-optimal plan with no run-time tuning is picked. The initialization times are short and the algorithms for transforms would be slow. It is suitable for shorter runs.
- FFTW_MEASURE. This flag allows for experiments with different algorithms and chooses the fastest algorithm. It is the default planning option.
- FFTW_PATIENT. It is similar to FFTW_MEASURE but more algorithms are considered. FFTW_PATIENT was used in all our computations.
- FFTW_EXHAUSTIVE. Here even more algorithms are taken into account while determining the fastest.
- FFTW_WISDOM_ONLY. It creates a plan only if wisdom (refer to section 2.6.1.4 for more details) is available for the given problem, otherwise a NULL plan is returned. It can be combined with other flags, e.g. 'FFTW_PATIENT + FFTW_WISDOM_ONLY' if wisdom was created with FFTW_PATIENT or FFTW_EXHAUSTIVE. In simple words, using this flag users can detect if wisdom is disposable. We use this flag in our computations in order to continue simulations from the previously stored states.

Plans created with FFTW_MEASURE, FFTW_PATIENT or FFTW_EXHAUSTIVE options produce near-optimal performance.

2.6.1.4 Wisdom

FFTW implements a method for storing plans to disk and restoring them and this mechanism is called *wisdom*. It is a local database of plans for a program, for a hardware configuration. It is designed for situations where many transforms of the same size are needed.

The wisdom mechanism can be used to save and reload many plans at once and as many times as required.When a plan is created, the planner accumulates sufficient information needed to reconstruct the plan. It is stored in a global, private data structure internally managed by FFTW. This information can be exported to a file, string or any other medium using the interfaces provided by the library.

However there are pitfalls to using wisdom. We highlight a few below.

An optimal plan created on one processor for a program can be exported and imported to the same program running on another processor, but this plan may be sub optimal for the second program as it was optimized for the former. Moreover, FFTW's ability to adapt to the changing hardware is negated by doing so.

Even between runs of the same binary on an identical hardware, an optimal plan may change because of the differences in the virtual memory environment [Frigo and Johnson, 1998].

Using the same wisdom for two different binaries, a plan may be sub-optimal due to the differing code alignments. The risk of sub-optimal plans grows with the changes to hardware and software between the creation of wisdom and its use. As long as these remain unchanged, the plans are expected to be optimal.

Plan creation with FFTW_MEASURE, FFTW_PATIENT or FFTW_EXHAUSTIVE requires more time. This is because FFTW measures the execution time of many plans and selects the best one. Yet this saves computational time while performing the transforms.

Some pros of implementing the wisdom mechanism are:

- Different program binaries can use the same plan though it may be sub-optimal.
- Starts and stops of the runs can be handled efficiently. There is no need for plan recreation as the information saved in the wisdom files from the initial run can be imported and reused.
- It can be made sure that the results of different runs are bit-wise identical. This is helpful in debugging the code for errors.
- Wisdom is used for any size of the data array if the planner flags are not more "patient" than those with which wisdom was created [Frigo and Johnson, 1998]. For example, wisdom created with FFTW_MEASURE can be used for FFTW_ESTIMATE if there is a modification later.

2.6.1.5 Distributed memory with MPI

FFTW also has parallel routines for multiprocessor/multicore systems with shared-memory. Both parallel one-dimensional and multi-dimensional transforms are supported. They function just like the uniprocessor transforms except for an extra initialization routine call and a routine to set the number of threads for use. Although additional processors are used in the computation this does not necessarily result in performance gains. There is an overhead for synchronization and that may outweigh the computational problem. Multi-threaded FFTWs are only beneficial if the problem is sufficiently large.

FFTW can also be used with MPI – a message passing library for multiprocessor systems and the acronym stands for Message Processing Interface - with distributed memory. MPI allows each processor to have its own separate memory, a parallelism referred to as *distributed-memory parallelism*. In this arrangement each process holds a portion of data for computation and so the data structures and the programming interfaces are different from the serial or multi-threaded FFTW routines.

Distributed-memory parallelism is useful when transforming large arrays that do not fit into the memory of a single processor. The total array size divided by the number of processes gives the amount of storage per-process required by FFTW's MPI routines. As with the multi-threaded routines, communication between processes can add significant computational time if the array size is small. Distributed parallelism is favorable based on the array size, the available hardware and the MPI implementation.

FFTW uses 1D block distribution of data, distributed along the first dimension. Each block of data can be referred to as a slab. For example, a 100×200 complex DFT can be distributed over 4 processes, each with a slice of the data, 25×200. That is, process 0 with rows 0 through 24, process 1 with rows 25 through 49, process 2 with rows 50 through 74 and process 3 from row 75 to row 100. By default, FFTW divides the data equally among the processes if possible and the rows are assigned in rank order (i.e) process 0 gets the first block of rows, then it is process 1 and so on.

To refer back to the Fourier representations in (2.48), (2.49), and (2.47), the third component has sine transforms in the x_3 direction and we use FFTW's one dimensional real-odd discrete Fourier Transform (RODFT00) to compute it, where the odd symmetry means $\hat{u}_k = -\hat{u}_{N-k}$. Because of this redundancy, only the first n numbers starting at $k = 1$ are stored (the value of the Fourier coefficient at the index $k = 0$ is zero). Here, $N = 2(n+1)$. In conjunction with P3DFFT, which is discussed in the section 2.6.2, a provision for custom transforms in the x_3 direction is made available in the parallel code (see Appendix A.3).

2.6.2 P3DFFT

P3DFFT [Pekurovsky, 2012] is a software library for large-scale computer simulations on parallel platforms. P3DFFT stands for Parallel Three-Dimensional Fast Fourier Transform. It relies on lower-level libraries such as FFTW and ESSL (Engineering Scientific Subroutine Library), to perform the 1D FFTs.

2.6.2.1 Features

One of the salient features of this library is its 2D processor decomposition. The processors are arranged into a 2D grid according to their MPI rank. A MPI rank is an integer assigned to each processor involved in performing a task – if there are N processors executing a program, they will have ranks $0, \ldots, N-1$. The processor with the rank 0 is the master processor which coordinates the task with other processors. It is the only processor with the privilege to implement read and write operations to the disk.

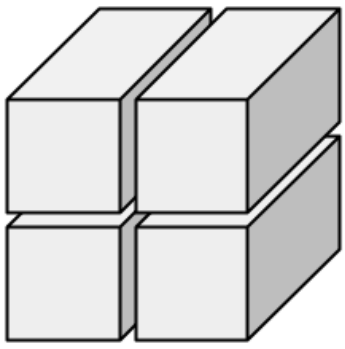


Figure 2.1 An illustration of a 2D block (pencil decomposition) of a 3D grid.

To instruct about the 3D data array, two of its dimension is decomposed into blocks. The number of blocks is in proportion to the number of processors requested at the start of the

simulation. Each block of the data array is assigned to a task in the rank order in the 2D processor grid. The third dimension of the data array is undivided. This decomposition is called pencil decomposition [Pekurovsky, 2012] (also referred to as drawer or block decomposition).

In contrast, a slab decomposition implements a 1D decomposition on a 3D data array. If data is distributed along a plane, say X, then any computation along the other two planes, $Y - Z$, is in local memory. This allows to use a maximum of N cores on an array of size N^3. This is a severe limitation because modern-day super computers provide several hundreds of thousands of cores for use. To overcome this issue, Pekurovsky [2012] developed P3DFFT. This allows for a maximum of N^2 cores equal to the maximum number of pencils.

Further, decomposition of the input and output arrays, although it is two dimensional, differs one from another. It is because in a 3D Fourier transform, data is transposed twice (for two-dimensional decomposition) for rearrangement and for performing the 1D FFT on the data local in memory of each processor. By relinquishing the final transpose at this stage and executing it after completing the 3D transform, the computational time is significantly reduced.

The other main features of P3DFFT are:

- Real-to-complex and complex-to-real Fast Fourier Transforms (FFT) in 3D are implemented.
- Transforms for cosine, sine and combined Fourier-Chebyshev (FFT in 2D and Chebyshev in the third dimension) are available. Users can also substitute their own transform in the third dimension.
- Fortran and C interfaces are supported;
- Functionalities for in-place, out-of place and pruned transforms;
- The library is built for performance for all range of core counts;
- It uses MPI as the primary method for interprocessor communication.

2.6.2.2 Data mapping

P3DFFT splits down the data array into 2D blocks. In many cases, this decomposition is uneven: some tasks receive more array elements than others. However, P3DFFT attempts to minimize this imbalance.

In the physical space, we have an array of real numbers local in memory in the X dimension, distributed among P_1 processors in the Y dimension and among P_2 processors in the Z dimension. P_1 and P_2 are processor grid dimensions. This array type is an input of real-complex forward transform and an output of complex-real backward transform.

As an example, for a grid dimension $128 \times 256 \times 97$ and a processor grid 3×4, the 256 elements in the Y dimension are decomposed into three rows, each containing 86, 85 and 85 elements respectively.

In comparison, in the Fourier space, we have an array of complex numbers local in memory in the Z dimension, distributed among P_2 processors in the Y dimension and among P_1

Table 2.2 Mapping of data array onto processor grid and memory layout.

Configuration	Physical space	Fourier space
STRIDE1 defined	$N_x, N_y/P_1, N_z/P_2$	$N_z, N_y/P_2, (N_x+2)/(2P_1)$
STRIDE1 undefined	$N_x, N_y/P_1, N_z/P_2$	$(N_x+2)/(2P_1), N_y/P_2, N_z$

processors in the X dimension. This array type serves as an output of real-to-complex forward transform and an input to complex-to-real backward transform.

Using the previous example, the transposed local arrays here would have 22, 22 and 21 elements in the X dimension for processors 1 through 3. The sum of all these numbers is $65 = (N_x + 2)/2$. The additional two modes are for the Nyquist frequency.

Thus two array types are distributed among processors in a different way from each other.

The memory layout in the physical space implements column-major ordering. That is, consecutive elements of a column reside next to each other in memory. In the Fourier space, with the help of the stride feature, a similar memory layout can be enforced. We remind that the transposed array in this case is of the form (Z, Y, X). With the elements local in Z, cache utilization is improved. Consequently, better performance can be seen in the transforms. By default, the original ordering of the real array is preserved for the transposed array.

Another forethought should be on the choice of the processor grid $P_1 \times P_2$, because they can make a difference in how balanced a decomposition is. When the load imbalance is greater, less performance can be expected.

The stride (STRIDE1 defined) and non-stride (STRIDE1 undefined) mapping of the data array onto the processor grid and memory layout are presented in Table 2.2.

2.7 Summary

The key takeways in this chapter are:

- an overview of the methods of discretization – spectral, Galerkin, Fourier.
- the notion of aliasing and how to remove it using the "3/2" rule.
- the three-dimensional Fourier representations (2.47)–(2.49) for the variables $(\mathbf{v}, \mathbf{b}, \theta)$.
- the definition of the differential operators (derivative, divergence, Laplacian and curl) in the Fourier space.
- the procedure on how to project a vector field $\mathbf{f}$ onto the space of solenoidal fields.
- the representation of the equations of hydromagnetic convection in the Fourier space and their integration by the ETD3RK method.
- the symmetries of the invariant structures examined in the phase space of a dynamical system using the notation from [Podvigina, 2006].

- the computation of the Fourier transforms in FFTW and P3DFFT; distribution of the data array in the processor and memory.

In the next chapter, we present and discuss the numerical results obtained from our dynamo simulations.

Chapter 3

Numerical Results

In this chapter we discuss the computations involved in solving the convective dynamo equations and the results obtained in our simulations: evolution of perturbations, the regimes with their respective periodicities, the attractors and metastable states observed in their time evolution. We also provide additional materials in the form of videos to support some of our claims.

Before we present these results, we remind the reader of the following. Our nonlinear convective dynamo is governed by:

- the Navier-Stokes equation, comprising of the Archimedes (buoyancy), Coriolis and Lorentz forces

$$\frac{\partial \mathbf{v}}{\partial t} = P\nabla^2\mathbf{v} + \mathbf{v}\times(\nabla\times\mathbf{v}) + PR\theta\mathbf{e}_3 + P\sqrt{Ta}\mathbf{v}\times\mathbf{e}_3 - \mathbf{b}\times(\nabla\times\mathbf{b}) - \nabla p\,, \tag{3.1}$$

- the magnetic induction equation

$$\frac{\partial \mathbf{b}}{\partial t} = \nabla\times(\mathbf{v}\times\mathbf{b}) + \frac{P}{P_m}\nabla^2\mathbf{b}\ , \tag{3.2}$$

- the heat transfer equation

$$\frac{\partial \theta}{\partial t} = \nabla^2\theta - (\mathbf{v}\cdot\nabla)\theta + v_3\ . \tag{3.3}$$

Here, $\mathbf{v}$ denotes the flow velocity in the fluid layer, $\mathbf{b}$ is the magnetic field, and

$$\theta = T - (T_0 + (T_1 - T_0)x_3)\ ,$$

is the difference between the temperature T of the fluid and the linear temperature profile for the fluid at equilibrium. T_0 and T_1 are constant temperatures on the boundaries, $x_3 = 0$ and $x_3 = 1$, respectively.

The fluid is incompressible and the magnetic field is solenoidal:

$$\nabla\cdot\mathbf{v} = 0\ , \tag{3.4a}$$

$$\nabla \cdot \mathbf{b} = 0 \ . \tag{3.4b}$$

The standard nondimensional parameters describing thermal convection in the presence of magnetic field are used:

$$P = 1,\ P_m = 8,\ R = 2300,\ L = 2\sqrt{2} \ .$$

We consider stress-free perfectly electrically conducting boundaries. Therefore the following conditions are satisfied on the horizontal boundaries:

$$\left.\frac{\partial v_1}{\partial x_3}\right|_{x_3=0,1} = \left.\frac{\partial v_2}{\partial x_3}\right|_{x_3=0,1} = \left. v_3 \right|_{x_3=0,1} = 0, \tag{3.5}$$

$$\left.\frac{\partial b_1}{\partial x_3}\right|_{x_3=0,1} = \left.\frac{\partial b_2}{\partial x_3}\right|_{x_3=0,1} = \left. b_3 \right|_{x_3=0,1} = 0, \tag{3.6}$$

$$\theta\,|_{x_3=0,1} = 0. \tag{3.7}$$

The fields $(\mathbf{v}, \mathbf{b}, \theta)$ have periods L in the horizontal directions x_1 and x_2,

$$\mathbf{v}(\mathbf{x}) = \mathbf{v}(x_1 + m_1 M_1 L, x_2 + m_2 M_2 L, x_3), \tag{3.8}$$

$$\mathbf{b}(\mathbf{x}) = \mathbf{b}(x_1 + m_1 M_1 L, x_2 + m_2 M_2 L, x_3), \tag{3.9}$$

$$\theta(\mathbf{x}) = \theta(x_1 + m_1 M_1 L, x_2 + m_2 M_2 L, x_3), \quad \forall m_1, m_2 \in \mathbb{Z}. \tag{3.10}$$

and are expanded as truncated Fourier series (2.47), (2.48), (2.49) satisfying their respective boundary conditions, (1.23) and (1.36).

The standard pseudospectral algorithms [Boyd, 2001, Canuto *et al.*, 1988, Peyret, 2002] are used to solve the dynamo equations numerically and computations are performed for all pairs of integers (M_1, M_2) in the range $1 \leq M_1, M_2 \leq 4$ with the scale ratios confined to the interval $0.25 \leq \varepsilon \leq 1$.

3.1 Numerical techniques

Simulations involving three large-scale perturbations were executed for each of the (M_1, M_2)-periodicity boxes and are listed below. The runs are labelled accordingly by indices 1 to 3. For instance, for the (3,2)-periodicity box in run 1, we referred to it as 3x2.1.

- Run 1. Here only the coefficients of the harmonics associated with the wave vectors $(2\pi/(M_1 L), 0, 0), (0, 2\pi/(M_2 L))$ and $(0, 0, \pi)$ along with their complex-conjugates of the flow and the magnetic field were randomly perturbed. The energy of the initial perturbation is in the range $10^{-5} - 0.2$.

- Run 2. We generate random Fourier coefficients with the kinetic energy several times larger than the magnetic energy.

- Run 3. Random Fourier coefficients are employed with the kinetic energy several times smaller than the magnetic energy.

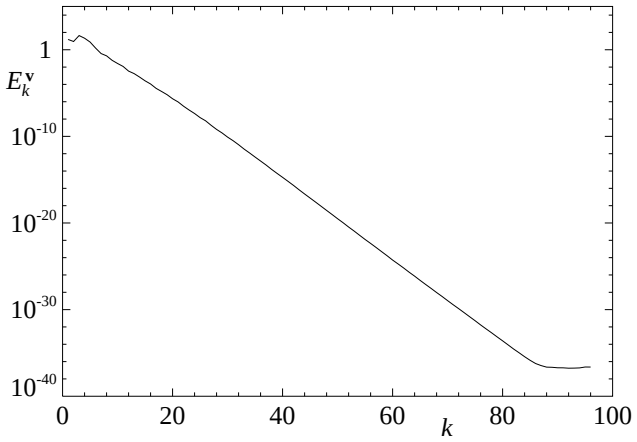


Figure 3.1 Energy spectrum for the flow at $t = 220$ in run 2 for $M_1 = M_2 = 2$; the kinetic energy distribution E_k^v over shells C_k is shown for fully populated shells only.

The Fourier coefficients in runs 2 and 3 possess an exponentially decay spectrum i.e., their amplitude exponentially decays on increasing the length of the wave vector and, the gradient part is removed in order to satisfy (3.4). The initial conditions here can be regarded as large initial perturbations in which the kinetic and magnetic energies exceed the energies of the respective fields or have similar magnitudes. Runs 2 and 3 were performed to detect if additional attractors coexist for the same dynamical system under consideration.

We define the energy of a vector field $\mathbf{f}$ as

$$E^{\mathbf{f}} = \left\langle \frac{1}{2}|\mathbf{f}|^2 \right\rangle,$$

which is a reference to the spatially averaged energy density.

For all the (M_1, M_2)-periodicity box runs were performed with the resolution of $N_1 = 128M_1$, $N_2 = 128M_2$ and $N_3 = 96$ harmonics. We call this as the "standard" resolution in our computations. In order to ensure that this resolution is sufficient, the energy spectrum of the unknown fields were computed, that is, the distribution of energy E_k contained in harmonics associated with the wave vectors in successive spherical shells

$$C_k = \{\mathbf{n} \mid k - 1 < |\mathbf{n}| \leq k\}, \quad k \in \mathbb{N},$$

of width one. More precisely, the minimum energy spectrum decay estimates are computed by

$$\log_{10} \min_t \left(\max_{1 \leq k \leq k_{\min}} E_k(t) \,/\, \min_{1 \leq k \leq k_{\min}} E_k(t) \right). \tag{3.11}$$

Here, $k_{\min} = \min(N_1/2, N_2/2, N_3)$, and the minimum and maximum in the above ratio are computed over the fully populated shells. In all the runs, the minimum energy spectrum decay for the flow, magnetic field and temperature over any length of time is 7, 4 and 12 orders of magnitude, respectively. Sometimes, due to round-off errors plateaux were seen at

the far end of the shells in the energy spectra plots, E_k vs k (refer to Fig. 3.1). So these outer shells constituting the plateaux were excluded while computing the quantity (3.11).

The energy spectra decays observed in the computations for different periodicity boxes are shown in Table 3.1. These numbers prove that the resolution employed is sufficient, and hence we regard them as our principal runs. However, runs which have its energy spectra decay below 7, were simulated again with the constant "alternative" resolution of $256 \times 256 \times 257$ harmonics. The main reason for the low decays in our computations is the insufficient number of harmonics in the vertical direction. The wave numbers in the vertical direction is in the range $[0, 256]$. Conclusions drawn from the new runs were checked against the results in the standard runs: all attractors except for the attractors in the triple-period (1x3/3x1) fluid cells are chaotic, and therefore the comparisons are effected only qualitatively. The minimum decays from the alternative resolution are at least by 10, 6, and 15 decades, respectively. The estimates from the alternative resolution are also shown in Table 3.1.

Time stepping was performed by the third-order scheme ETD3RK. In all our simuations time steps 0.0005, 0.00075 and 0.001 were employed. Smaller steps were used when the initial magnetic energy exceeded the kinetic energy and for the largest period factors M_1 and M_2.

3.2 Results of simulation

The general characteristics of the regimes obtained in our computations are shown in Table 3.2. The attractors or terminal regimes in the runs for the triple-period fluid cells are identified as non-generating stable hydrodynamic travelling waves whereas in the other periodicity cells these regimes are found to be chaotic. Some attractors during their temporal evolution exhibit a behaviour that is apparently regular for some finite time intervals and subsequently abandon it. This happens when their trajectory approaches an invariant object in the phase space, which is only mildly unstable (i.e., it has a low-dimensional unstable manifold and the growth rates of the non-decaying perturbation modes are small) and whose behaviour is regular (in our runs, these objects are either steady states or travelling waves). Following, e.g., Hutt *et al.*, 2017, we call such invariant objects as metastable states. If a trajectory finds itself close to a metastable state, it is repelled from the state – since the positive growth rates of the perturbation modes are small – along the unstable manifold in a relatively long time. But prior to leaving the neighbourhood of the metastable state, the trajectory imitates the quasi-regular behaviour of the trajectory (or trajectories) constituting the metastable state.

Metastable states are often encountered in the initial saturation phase and do not significantly affect the evolution. However, they can play an important role in shaping the behaviour of an attractor from repeated visitation of a trajectory. For instance, this occurs in our double-period simulations: the attractor is influenced by the presence of a hydrodynamically stable steady flow permanently staying in its vicinity in the run 2x1.1. In the other five runs a hydrodynamically stable flow of the travelling wave capable of magnetic field generation is repeatedly approached, and so the magnetic field does not ultimately decay.

The numerical values characterising an attractor or a metastable state in Table 3.2 are the actual energies obtained in the respective runs. Although some values are smaller than

Table 3.1 Minimum energy spectrum decay (orders of magnitude) in the simulated regimes of spatial periods $M_1\,L$ and $M_2\,L$ in x_1 and x_2, respectively. In each cell of the Table, three lines show the decays in the runs for the three different initial conditions that are labelled by the suffices 1, 2 and 3 in the names of the runs (see the beginning of the present section). In each line, commas separate three groups of numbers showing energy spectra decays of the flow, magnetic field and difference between the temperature and linear temperature profile for the fluid at equilibrium in the respective run. A single number in a group, or the first number in a group of two numbers refers to the value in the run with the standard resolution of $128M_1 \times 128M_2 \times 97$ harmonics in the two horizontal Cartesian axes and the vertical directions, respectively. The second number (after the slash) in a group of two numbers refers to the decay in the alternative resolution run involving $256 \times 256 \times 257$ harmonics, when such runs have been performed.

M_1, M_2	1	2	3	4
1		15.8,9.6,21.8	21.4,8.0,24.6	7.8/11.7,4.3/7.2,12.7/17.6
		15.2,12.2,21.7	9.9/13.4,6.3/10.5,15.9/18.8	7.2/11.6,4.3/6.3,12.0/16.8
		17.1,9.6,23.4	11.1/13.6,6.0/13.0,16.0/19.6	9.5/11.0,4.2/6.8,13.6/16.2
2	16.2,11.4,21.8	36.0,14.1,38.0	25.3,9.0,27.3	18.8/12.2,6.5/7.7,20.8/17.1
	15.0,8.5,20.5	37.8,15.0,38.1	24.6,7.9,26.3	18.3/11.2,6.5/7.6,20.0/15.7
	25.0,9.0,27.2	36.9,14.6,38.0	24.9,8.5,26.5	18.6/10.9,5.9/7.2,20.2/15.7
3	9.8/14.9,5.7/12.1,14.7/20.9	25.1,8.2,26.6	24.7,8.8,27.1	18.4/11.8,6.4/7.2,20.9/17.17
	9.4/14.1,5.6/11.4,14.7/20.1	25.0,8.7,26.9	19.7,8.4,25.3	18.2/10.2,6.1/7.6,20.3/15.6
	10.0/13.9,5.8/11.0,15.1/19.2	25.2,8.8,27.2	18.9,8.3,24.6	18.1/10.4,6.1/7.3,20.6/15.2
4	7.4/11.6,4.0/6.9,13.1/16.7	18.4/12.6,5.9/7.3,20.7/17.5	18.0/23.8,6.3/7.5,20.4/26.2	14.1/24.4,6.1/7.9,19.5/26.5
	7.3/10.8,4.2/7.3,13.2/16.3	18.8/11.4,6.0/7.5,20.5/16.6	18.6/11.0,6.0/6.9,20.3/14.6	18.1/10.8,5.6/6.8,20.5/15.5
	7.2/11.3,4.1/7.2,12.9/15.7	18.5/10.6,6.3/7.5,20.6/16.1	18.6/10.8,6.2/7.6,20.9/15.7	10.7/10.2,5.7/7.2,16.1/15.4

Table 3.2 Attractors (A, column 2) and metastable states (MS) in simulations in the (M_1, M_2)-periodicity boxes. Columns 3–6: kinetic, magnetic, heat (top to bottom lines in each cell of the table) initial, minimum, average and maximum energies, respectively, in the saturated regimes (A) or when the trajectory approaches the metastable states (MS). Column 7: time intervals when the trajectory is close to the attractor (the right end is then the time at which the run was terminated) or the metastable state. Column 8 (regime type): C—chaotic; HD—hydrodynamic; MHD—magnetohydrodynamic; Q—quasiperiodic; R1—steady rolls parallel to the x_2 axis; S—steady; S2—steady rolls parallel to the x_2 axis; and TW—travelling wave. Column 9: the symmetry pair for the flow and magnetic field.

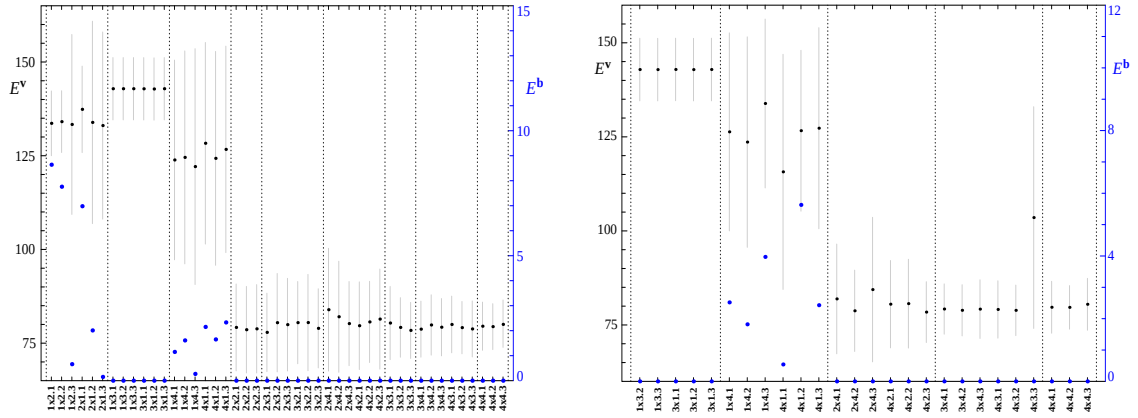
Run	A/MS	$E_{\rm in}$	$E_{\rm min}$	$E_{\rm av}$	$E_{\rm max}$	Time	Type	Symmetries
1×2.1	A	135.92	109.88	133.34	156.64	[75:1016.90]	MHD	
		4	4.22×10^{-6}	8.64	20.11			
		0.0476	0.0392	0.0452	0.05			
	MS		138.66	143.55	148.48	[85:152],	HD	$r\gamma^1_{L/2}$,
			1.54×10^{-5}	0.012	0.13	[655:663],	Q	$r\gamma^1_{L/2}$
			0.0441	0.0451	0.0464	[805:812]	TW	
	MS		130.80	130.87	130.93	[48:50.5]	HD	
			2.09×10^{-5}	1.63×10^{-4}	5.57×10^{-4}		S	
			0.0446	0.0446	0.0460			
1×2.2	A	100	112.36	134.09	154.22	[20:300.08]	MHD	
		25	0.0350	7.76	19.27			
		0.01	0.0413	0.0451	0.0493			
	MS		138.04	143.32	148.73	[24:38],	HD	$r\gamma^1_{L/2}$,
			0.0602	0.21	0.55	[120:128],	Q	$r\gamma^1_{L/2}$
			0.0441	0.0451	0.0477	[170:190]	TW	
1×2.3	A	100	34.38	133.34	158.94	[40:1691.82]	MHD	
		400	4.42×10^{-74}	0.66	19.65			
		0.01	0.0115	0.0426	0.0506			
	MS		138.84	143.56	148.48	[325:1560]	HD	$r\gamma^1_{L/2}$,
			4.60×10^{-71}	5.44×10^{-5}	1.23		Q	$r\gamma^1_{L/2}$
			0.0441	0.0451	0.0464		TW	
	MS		34.38	82.10	143.21	[40:293]	HD	
			5.97×10^{-74}	4.06×10^{-7}	8.07×10^{-5}		C	
			0.0115	0.0291	0.0478			
2×1.1	A	135.92	97.18	137.37	150.22	[40:341.81]	MHD	$\gamma^2_{L/2}$,
		3.76	0.91	6.98	23.93		C	$q\gamma^2_{L/2}$
		0.0476	0.0381	0.0475	0.0501			
	MS		150.14	150.19	150.22	[62.8:63.25]	HD	
			0.96	1.19	1.45	[182.4:182.7]	S2	
			0.0498	0.0499	0.0500	[298.8:299.1]	R1	
2×1.2	A	100	34.03	133.89	159.26	[40:2218.50]	MHD	
		25	1.61×10^{-73}	2.01	19.70			
		0.01	0.0109	0.0432	0.0505			
	MS		138.85	143.56	148.48	[300:1500],	HD	$r\gamma^2_{L/2}$,
			3.99×10^{-70}	5.83×10^{-6}	1.32	[1661:1678],	Q	$r\gamma^2_{L/2}$
			0.0441	0.0451	0.464	[1723:1740]	TW	
	MS		34.03	83.34	133.71	[40:270],	HD	
			3.14×10^{-73}	5.28×10^{-15}	1.12×10^{-12}	[2000:2218]	C	
			0.0109	0.0294	0.0445			

Run	A/MS	E_{in}	$E_{\min}$	E_{av}	$E_{\max}$	Time	Type	Symmetries
2x1.3	A	1	33.85	133.05	158.90	[40:3505.05]	MHD	
		400	1.71×10^{-158}	0.15	18.76			
		0.01	0.0110	0.0424	0.0505			
	MS		138.85	143.56	148.48	[670:3400]	HD	$r\gamma^2_{L/2}$,
			1.66×10^{-154}	3.04×10^{-6}	1.18		Q	$r\gamma^2_{L/2}$
			0.0441	0.0451	0.0464		TW	
	MS		33.85	82.71	153.46	[40:633.2]	HD	
			1.83×10^{-158}	7.26×10^{-12}	5.66×10^{-9}		C	
			0.0110	0.0292	0.0504			
1x3.1	A	135.92	132.64	142.87	156.49	[25:483.79]	HD	$r\gamma^1_{L/2}$,
		4	1.34×10^{-26}	6.34×10^{-5}	5.03×10^{-3}		Q	$r\gamma^1_{L/2}$
		0.0476	0.0435	0.0453	0.0476		TW	
1x3.2	A	100	132.58	142.87	156.50	[100:558.16]	HD	$r\gamma^1_{L/2}$,
		25	7.25×10^{-26}	5.26×10^{-4}	0.0512		Q	$r\gamma^1_{L/2}$
		0.01	0.0435	0.0453	0.0477		TW	
1x3.3	A	1	132.64	142.87	156.46	[85:1992.79]	HD	$r\gamma^1_{L/2}$,
		400	3.78×10^{-103}	1.86×10^{-8}	6.68×10^{-6}		Q	$r\gamma^1_{L/2}$
		0.01	0.0435	0.0453	0.0476		TW	
3x1.1	A	135.92	132.59	142.85	156.49	[1275:1348.48]	HD	$r\gamma^2_{L/2}$,
		3.76	3.06×10^{-6}	2.71×10^{-3}	0.0432		Q	$r\gamma^2_{L/2}$
		0.0476	0.0435	0.0453	0.0477		TW	
3x1.2	A	100	132.64	142.80	156.46	[560:597.69]	HD	$r\gamma^2_{L/2}$,
		25	5.42×10^{-6}	2.25×10^{-4}	1.76×10^{-3}		Q	$r\gamma^2_{L/2}$
		0.01	0.0435	0.0453	0.0476		TW	
	MS		139.71	139.73	139.85	[8:16]	HD	$\gamma^2_{L/2}$,
			3.60×10^{-5}	1.38×10^{-4}	5.35×10^{-4}		S2	$\gamma^2_{L/2}$
			0.0468	0.0468	0.0468			
3x1.3	A	100	132.51	142.87	156.63	[160:561.06]	HD	$r\gamma^2_{L/2}$,
		400	2.54×10^{-22}	1.79×10^{-3}	0.0991		Q	$r\gamma^2_{L/2}$
		0.01	0.0435	0.0453	0.0477		TW	
	MS		139.71	139.72	139.74	[10:15.5]	HD	
			5.84×10^{-5}	2.84×10^{-4}	1.27×10^{-3}		S2	
			0.0468	0.0468	0.0468			
1x4.1	A	135.92	56.81	123.87	183.43	[50:202.18]	MHD	
		4	1.92×10^{-5}	1.15	14.76		C	
		0.0476	0.0180	0.0359	0.0487			
	MS		131.67	142.41	151.88	[115:121],	HD	
			1.92×10^{-4}	8.71×10^{-4}	3.95×10^{-3}	[9:15.5]	Q	
			0.0439	0.0460	0.0478			
1x4.2	A	100	53.62	128.35	215.63	[50:200.89]	MHD	
		25	3.88×10^{-4}	3.21	17.27		C	
		0.01	0.0183	0.0389	0.05			
1x4.3	A	1	43.52	122.10	200.92	[50:211.14]	MHD	
		400	3.01×10^{-7}	0.27	2.92		C	
		0.01	0.0136	0.0330	0.0485			

Run	A/MS	E_{in}	E_{min}	E_{av}	E_{max}	Time	Type	Symmetries
4x1.1	A	135.92 4 0.0476	49.93 1.77×10^{-6} 0.0168	128.33 2.15 0.0391	203.82 16.71 0.0487	[15:381.79]	MHD C	
	MS		131.45 1.92×10^{-4} 0.0444	142.83 4.22×10^{-4} 0.0462	152.27 8.51×10^{-4} 0.0476	[223:242]	HD Q TW	
4x1.2	A	100 25 0.01	51.86 3.60×10^{-6} 0.0172	124.30 1.65 0.0358	205.75 16.04 0.0499	[50:310.23]	MHD C	
	MS		146.06 7.85×10^{-3} 0.0484	147.19 0.0799 0.0486	147.87 0.30 0.0487	[246:247.75]	HD S2	
4x1.3	A	100 900 0.01	51.20 9.13×10^{-6} 0.0179	126.69 2.33 0.0388	187.53 15.10 0.0487	[25:286.47]	MHD C	
	MS		130.19 8.92×10^{-4} 0.0441	143.60 1.78×10^{-3} 0.0466	151.34 3.03×10^{-3} 0.0483	[46:51.5]	HD Q	
	MS		122.37 0.47 0.0419	123.00 0.89 0.0419	123.76 1.50 0.0419	[12.5:15]	HD S2	$\gamma^2_{L/2}$, $\gamma^2_{L/2}$
2x2.1	A	135.92 4 0.0476	47.29 9.49×10^{-244} 0.0160	79.2 2.01×10^{-11} 0.0267	131.63 6.90×10^{-9} 0.0393	[40:1001.34]	HD C	
2x2.2	A	100 25 0.01	48.34 1.77×10^{-75} 0.0166	78.59 3.59×10^{-12} 0.0265	118.35 4.69×10^{-10} 0.0373	[40:331.19]	HD C	
2x2.3	A	1 1225 0.01	46.40 4.30×10^{-96} 0.0171	78.83 1.18×10^{-11} 0.0265	135.50 1.98×10^{-9} 0.0399	[40:409.44]	HD C	
2x3.1	A	135.92 4 0.0476	55.55 1.06×10^{-35} 0.0176	77.88 3.82×10^{-8} 0.0258	135.61 1.60×10^{-6} 0.0358	[40:204.50]	HD C	
2x3.2	A	100 25 0.01	53.37 8.70×10^{-109} 0.0177	80.47 1.47×10^{-11} 0.0263	159.21 7.92×10^{-9} 0.0413	[40:540.45]	HD C	
2x3.3	A	1 400 0.01	47.40 8.98×10^{-76} 0.0161	79.94 1.51×10^{-11} 0.0263	149.96 2.21×10^{-9} 0.04	[40:417.59]	HD C	
3x2.1	A	135.92 4 0.0476	55.47 1.56×10^{-48} 0.0174	80.48 4.52×10^{-9} 0.0264	136.12 1.16×10^{-6} 0.0395	[40:318.22]	HD C	
3x2.2	A	100 25 0.01	53.78 1.79×10^{-66} 0.0188	80.50 1.12×10^{-9} 0.0263	149.84 2.41×10^{-7} 0.0394	[40:364.87]	HD C	
3x2.3	A	100 400 0.01	54.77 1.61×10^{-65} 0.0183	78.96 1.10×10^{-7} 0.0261	146.56 2.59×10^{-5} 0.0382	[40:363.77]	HD C	

Run	A/MS	E_{in}	E_{min}	E_{av}	E_{max}	Time	Type	Symmetries
2x4.1	A	135.92 4 0.0476	56.35 1.25×10^{-27} 0.0171	83.93 1.77×10^{-6} 0.0263	165.97 2.77×10^{-4} 0.0382	[40:244.38]	HD C	
2x4.2	A	100 25 0.01	54.33 1.94×10^{-60} 0.0196	82.05 9.28×10^{-7} 0.0262	159.80 1.62×10^{-4} 0.0375	[40:381.79]	HD C	
2x4.3	A	100 400 0.01	57.54 6.23×10^{-49} 0.0188	80.22 1.55×10^{-7} 0.0259	152.08 1.46×10^{-5} 0.0348	[40:326.87]	HD C	
4x2.1	A	135.92 4 0.0476	56.54 6.00×10^{-36} 0.0194	79.66 1.60×10^{-5} 0.0261	153.39 8.01×10^{-4} 0.0355	[40:290.67]	HD C	
4x2.2	A	225 25 0.01	55.31 1.49×10^{-51} 0.0194	80.67 7.81×10^{-6} 0.0260	146.53 1.50×10^{-3} 0.0349	[40:366.17]	HD C	
4x2.3	A	1 100 0.01	52.71 1.88×10^{-58} 0.0161	81.42 6.09×10^{-7} 0.0260	164.80 7.67×10^{-5} 0.0379	[40:391.69]	HD C	
3x3.1	A	135.92 4 0.0476	59.34 4.48×10^{-27} 0.0195	80.3 6.42×10^{-7} 0.0262	133.18 5.76×10^{-5} 0.0354	[40:201.10]	HD C	
3x3.2	A	100 25 0.01	55.04 9.71×10^{-49} 0.0186	79.20 4.57×10^{-6} 0.0259	108.71 4.75×10^{-4} 0.0327	[40:321.50]	HD C	
3x3.3	A	100 400 0.01	56.12 3.92×10^{-40} 0.0182	78.41 2.25×10^{-5} 0.0259	106.72 2.38×10^{-3} 0.0334	[40:312.34]	HD C	
3x4.1	A	135.92 4 0.0476	60.18 6.08×10^{-30} 0.0197	78.74 3.19×10^{-6} 0.0256	120.72 3.62×10^{-4} 0.0319	[40:200.17]	HD C	
3x4.2	A	225 100 0.01	63.75 1.11×10^{-40} 0.0203	79.86 1.20×10^{-6} 0.0259	121.48 8.99×10^{-5} 0.0323	[40:295.39]	HD C	
3x4.3	A	1 900 0.01	62.18 5.22×10^{-38} 0.0206	79.26 7.20×10^{-6} 0.0258	117.60 5.29×10^{-4} 0.0327	[40:285.44]	HD C	
4x3.1	A	135.92 4 0.0476	60.75 6.26×10^{-25} 0.0190	79.99 1.07×10^{-5} 0.0259	112.65 5.83×10^{-4} 0.0338	[40:200.01]	HD C	
4x3.2	A	225 100 0.01	59.81 2.34×10^{-44} 0.0202	79.15 7.68×10^{-6} 0.0258	110.28 5.46×10^{-4} 0.0321	[40:337.28]	HD C	
4x3.3	A	100 400 0.01	53.78 1.44×10^{-33} 0.0183	78.80 6.20×10^{-7} 0.0257	109.92 5.71×10^{-5} 0.0324	[40:296.86]	HD C	
4x4.1	A	135.92 4 0.0476	65.55 5.33×10^{-23} 0.0212	79.53 1.03×10^{-6} 0.0257	106.56 7.88×10^{-5} 0.0313	[40:200.51]	HD C	

Run	A/MS	E_{in}	E_{min}	E_{av}	E_{max}	Time	Type	Symmetries
4×4.2	A	225	61.65	79.41	105.68	[40:528.55]	HD	
		100	8.18×10^{-54}	3.14×10^{-6}	5.88×10^{-4}		C	
		0.01	0.0201	0.0258	0.0333			
4×4.3	A	100	62.51	80.20	113.30	[40:342.84]	HD	
		400	1.16×10^{-38}	2.58×10^{-5}	2.61×10^{-3}		C	
		0.01	0.0210	0.0259	0.0319			



(a) (b)

Figure 3.2 Space- and time-averaged kinetic (smaller black dots, left axis) and magnetic (larger blue dots, right axis) energies for the attractors obtained in the runs with the standard ($128M_1 \times 128M_2 \times 97$ harmonics), (a), and alternative ($256 \times 256 \times 257$ harmonics), (b), resolution. Gray vertical lines: r.m.s. deviations of the space-averaged kinetic energy from their time averages. Thin vertical dotted lines separate the runs into groups of regimes residing in identical periodicity cells.

machine epsilon we note that they are meaningful. This explains why the magnetic energy of an attractor or a state is strictly positive even when it is classified as hydrodynamic. While implementing time-stepping only the values at the previous computational time are used. So all values, eventually with round-off errors, that we add up to yield a resultant or a newly computed value have comparable orders of smallness and end up with a value that is much smaller.

Further, our simulations have shown that convection continues to act as dynamos only in elongated (M_1, M_2)-periodicity boxes, i.e., for $M_1 = 1$ or $M_2 = 1$, where the length of the complementary large-scale period fails to constrain the ability of the flow to generate magnetic field. The mean, over space and time, of the kinetic and magnetic energy densities upon saturation are shown in Figure 3.2 for both the resolutions. They show that only sufficiently vigorous flows (whose mean kinetic energy density is over 100) generate magnetic field. But, this is only a necessary condition: flows obtained in the simulations for the periodicity cells 1×3 and 3×1 satisfy this condition, nonetheless they do not act as dynamos.

In the subsequent investigation, for each run we also have computed estimates of the magnetic Reynolds number $\text{Rm} = l|\mathbf{v}|/\eta$, where l is the characteristic length scale of the

flow, $|\mathbf{v}|$, the characteristic flow velocity and $\eta = PP_m^{-1}$, the molecular magnetic diffusivity. Within the framework of the approach of [Doering and Gibbon, 1995], we have computed four different estimates of the characteristic length scale:

$$\begin{aligned}\ell_1 &= \left((t_1 - t_0)^{-1} \int_{t_0}^{t1} \left(\int_{\mathscr{C}} |\mathbf{v}|^2 \, \mathrm{d}\mathbf{x} \Big/ \int_{\mathscr{C}} |\nabla \mathbf{v}|^2 \mathrm{d}\mathbf{x} \right) \mathrm{d}t \right)^{1/2} \\ &= \left((t_1 - t_0)^{-1} \int_{t_0}^{t_1} \left(\sum_n \delta_{\mathbf{n}} \mid \mathbf{v}_{\mathbf{n}} \mid^2 \right) \left(\sum_{\mathbf{n}} \delta_{\mathbf{n}} (\alpha_1^2 n_1^2 + \alpha_2^2 n_2^2 + \alpha_3^2 n_3^2) \mid \mathbf{v}_{\mathbf{n}} \mid^2 \right)^{-1} \mathrm{d}t \right)^{1/2}, \end{aligned} \tag{3.12a}$$

$$\begin{aligned}\ell_2 &= \left(\int_{t_0}^{t1} \int_{\mathscr{C}} |\mathbf{v}|^2 \, \mathrm{d}\mathbf{x} \, \mathrm{d}t \Big/ \int_{t_0}^{t1} \int_{\mathscr{C}} |\nabla \mathbf{v}|^2 \mathrm{d}\mathbf{x} \, \mathrm{d}t \right)^{1/2} \\ &= \left(\int_{t_0}^{t_1} \left(\sum_{\mathbf{n}} \delta_{\mathbf{n}} \mid \mathbf{v}_{\mathbf{n}} \mid^2 \right) \mathrm{d}t \Big/ \int_{t_0}^{t_1} \left(\sum_{\mathbf{n}} \delta_{\mathbf{n}} (\alpha_1^2 n_1^2 + \alpha_2^2 n_2^2 + \alpha_3^2 n_3^2) \mid \mathbf{v}_{\mathbf{n}} \mid^2 \right) \mathrm{d}t \right)^{1/2}, \end{aligned} \tag{3.12b}$$

$$\begin{aligned}\ell_3 &= \left((t_1 - t_0)^{-1} \int_{t_0}^{t1} \left(\int_{\mathscr{C}} |\mathbf{v}|^2 \mathrm{d}\mathbf{x} \Big/ \int_{\mathscr{C}} \left| \left(\frac{\partial^2 \mathbf{v}}{\partial x_1^2}, \frac{\partial^2 \mathbf{v}}{\partial x_2^2}, \frac{\partial^2 \mathbf{v}}{\partial x_3^2} \right) \right|^2 \mathrm{d}\mathbf{x} \right) \mathrm{d}t \right)^{1/4} \\ &= \left((t_1 - t_0)^{-1} \int_{t_0}^{t1} \left(\sum_{\mathbf{n}} \delta_{\mathbf{n}} \mid \mathbf{v}_{\mathbf{n}} \mid^2 \right) \left(\sum_{\mathbf{n}} \delta_{\mathbf{n}} (\alpha_1^4 n_1^4 + \alpha_2^4 n_2^4 + \alpha_3^4 n_3^4) \mid \mathbf{v}_{\mathbf{n}} \mid^2 \right)^{-1} \mathrm{d}t \right)^{1/4}, \end{aligned} \tag{3.12c}$$

$$\begin{aligned}\ell_4 &= \left(\int_{t_0}^{t1} \int_{\mathscr{C}} |\mathbf{v}|^2 \, \mathrm{d}\mathbf{x} \, \mathrm{d}t \Big/ \int_{t_0}^{t1} \int_{\mathscr{C}} \left| \left(\frac{\partial^2 \mathbf{v}}{\partial x_1^2}, \frac{\partial^2 \mathbf{v}}{\partial x_2^2}, \frac{\partial^2 \mathbf{v}}{\partial x_3^2} \right) \right|^2 \mathrm{d}\mathbf{x} \, \mathrm{d}t \right)^{1/4} \\ &= \left(\int_{t_0}^{t_1} \left(\sum_{\mathbf{n}} \delta_{\mathbf{n}} \mid \mathbf{v}_{\mathbf{n}} \mid^2 \right) \mathrm{d}t \Big/ \int_{t_0}^{t_1} \left(\sum_{\mathbf{n}} \delta_{\mathbf{n}} (\alpha_1^4 n_1^4 + \alpha_2^4 n_2^4 + \alpha_3^4 n_3^4) \mid \mathbf{v}_{\mathbf{n}} \mid^2 \right) \mathrm{d}t \right)^{1/4}, \end{aligned} \tag{3.12d}$$

where $\delta_{\mathbf{n}} = 2$ if $n_3 = 0$, and $\delta_{\mathbf{n}} = 1$ otherwise. The characteristic flow velocity $|\mathbf{v}|$ is defined as the root-mean square velocity averaged over the time interval for which the magnetic Reynolds number is estimated. Thus four mean estimates of the magnetic Reynolds number, $\mathrm{Rm}_n = \ell_n |\mathbf{v}| / \eta$, are obtained.

From Table 3.3 we see that in general Rm_1 and Rm_2 are smaller than Rm_3 and Rm_4. The differences $|\mathrm{Rm}_1 - \mathrm{Rm}_2|$ and $|\mathrm{Rm}_3 - \mathrm{Rm}_4|$ are also small compared to the difference between Rm_1 and Rm_3, or between Rm_2 and Rm_4. This behaviour of the estimates of the magnetic Reynolds number Rm_n imitates the behaviour of the estimates of the length scales ℓ_n. The discrepancy $|\mathrm{Rm}_3 - \mathrm{Rm}_4|$ is smaller for the non-generating regimes (below 0.018) than for the generating regimes (which is of magnitude greater than and upto 0.16).

The characteristic length scales ℓ_n are small and tend to decrease relative to the horizontal size of the periodicity boxes. This behaviour of the length scales is against our intuition and expectations. All our solutions from the non-elongated $M_1 \times M_2$ periodicity boxes have

Table 3.3 Magnetic Reynolds number estimates in runs for perturbations of moderate scale separation (in the order of increasing Rm_4). Column G: "M" indicates, that magnetic field generation does not terminate in the perturbed regime; Interval: the time interval, for which the magnetic Reynolds number estimates are computed; ℓ_n: estimates of the internal spatial scale of the perturbed regime, see (3.12); Rm_n the estimate of the magnetic Reynolds number based on l_n; $|\,\mathbf{v}\,|$: the characteristic flow velocity in the time interval under consideration.

Run	G	Interval	ℓ_1	Rm_1	ℓ_2	Rm_2	ℓ_3	Rm_3	ℓ_4	Rm_4	$\lvert\mathbf{v}\rvert$
4x4.2		51–528.55	0.223	22.413	0.223	22.416	0.206	20.722	0.206	20.720	12.58
4x2.1		232.40–290.65	0.219	21.769	0.219	21.770	0.241	23.988	0.241	23.983	12.43
3x3.3		220.46–312.34	0.220	21.843	0.220	21.842	0.241	24.011	0.241	24.004	12.43
2x2.3		310.15–409.44	0.219	21.879	0.220	21.885	0.242	24.115	0.242	24.103	12.46
2x3.1		100–204.50	0.219	21.920	0.219	21.923	0.241	24.113	0.241	24.108	12.50
4x3.3		223.17–296.86	0.221	22.050	0.221	22.047	0.242	24.176	0.242	24.167	12.48
2x3.2		505.52–540.45	0.220	22.092	0.220	22.101	0.242	24.280	0.242	24.278	12.53
3x2.3		251.39–363.77	0.221	22.183	0.221	22.190	0.243	24.365	0.242	24.358	12.55
3x4.1		42.18–200.17	0.222	22.273	0.222	22.277	0.243	24.377	0.243	24.369	12.55
3x4.3		219.42–295.39	0.221	22.281	0.221	22.283	0.243	24.400	0.242	24.393	12.57
4x3.2		290.74–337.38	0.222	22.337	0.222	22.332	0.243	24.414	0.243	24.406	12.56
4x4.3		201–336	0.223	22.385	0.223	22.385	0.243	24.439	0.243	24.436	12.56
4x3.1		42–200.02	0.223	22.396	0.223	22.397	0.243	24.481	0.243	24.480	12.57
3x4.2		246.87–285.44	0.223	22.454	0.223	22.446	0.243	24.497	0.243	24.486	12.57
3x3.2		278.28–321	0.224	22.467	0.223	22.459	0.244	24.533	0.244	24.518	12.56
2x3.3		309.16–417.59	0.221	22.382	0.221	22.415	0.243	24.554	0.243	24.560	12.65
3x3.1		42.91–201.10	0.222	22.468	0.222	22.473	0.243	24.574	0.243	24.568	12.62
2x2.2		200–331.19	0.221	22.333	0.221	22.356	0.245	24.698	0.244	24.680	12.62
4x4.1		100–200.51	0.223	22.619	0.224	22.623	0.244	24.689	0.244	24.685	12.65
2x2.1		719.70–800.70	0.221	22.484	0.221	22.502	0.243	24.788	0.243	24.779	12.73
2x4.2		317.09–381	0.223	22.741	0.224	22.784	0.243	24.795	0.243	24.810	12.73
2x4.3		223.93–326	0.224	22.806	0.224	22.816	0.244	24.850	0.244	24.846	12.72
3x2.1		231.77–318	0.223	22.723	0.223	22.728	0.244	24.863	0.244	24.857	12.72
4x2.3		268.69–366.17	0.224	22.855	0.224	22.860	0.244	24.902	0.244	24.895	12.73
2x4.1		102–207.69	0.225	22.959	0.226	22.973	0.245	24.954	0.245	24.961	12.73
3x2.2		304.32–364	0.223	22.978	0.224	23.022	0.244	25.093	0.244	25.104	12.86
4x2.2		334.41–391.69	0.225	23.106	0.226	23.135	0.245	25.106	0.245	25.113	12.81
3x1.1	M	100–249.39	0.227	27.379	0.227	27.279	0.247	29.761	0.246	29.608	15.04
2x1.2	M	1946.90–1980.90	0.227	26.638	0.228	26.815	0.254	29.868	0.255	29.949	14.68
3x1.2	M	285.85–377.67	0.227	28.423	0.226	28.342	0.245	30.675	0.244	30.537	15.65
1x4.3	M	130–211.14	0.242	29.726	0.243	29.786	0.254	31.235	0.254	31.163	15.34
4x1.3	M	108.81–241.99	0.233	29.797	0.233	29.727	0.248	31.701	0.247	31.589	15.96
4x1.2	M	200–310.23	0.238	30.258	0.238	30.170	0.253	32.107	0.252	31.986	15.85
4x1.1	M	168–330	0.238	29.814	0.238	29.770	0.256	32.060	0.256	32.015	15.65
1x4.1	M	140–192	0.242	30.969	0.241	30.894	0.252	32.300	0.251	32.165	16.00
1x4.2	M	120–200	0.237	31.140	0.236	31.098	0.259	34.047	0.257	33.888	16.45
1x2.1	M	185–245	0.238	30.339	0.238	30.333	0.267	34.018	0.267	34.003	15.92
1x2.3	M	1606–1691	0.238	30.534	0.238	30.526	0.267	34.206	0.267	34.181	16.01
2x1.3	M	3460–3505.05	0.239	30.794	0.238	30.787	0.266	34.407	0.266	34.383	16.13
1x2.2	M	50–300	0.239	31.193	0.239	31.188	0.266	34.715	0.266	34.684	16.32
2x1.1	M	152–277.81	0.248	33.000	0.248	32.987	0.261	34.777	0.261	34.739	16.64
1x3.3		50–354.35	0.234	31.566	0.234	31.568	0.263	35.499	0.263	35.483	16.87
1x3.1		25–487.12	0.234	31.668	0.234	31.659	0.263	35.589	0.263	35.566	16.90
1x3.2		83–558.16	0.234	31.669	0.234	31.660	0.263	35.590	0.263	35.567	16.90
3x1.3		330.20–365.60	0.234	31.676	0.234	31.667	0.263	35.597	0.263	35.574	16.90

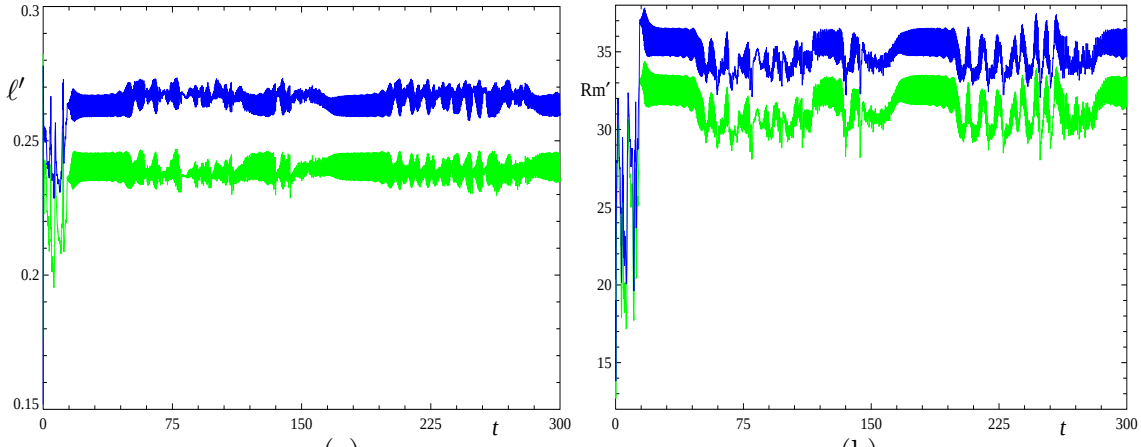


Figure 3.3 Temporal behaviour of the time-dependent estimates of the length scales (3.12), ℓ_1' (green line) and ℓ_3' (blue line), (a), and of the magnetic Reynolds number, Rm_1' (green line) and Rm_3' (blue line), in the run 1×2.2 .

chaotic attractors. The small size of the flow structures, their variability and the resultant spatial intermittency do not contribute to a concerted action of the flow structures at various places in the periodicity box. Apparently, this is the main reason for the lack of dynamo action in such boxes.

The magnetic Reynolds number estimates Rm_n sharply separate the regimes in elongated periodicity boxes from those in larger-size boxes (where magnetic generation fails) at the thresholds $\mathrm{Rm}_1 = \mathrm{Rm}_2 = 25$, $\mathrm{Rm}_3 = \mathrm{Rm}_4 = 27$. While these values are higher than the critical Reynolds numbers Rm_c for the onset of magnetic field generation by Beltrami flows ($\mathrm{Rm}_c = 9$ for the 1:1:1 ABC flow [Arnold and Korkina, 1983] and Rm_c=16 for a Beltrami flow in a sphere [Zheligovsky, 1993]), they are significantly lower than the critical numbers of many other kinematic dynamos presented in the literature (e.g., cf. Rm_c=515.63 for the Christopherson flow [Zheligovsky and Galloway, 1998]). The characteristic flow velocity which contributes significantly to the magnetic Reynolds numbers decrease substantially with the horizontal area of the periodicity box, implying that in larger periodicity boxes convective motion tend to become less vigorous and affect their ability to generate magnetic field.

For $t_1 \to t_0$, the length scale estimates ℓ_1 and ℓ_2 have the same limits, which we denote by $\ell_1'(t_0)$; similarly, the limits of ℓ_3 and ℓ_4 are the same, which we denote by $\ell_3'(t_0)$. By employing the root-mean square velocity (averaging over the periodicity cell) as the momentary characteristic velocity $|\mathbf{v}(t)|$, we define the time-dependent estimates of the magnetic Reynolds number, $\mathrm{Rm}_n'(t) = \ell_n'(t)|\mathbf{v}(t)|/\eta$.

Figure 3.3 illustrates the temporal behaviour of the estimates ℓ_1', ℓ_3', Rm_1' and Rm_3' in the run 1×2.2. Although the behaviour of ℓ_j' and Rm_j' are quite similar, Rm_j' relatively has a larger variation in amplitude than ℓ_j', and the behaviour of the two quantities differs significantly in details. As an example, while ℓ_3' experiences an overall increase towards the end of evolution shown in Figure 3.3, ℓ_1', Rm_1' and Rm_3' are on average growing.

In the subsequent sections we will consider in detail the generating regimes of thermal convection in the presence of magnetic field - at least initially.

3.2.1 Regimes with the periodicites 1×2/2×1

The six runs in the 2×1 and 1×2 periodicity fluid volumes are referred to as the "double period" runs. The attractors and metastable states encountered in these periodicity cells are presented in Table 3.2. The temporal behaviour of the kinetic and magnetic energy in the run 2×1.2 (see Figure 3.4(i)) indicates that the regime encountered in these periodicity cells is a succession of three distinct phases separated by short transitions: first, the magnetic field decay phase ($50 < t < 250$ and $2000 < t \leq 2200$ in figure), second, the generation phase ($300 < t < 1500$) and third, the saturation phase ($1550 < t < 1950$). Each phase is characterised by well-defined distinct values of the average kinetic energy, length scales and the respective estimates of the magnetic Reynolds numbers.

The generation phase occurs when the trajectory in the phase space finds itself close to a travelling wave transporting a time-periodic flow capable of kinematic magnetic field generation; by contrast, we have failed to identify an analogous non-dynamo underlying hydrodynamic object of a simple temporal structure in the decay phase.

Table 3.4 presents the estimates of length scales and the magnetic Reynolds numbers of the 3 phases, which are indicated by their labels: D (the magnetic field decay phase), G (the generation phase), S (the saturation phase) and T (the initial transitory phase). These phases are identified in the evolutions of convective dynamo in all our double-period runs except 2×1.1, and their lengths or durations differ; So in order to reliably estimate the mean values of an entire trajectory in the attractor requires an integration up to at least order 10^5.

Fully non-linear convective dynamo regimes approaching flows of simple temporal structure (such as steady states, periodic or quasiperiodic flows) for certain finite times and that which can generate magnetic field kinematically were detected numerically (see for instance, Chertovskih *et al.*, 2010, 2015, and 2017). Remarkably, in [Kosobokov, 1995] a development towards a periodic regime by the geodynamo was also observed in the paleomagnetic data in durations of the Earth's geomagnetic epochs.

From table 3.4, we see that in the G phases the values of ℓ_n and Rm_n for the different regimes are consistent: for each n, the Rm_n values differ by less than 0.1 between runs. The Rm_1 and Rm_2 values characterising the D phases differ by less than 0.3, and the Rm_3 and Rm_4 values by less than 0.2.

Evidently, even for the G phases these values do not coincide because

- the time intervals, that were used to compute the values in the vicinity of quasiperiodic metastable states were not synchronised between runs taking the quasiperiodicity into account;

- in phase G the flow closely approaches a travelling wave responsible for the respective hydrodynamic metastable state and it does not coincide with it;

Table 3.4 Magnetic Reynolds number estimates in various phases of the regimes observed in the runs with the periodicities 1×2/2×1. Column Interval: the time interval, for which the magnetic Reynolds number estimates are computed; ℓ_n: estimates (3.12) of the internal spatial scale of the perturbed regime; Rm_n the estimate of the magnetic Reynolds number based on ℓ_n; $|\mathbf{v}|$: the characteristic flow velocity in the time interval under consideration.

Run	Phase	Interval	ℓ_1	Rm_1	ℓ_2	Rm_2	ℓ_3	Rm_3	ℓ_4	Rm_4	$\lvert\mathbf{v}\rvert$
1×2.1	T	10–65	0.215	22.690	0.214	22.615	0.249	26.325	0.249	26.265	13.20
	G	75–170	0.240	32.566	0.240	32.553	0.263	35.653	0.263	35.638	16.93
	S	185–250	0.238	30.425	0.238	30.418	0.267	34.099	0.267	34.083	15.96
1×2.2	T	0–13.8	0.220	24.205	0.220	24.172	0.245	26.928	0.244	26.765	13.72
	G	20–40	0.240	32.557	0.240	32.543	0.263	35.665	0.263	35.648	16.93
	S	50–115	0.238	30.439	0.238	30.432	0.267	34.165	0.267	34.151	15.98
1×2.3	D	10–293	0.214	22.367	0.214	22.350	0.246	25.699	0.246	25.627	13.03
	G	300–1597	0.240	32.584	0.240	32.571	0.263	35.667	0.263	35.653	16.94
	S	1606–1691.82	0.238	30.534	0.238	30.526	0.267	34.206	0.267	34.181	16.01
2×1.1	T	0–8	0.244	32.150	0.244	32.151	0.260	34.363	0.260	34.362	16.50
	S	12–277.81	0.248	33.010	0.248	32.993	0.261	34.789	0.261	34.737	16.65
2×1.2	D	40–270	0.216	22.533	0.214	22.374	0.246	25.743	0.245	25.585	13.06
	G	285–1555	0.241	32.658	0.240	32.621	0.263	35.727	0.263	35.699	16.95
	S	1570–1962	0.239	31.254	0.239	31.170	0.266	34.814	0.266	34.743	16.33
2×1.3	D	40–633.2	0.215	22.265	0.214	22.161	0.246	25.559	0.245	25.448	12.97
	G	646–3445	0.240	32.611	0.240	32.579	0.263	35.686	0.263	35.662	16.95
	S	3460–3505.05	0.239	30.794	0.238	30.787	0.266	34.407	0.266	34.383	16.13

- the magnetic field affects the flow especially in the beginning of phases D and towards the end of phases G.

The Rm_n values obtained for phases S are less consistent and this can be expected.

Two different attractors are identified in the six runs of the double-period fluid volumes and this explains the difference of the Rm_n estimates in the S phases. The coincidence of attractors in the five runs was verified by comparing some of their quantitative characteristics: for instance, the same growth and decay rates in the respective phases G and D, and the same kinetic energy levels in each phase; this remains true for more subtle quantities, such as the characteristic length scales (3.12) and the estimates of the magnetic Reynolds numbers.

Figure 3.4 illustrates the exhaustive features of individual trajectories approaching the attractor in the 1×2/2×1 periodic simulations. The similarity of regimes is evident in the plots of kinetic and magnetic energies in the double period runs 1×2.3, 2×1.2 and 2×1.3, (see Fig. 3.4(e), (i) and (k)) as well as in the projections of trajectories on the kinetic and magnetic energy plane, (Fig. 3.4(f), (j) and (l)).

The G phases are made up of relatively thin “cylinders”, i.e., spirals that are densely populated by almost horizontal, slowly rising relatively narrow loops. The subsequent upper sections, gradually shifted to the left, constitute the S phases. The irregularly entangled, relatively wide curves in plots Fig. 3.4(b) and (d) in the left column represent the initial T phases, and the wide “cylinders”, i.e., spirals composed of relatively less densely populated, almost horizontal, slowly lowering, relatively large-radius loops (in the run 1×2.3 , see

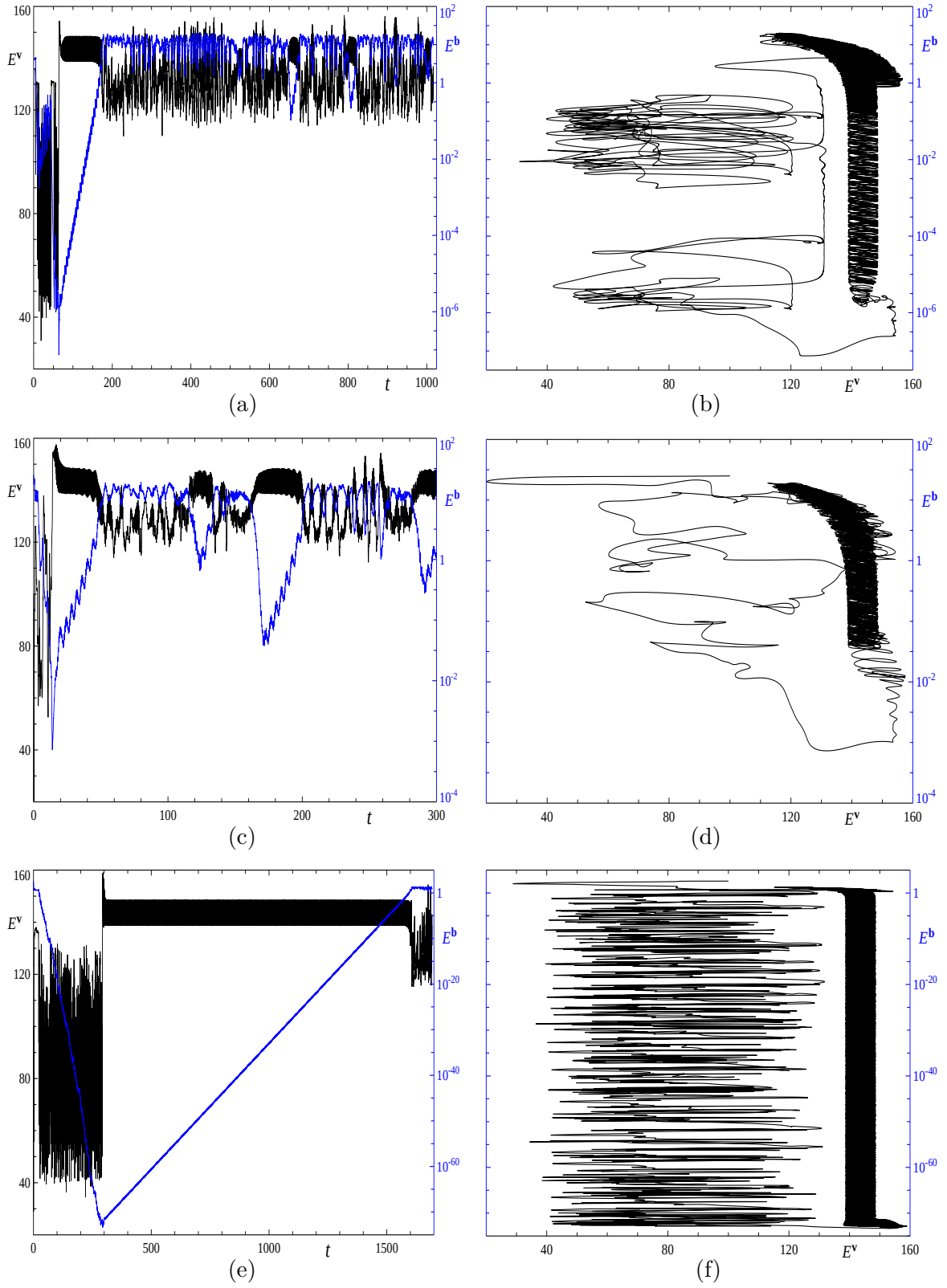


Figure 3.4 The figure is continued at the next page.

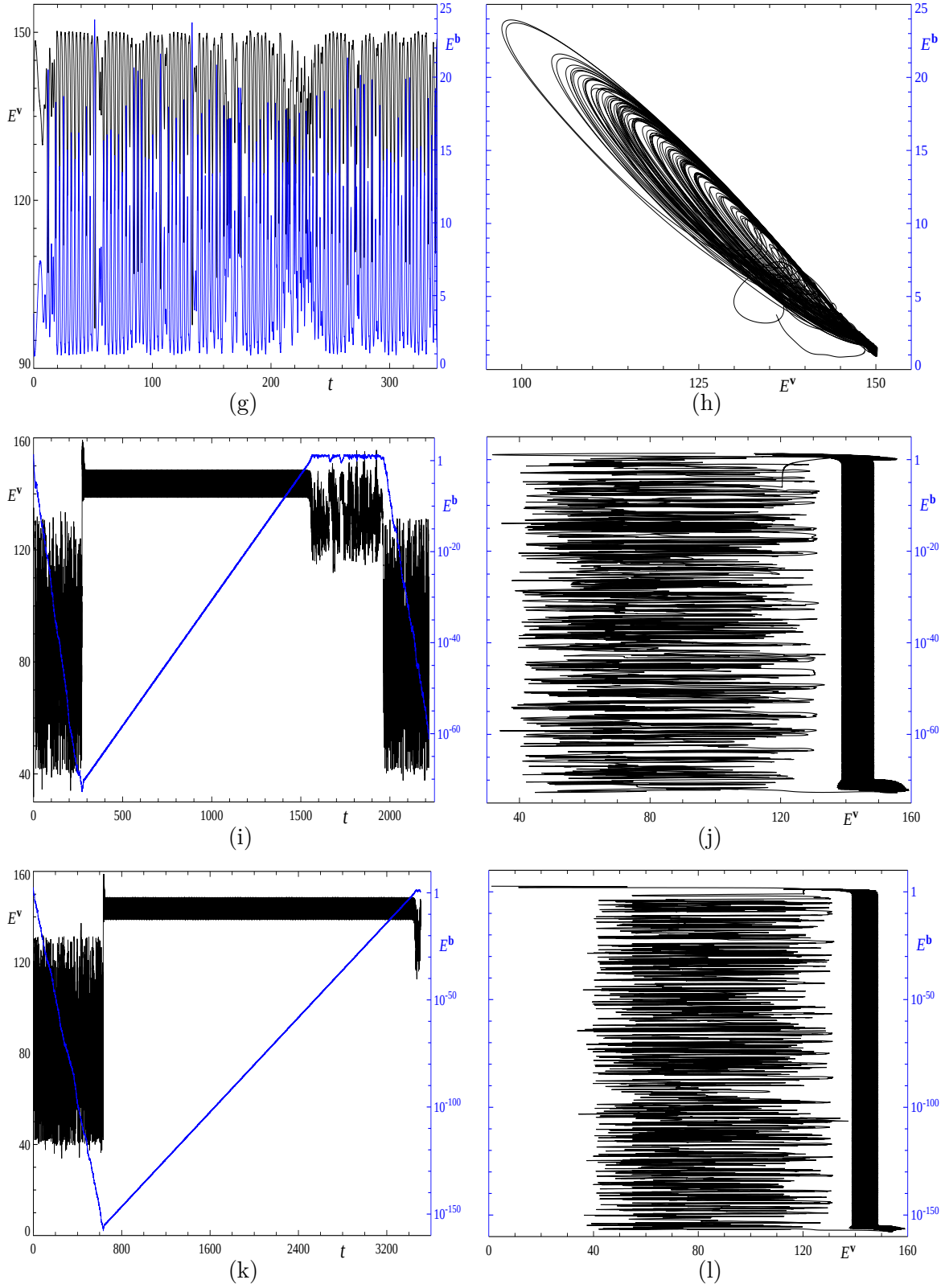
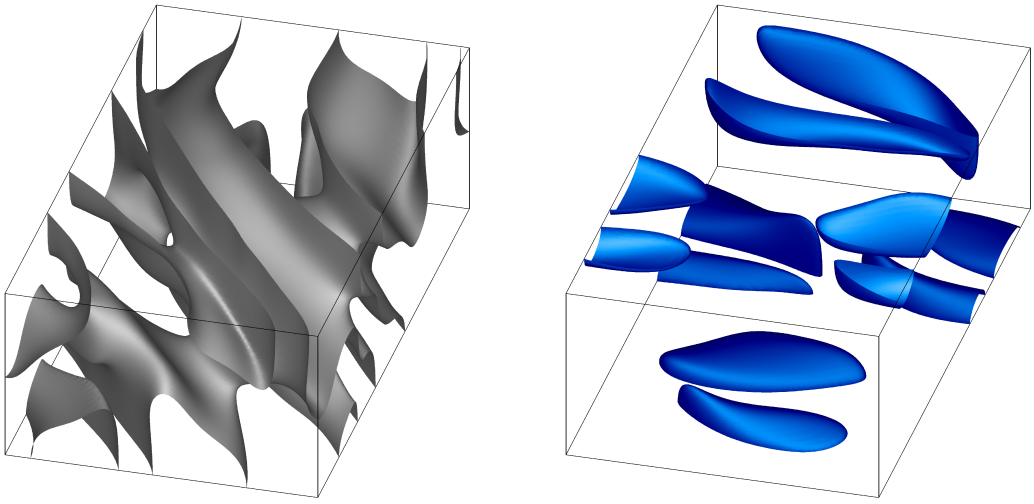


Figure 3.4 Temporal behaviour of the kinetic (left vertical axis, black line) and magnetic (right vertical axis, blue line) energies in the double-period runs 1x2.1 (a), 1x2.2 (c), 1x2.3 (e), 2x1.1 (g), 2x1.2 (i), 2x1.3 (k). The evolution of the magnetic energy (vertical axis) versus kinetic energy (horizontal axis) in the runs 1x2.1 (b), 1x2.2 (d), 1x2.3 (f), 2x1.1 (h), 2x1.2 (j), 2x1.3 (l).



(a) (b)

Figure 3.5 Isosurfaces of the kinetic (a) and magnetic (b) energy densities at the level of 1/2 of the respective maxima in the run 1x2.3 at time $t = 1650.707$. One double-period fluid box is shown.

Fig. 3.4(f)) occurring after a very short transient phase describe the D phase. In the run 1x2.1, the trajectory exponentially approaches a short plateau in $45 < t < 51$ which can be clearly seen upon magnifying the figure in the electronic version of the thesis; it represents a mildly unstable non-generating hydrodynamic steady state. The plot Fig. 3.4(i) for the run 2x1.2 demonstrates that the attractor consists of a cyclic repetition of phases D, G and S.

Plots of isosurfaces of the kinetic Fig. 3.5(a), and magnetic (b) energy densities in the run 1x2.3 at a randomly chosen time $t = 1650.707$ demonstrate that the double-box periodicity of the physical fields do not degenerate into the one-box periodicity of the convective dynamo subjected to perturbation.

The temporal behaviour of the flow at various phases of the regime obtained in the run 2x1.2 is illustrated in Fig. 3.6. The plots show the real parts of the coefficients $\hat{v}^1_{111}$ and $\hat{v}^1_{222}$ on the time intervals, $0 \le t \le 320$ (phase T of the initial transient process, see Fig. 3.4(i)), $800 \le t \le 830$ (phase G, quasiperiodic metastable state), $1520 \le t \le 1620$ (phase S, including the transitions of the previous phase G and the subsequent phase D) and $2000 \le t \le 2200$ (phase D). This behaviour is typical for individual harmonics in the five double-period runs. Temporal power spectra of these coefficients in the phases G, S and D are shown in Fig. 3.7. These plots show that in phase G the magnetic field generation is performed by a flow possessing two basic frequencies: it is a travelling wave (basic frequency $f_{TW} = 0.317$) transporting a periodic orbit (basic frequency $f_{PO} = 1.961$), and in phases S and D the dynamical system is chaotic. The saturated behaviour in phase S has an intricate structure, for instance, the hydrodynamic quasiperiodic metastable state of phase G is revisited on the intervals $1660 \le t \le 1675$ and $1720 \le t \le 1740$.

Thus, the below conclusions assert that the generating flows in phases G in the runs 1x2.3, 2x1.2 and 2x1.3 are the same:

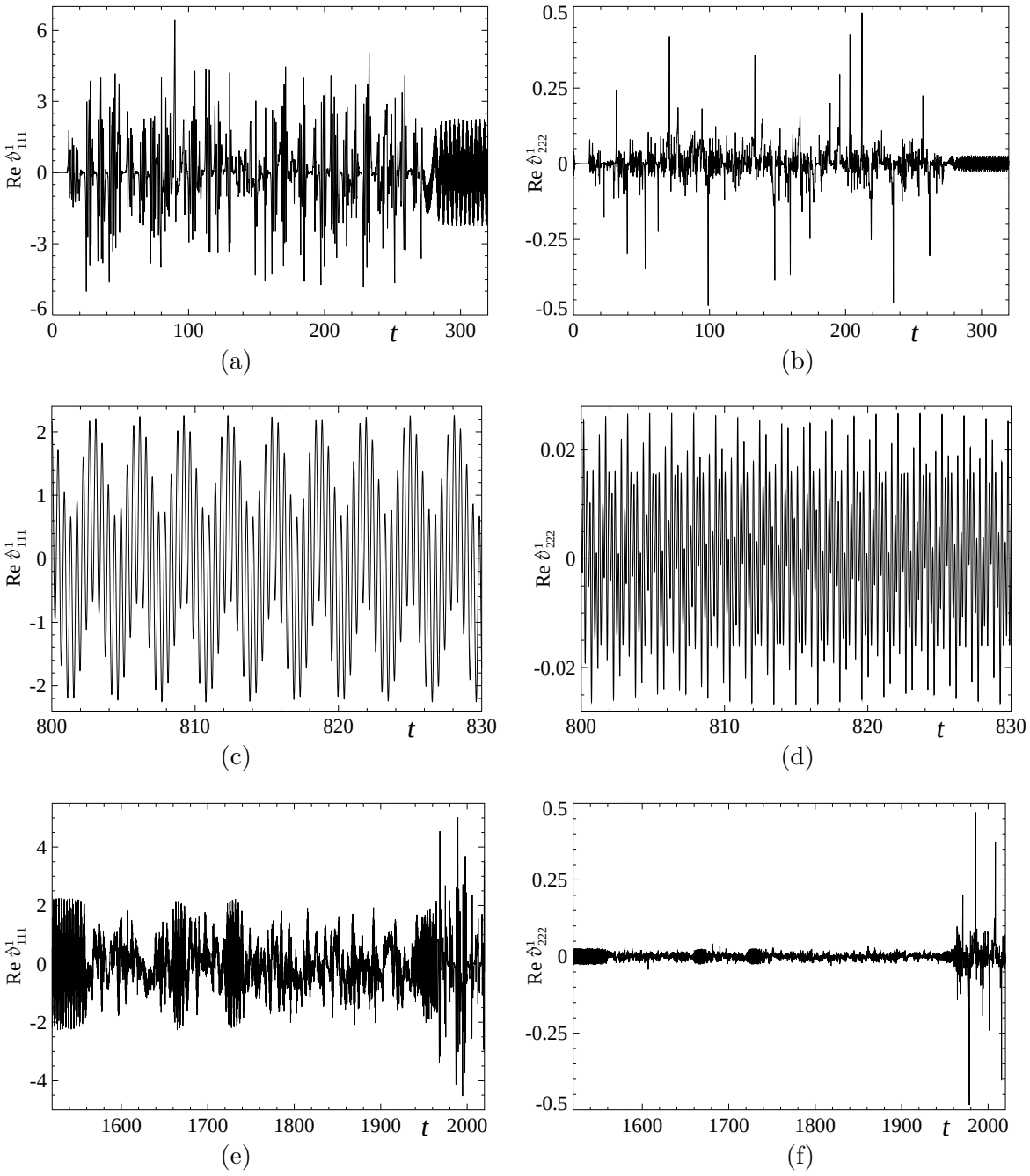


Figure 3.6 The figure is continued at the next page.

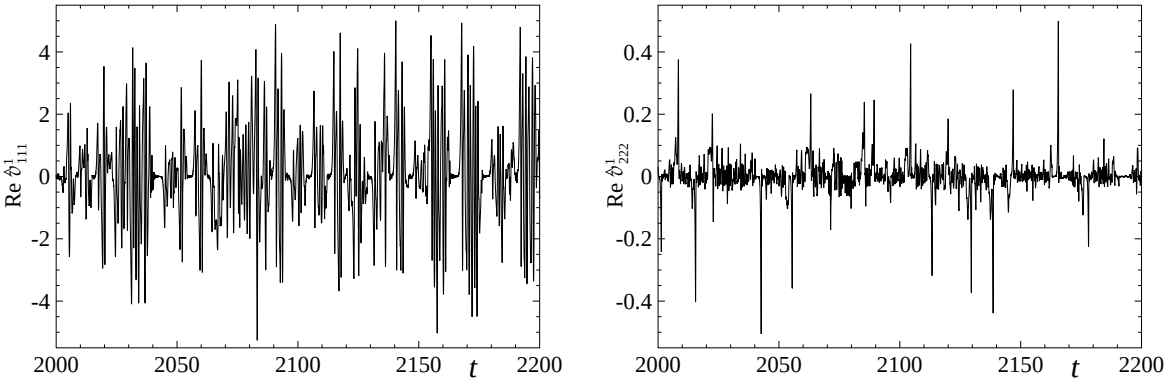


Figure 3.6 Temporal behaviour of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$ (left column) and $\mathrm{Re}\,\hat{v}^1_{222}$ (right column) at three time intervals: $0 \le t \le 320$ (phase T of the initial transient processes, chaotic behaviour and approach of the quasiperiodic metastable state), (a) and (b), $800 \le t \le 830$ (phase G in the vicinity of the quasiperiodic hydrodynamic metastable state), (c) and (d), $1520 \le t \le 2020$ (departure from phase G, phase S of the saturated chaotic regime and approach of the chaotic phase D), (e) and (f), and $2000 \le t \le 2200$ (phase D of magnetic field decay), (g) and (h), in the run 2×1.2 .

- they are quasiperiodic hydrodynamic travelling waves;
- the energy ranges and mean energies over time coincide;
- the basic frequencies are identical;
- the ranges of the magnetic Reynolds number estimates $\mathrm{Rm}_1^{'}$ and $\mathrm{Rm}_3^{'}$ and their mean values over time coincide;
- the ranges of the kinetic helicity and their temporal mean values are the same;
- the growth rates of the dominant magnetic modes are also equal;
- the flows have the symmetry $r\gamma^1_{L/2}$ in the 1×2 cell and $r\gamma^2_{L/2}$ in the 2×1 cell.

In Fig. 3.7, we present the Poincaré maps of the plane ($\mathrm{Re}\,\hat{v}^1_{111}$, $\mathrm{Re}\,\hat{v}^2_{111}$) with the points satisfying the condition $\mathrm{Re}\,\hat{v}^3_{222} = 0$. As it is stated above, the flow is quasiperiodic on the time interval $300 \le t \le 1500$ (phase G), and its points lie on smooth curves (see Fig. 3.7(a)). By contrast, in the chaotic phases S and D of the saturated and decaying magnetic field, the Poincaré map is a cloud of randomly distributed points, which is a signature of chaos. These maps in Fig. 3.7 are also symmetric about the centre of co-ordinates; for a travelling wave, this property does not originate from any spatial symmetry of the flow, but rather from the nature of the conditions used to construct the maps. In Fig. 3.7(b) for phase S, the approximate symmetry about the centre of coordinates can be attributed to the time intervals during which the trajectory moves closer to the generating quasiperiodic flow that we encounter in phase G. The absence of symmetry in Fig. 3.7(c) for phase D suggests that this phase does not have an underlying quasiperiodic flow.

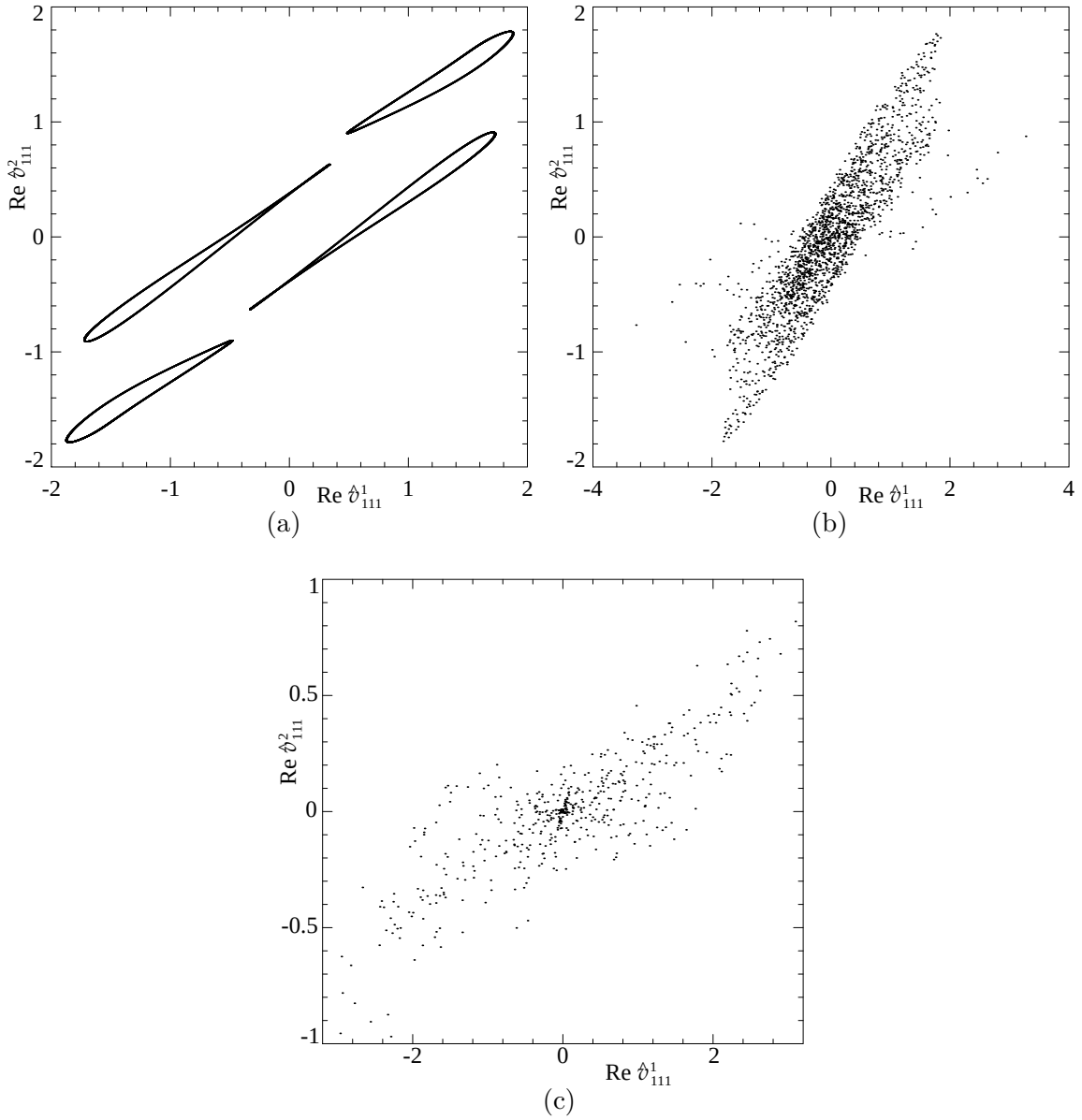


Figure 3.7 Poincaré mapping of the plane $(\mathrm{Re}\,\hat{v}^1_{111},\ \mathrm{Re}\,\hat{v}^2_{111})$ defined by the condition $\mathrm{Re}\,\hat{v}^3_{222} = 0$ on the time intervals $300 \le t \le 1500$ (phase G), (a), $1600 \le t \le 2015$ (phase S and its small neighbourhood), (b), and $2000 \le t \le 2200$ (phase D), (c), in the run 2×1.2 .

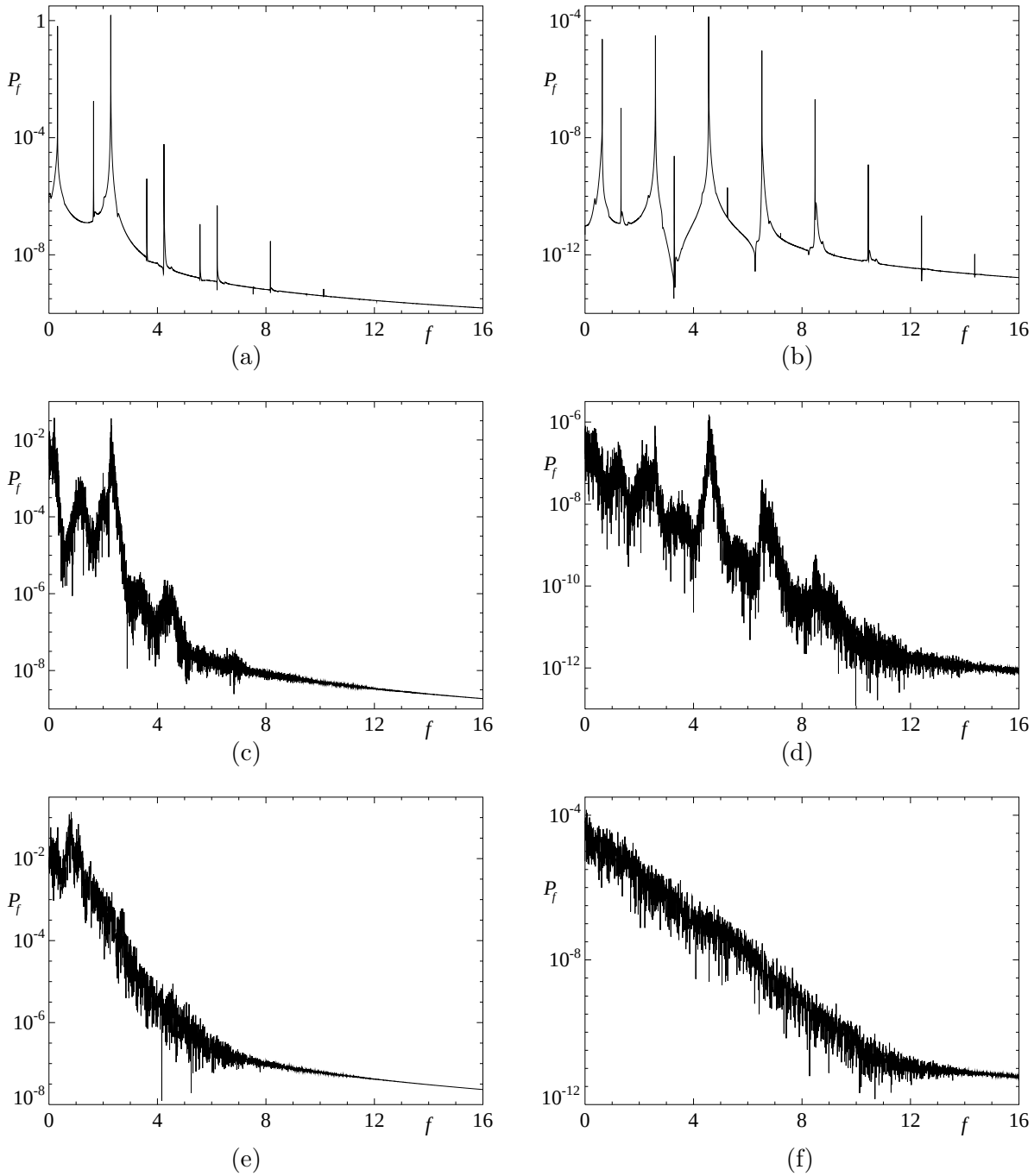


Figure 3.8 Temporal power spectra of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$ (left column) and $\mathrm{Re}\,\hat{v}^1_{222}$ (right column) at the time intervals $285 \le t \le 1555$ (phase G), (a) and (b), $1580 \le t \le 1940$ (phase S), (c) and (d), and $2000 \le t \le 2200$ (phase D), (e) and (f), in the run 2×1.2 .

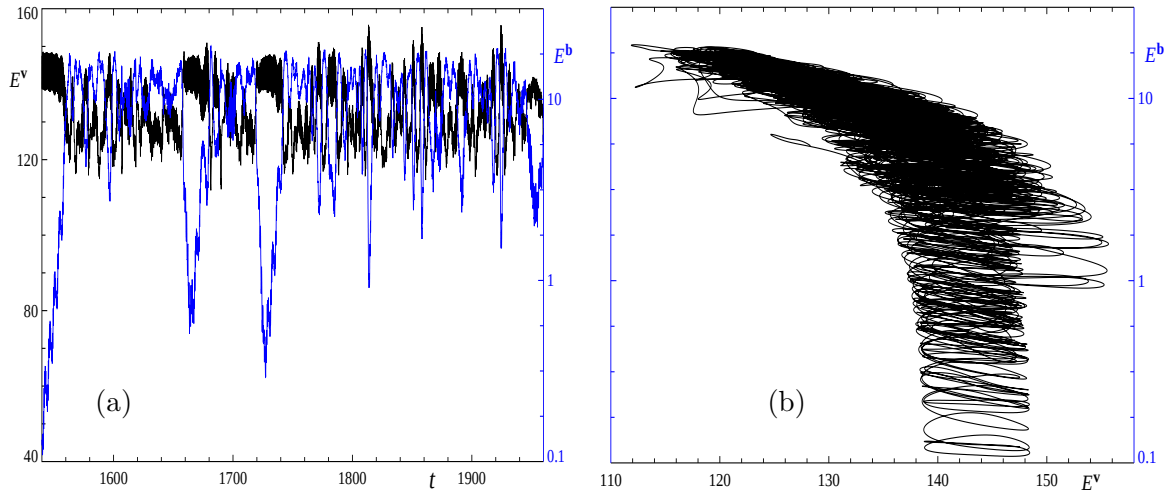


Figure 3.9 Temporal behaviour of the kinetic (left vertical axis, black line) and magnetic (right vertical axis, blue line) energies in the double-period run 2×1.2 for $1540 \le t \le 1960$, phase S (a). The evolution of the magnetic energy versus kinetic energy in this run at the same time interval (b).

The behaviour of the flow and magnetic field energies in Fig. 3.4(a) and (c) for the regimes 1×2.1 and 1×2.2 appear different from that of the regimes 1×2.3, 2×1.2 and 2×1.3. But, a closer examination of phase S in the regime 2×1.2 (see Fig. 3.9) portrays a similar behaviour of the trajectory approaching (once in Fig. 3.9(a) and four times in Fig. 3.9(b)) the same generating hydrodynamic travelling wave, as in phase G in the runs 1×2.3, 2×1.2 and 2×1.3.

Several evidences also validate the fact that the same travelling wave is visited in phase S and acts as the kinematic dynamo in phase G: their energies span the same range, the magnetic growth rates coincide up to small errors of order 10^{-3}, and in the temporal power spectra many peaks of the coefficients Re $\hat{v}^1_{111}$ and Re $\hat{v}^2_{222}$ in phase S are very close to the frequencies of the quasiperiodic flow of phase G (see Fig. 3.8). The difference in the behaviour of the kinetic and magnetic field energies in phases G and S is in a smaller duration of the chaotic evolutions that result in magnetic energy decay by several orders of magnitude, and in a shorter stay in the vicinity of the generating travelling wave; consequently, the same succession of the "mini-phases" G, S, and D occur within the phase S although their alternation is much faster. Several large-amplitude "excursions" seen in Fig. 3.4(c), that are relatively short episodes of the magnetic field decay and its subsequent growth give rise to irregular loops of large-radius around and under the right thin cylinder in Fig. 3.4(d). We conclude that in all the five double-period runs we observe the same attractor.

Fig. 3.10 shows the temporal behaviour of the mean helicity

$$h(t) = \langle (\nabla \times \mathbf{v}) \cdot \mathbf{v} \rangle \,,$$

in three runs for the double-period fluid volume. The mean helicity in phase G is zero. This is clearly seen in Fig. 3.10(b) in the run 2×1.2; in the run 1×2.2, although the phase G time

intervals are short, and the initial perturbations leading up to it are high and at certain times, the energy evolution already shows the advent of this phase, it takes a while for the helicity to decay to significantly smaller values (see Fig. 3.10(a)). The most notable feature in phase G is that the zero mean helicity does not prevent the kinematic magnetic field generation. This is in line with the findings of [Rasskazov *et al.*, 2018], where several examples of generating pointwise zero-helicity three-dimensional steady flows of incompressible electrically conducting fluid were given, and it was concluded that neither kinetic helicity, nor helicity spectrum control the dynamo properties of a flow. Here the presence or the absence of scale separation is not taken into account. In phase S we observe bursts of the mean helicity, whose amplitude is small at first, but gradually increases towards the end of this phase. The amplitude is maximum in phase D, when the flow fails to generate magnetic field.

The attractor in the run 2×1.1 is clearly distinct. The projection of the trajectory in the phase space onto the plane of the kinetic and magnetic energies shown in Fig. 3.4(h) resembles a wing of the Lorenz butterfly attractor. It is unlike the spiral-like structures observed from a similar projection of the attractor emerging in other double-period runs. The mean helicity in this run features intervals of almost periodic behaviour separated by significantly shorter intervals of irregular behaviour (see Fig. 3.10(c)), during which the kinetic energy experiences oscillations of twice higher temporal frequency. The intervals of almost periodic behaviour have opposite signs of the maximum in absolute value of helicity (for instance, cf. its behaviour at the intervals $18 < t < 50$ and $60 < t < 105$). The almost regular helicity behaviour can be expected to set in when the trajectory in the phase space approaches a certain time-periodic metastable state. We find that the 2×1.1 flow is close to a flow which has the following symmetries:

- independence of x_2 (the mean energy of the discrepancy over the periodicity cell, i.e., the mean total energy of the harmonics associated with wave vectors with a non-zero second component is below 0.25 in the saturated regime for $t > 18$; it has a tendency to increase towards the ends of the intervals of almost regular helicity behaviour).
- s_2 (for $t > 18$, the mean energy of the discrepancy, $\mathbf{v}(\mathbf{x}) - s_2\mathbf{v}(\mathbf{x})$ is below 24.0);
- $r\gamma^1_{L/2}$ (for $t > 18$, the mean energy of the discrepancy $\mathbf{v}(\mathbf{x}) - r\gamma^1_{L/2}\mathbf{v}(\mathbf{x})$ is below 14.4);
- the flow is approximately 1×1-periodic (for $t > 18$, the mean energy of the discrepancy, i.e., the total energy (averaged over the periodicity cell) of the harmonics associated with the wave vectors with an odd first component is below 6.1).

This indicates that the underlying metastable state has the above four symmetries and it is 1×1 periodic.

Three 1×1-periodic regimes of hydromagnetic convection were identified in [Chertovskih *et al.*, 2010] for the same set of parameter values considered in this study: R_1 (steady amagnetic rolls, parallel to a side of the square periodicity cell) and stable to 1×1-periodic MHD perturbations, $\mathrm{S}_7^{\mathrm{R}1}$ and $\mathrm{S}_8^{\mathrm{R}1}$(MHD steady states, stable to 1×1-periodic MHD perturbations; $\mathrm{S}_8^{\mathrm{R}1}$ is the state subjected to perturbations in the present study). Among these, only R_1 matches the flow in the regime 2×1.1 in kinetic energy, and has the four symmetries that hold

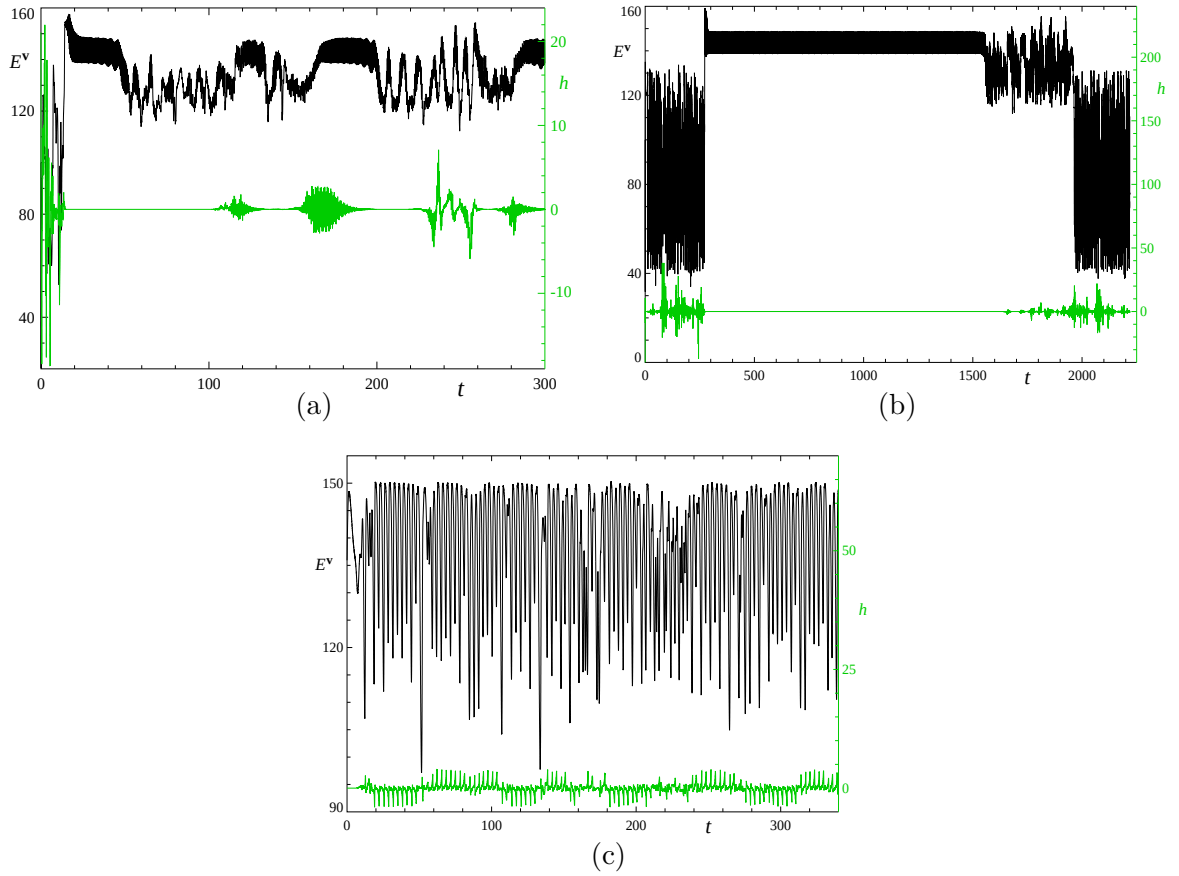


Figure 3.10 Temporal behaviour of the mean helicity (green line, right vertical axis) with the kinetic energy (black line, left vertical axis) in the runs 1×2.2 (a), 2×1.2 (b) and 2×1.1 (c).

approximately for 2×1.1. We therefore conclude that R_1 is the metastable state approached by the run 2×1.1. The following evidences also corroborate this:

- We performed a hydrodynamic run by setting the magnetic field to zero at each step in a 2×1-periodic fluid box for an initial condition where the magnetic energy is close to a local minimum in the run 2×1.1 (we chose the state at t=180). The x_2-dependence is suppressed by setting all harmonics with a non-zero second component to zero. This run converged in about 15 time units to the double replicated R_1 with the energy of discrepancy decreasing below 10^{-18}.

- The energy of discrepancies between the 2×1.1 flow at $t = 180$ and the flows of the MHD steady states R_1, S_7^{R1} and S_8^{R1} is 0.2, 0.2 and 16.1, respectively.

The rolls R_1 do not depend on x_2 and have the symmetries s_2 (rotation about the vertical axis by π) and $r\gamma^1_{L/2}$ (reflection about the midplane with a shift in x_1 in half a period). The symmetry $r\gamma^1_{L/2}s_2$ amounts to parity invariance (with an appropriate shift of the origin in x_1), which implies that the mean helicity of R_1 vanishes. Thus the flow in the 2×1.1 regime can be regarded as a perturbation of the parity-invariant metastable state R_1, and its total helicity

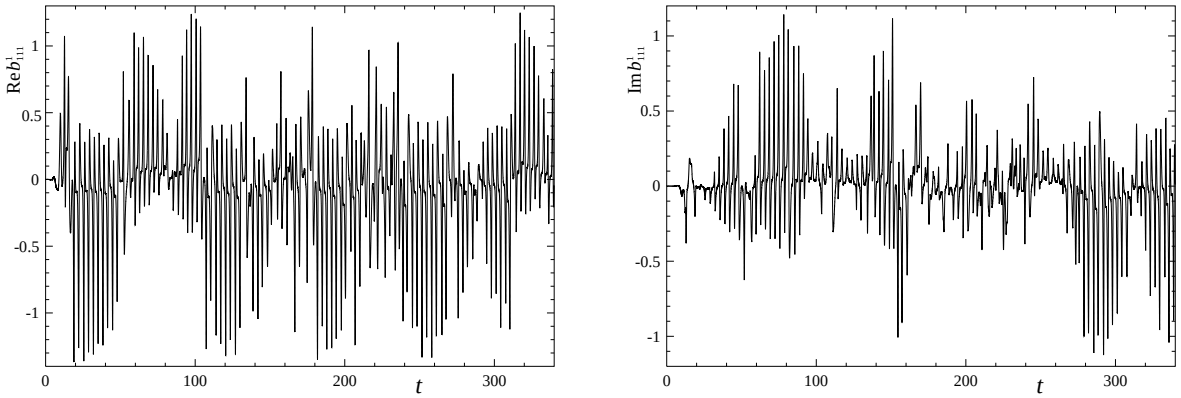


Figure 3.11 Temporal behaviour of the Fourier coefficients $\mathrm{Re}\,\hat{b}^1_{111}$, (a), and $\mathrm{Im}\,\hat{b}^1_{222}$ (b) in the run 2×1.1 .

is predominantly linear in the parity-antiinvariant component of the perturbation. Since the rolls R_1 are stable with respect to hydrodynamic perturbations, their perturbation giving rise to the 2×1.1 flow is the consequence of the action of the Lorentz force from the growing magnetic field. We note that the Lorentz force has the required parity-antiinvariant component only if the magnetic field has both non-zero parity-invariant and parity-antiinvariant components. Existence of two dominant magnetic modes, parity-invariant and parity-antiinvariant, with equal growth rates, is linked with the independence of R_1 on x_2: the two eigenmodes are of the form $e^{\pm ix_2}\mathbf{m}(x_1, x_3)$, their real and imaginary parts feature opposite parity invariance properties. In the course of further evolution the magnetic field grows till its magnitude becomes so large, that the Lorentz force starts affecting the flow significantly until the modified flow fails to sustain the magnetic field. At this moment the generation ceases and the flow converges back to R_1. This cycle of magnetic field generation and subsequent decay repeats although with some alterations. This is because the initial noise fails to decay in a cycle and eventually affects the next cycle. When the perturbation accumulates to significant levels, the periodic repetition of the cycles is broken and a chaotic "interlude" follows up to the onset of the next sequence of the cycles. We observe this as a series of repeated perturbed "teeth" in the helicity plot.

Exhaustive arguments explaining the almost regular behaviour of the kinetic helicity at certain time intervals and a similar behaviour with the helicity reversed at other time intervals can be obtained by constructing the appropriate asymptotic solutions for the weakly nonlinear regime 2×1.1. We plan to present these arguments in the near future. The presence of intervals of irregular behaviour of the helicity characterised by frequency doubling of the energy suggests that another metastable state of a twice shorter period exists in the vicinity of the attractor.

Temporal behaviour of the real part of the Fourier coefficient $\hat{b}^1_{111}$ (see Fig. 3.11(a)) can be interpreted as a kind of magnetic reversals behaviour: while for $18 \le t \le 50$ $\mathrm{Re}\,\hat{b}^1_{111}$ is predominantly negative, for $60 \le t \le 105$ it is positive; similar intervals of dominant signs

of the coefficient can be identified. The behaviour of the imaginary part (Fig. 3.11(b)) can be interpreted as a kind of magnetic reversals behaviour: it shows a similar intermittent behaviour; its intervals of dominant signs loosely correlate with those of the positive part.

3.2.2 Regimes with the periodicites 1×3/3×1

The temporal behaviour of the kinetic and magnetic energies observed in the triple-period fluid cells (1×3/3×1) is shown in Fig. 3.12: runs include 3×1.1 from the standard resolution and 3×1.2 from the standard and alternative resolution. All runs possess an initial chaotic transition phase and it varies roughly between 25 and 1250 time units. Subsequently, a regime with an underlying non-generating quasiperiodic flow sets in. This flow is a travelling wave with a periodic orbit. All runs from both resolutions reach this terminal non-generating travelling wave. Unlike in the double-period runs, the instability here fails to transform this flow into a generating flow. From the standard-resolution, 1×3.3 was integrated for almost 2000 time units, with its travelling wave persisting from $t = 85$ up to $t = 1992$. We performed this check to see if the magnetic field generation can be reinstated. Therefore we conclude that for convection in the triple-period elongated cells the attractor is unique and it is hydrodynamic.

In Fig. 3.12, we present the temporal evolution of the energies of the run 3×1.2 for the standard (128 x 384 x 97 harmonics) and alternative (256 x 256 x 257) space resolution. Their evolutions fail to coincide because of the inherent nature of instability in these regimes. But they are qualitatively similar with some minor differences. For instance, there is a relatively long oscillatory decrease of kinetic energy in the time interval $52 < t < 80$ witnessed in the run for the alternative resolution.

Inspecting the projection of the trajectories on the plane of magnetic and kinetic energies we notice spiral-columnar-like structures on the right of Fig. 3.12(b), (d). There is a similarity with the emergence of this terminal phase to the spiral loops observed in the double-period fluid volumes. Howeve, here the flow is quasiperiodic with an exponential decay of the magnetic field. The spiral loops are descending unlike in the double-period runs.

Isosurfaces of energy densities from the run 3×1.2 (Fig. 3.13) at two randonly chosen times show the distinction of structures observed in the two phases: figures (a) and (b) represent the initial chaotic transitory phase which extends up to $t = 550$ and figures (c) and (d) represent the terminal phase which roughly begins at $t = 565$. In the terminal phase the magnetic field has already started to decay and it is yet to acquire the structure of the least decay magnetic mode. Nonetheless the kinetic and magnetic energy distribution is ordered. The blobs of magnetic field which take more volume in the initial phase shrink considerably. The distribution of flow energy in the quasiperiodic hydrodynamic state is visibly more uniform in space. These plots also affirm that in the course of the temporal evolution the fields do not acquire the periods of the one-box solution.

The energy isosurfaces in Fig. 3.13 and isosurfaces of the individual components of the flow in Fig. 3.14 show that neither at the initial chaotic transitory phase nor in the terminal quasiperiodic state, the flow acquires a two-dimensional structure that would restrain the

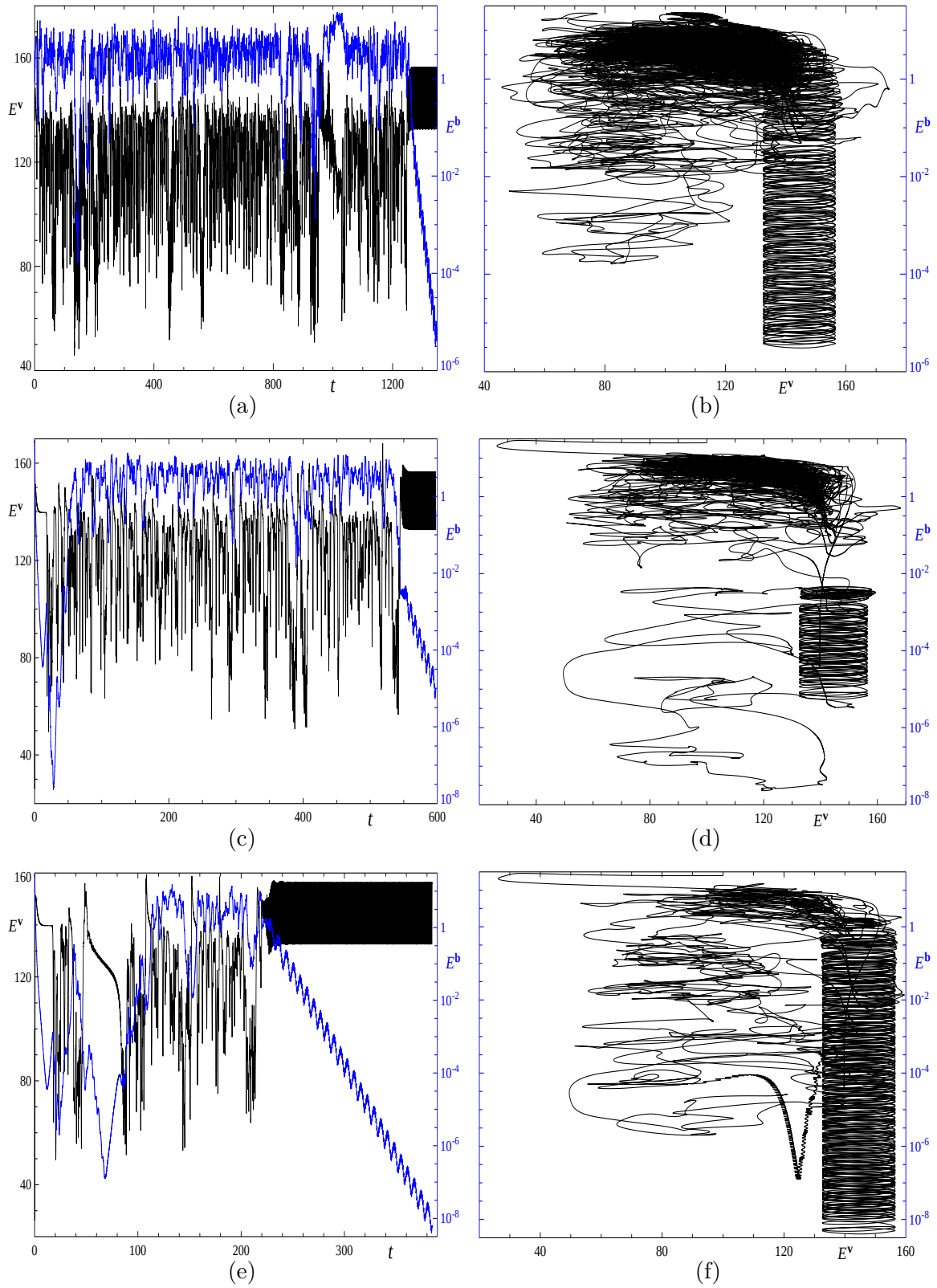


Figure 3.12 Temporal behaviour of the kinetic (left vertical axis, black line) and magnetic (right vertical axis, blue line) energies in triple-period runs (left column) and projection of the trajectories on the plane of magnetic and kinetic energies (right column). Runs 3x1.1 (a), (b) and 3x1.2 (c), (d), the standard space resolution ($128 \times 384 \times 97$ harmonics); run 3x1.2, (e), (f), the alternative resolution ($256 \times 256 \times 257$ harmonics).

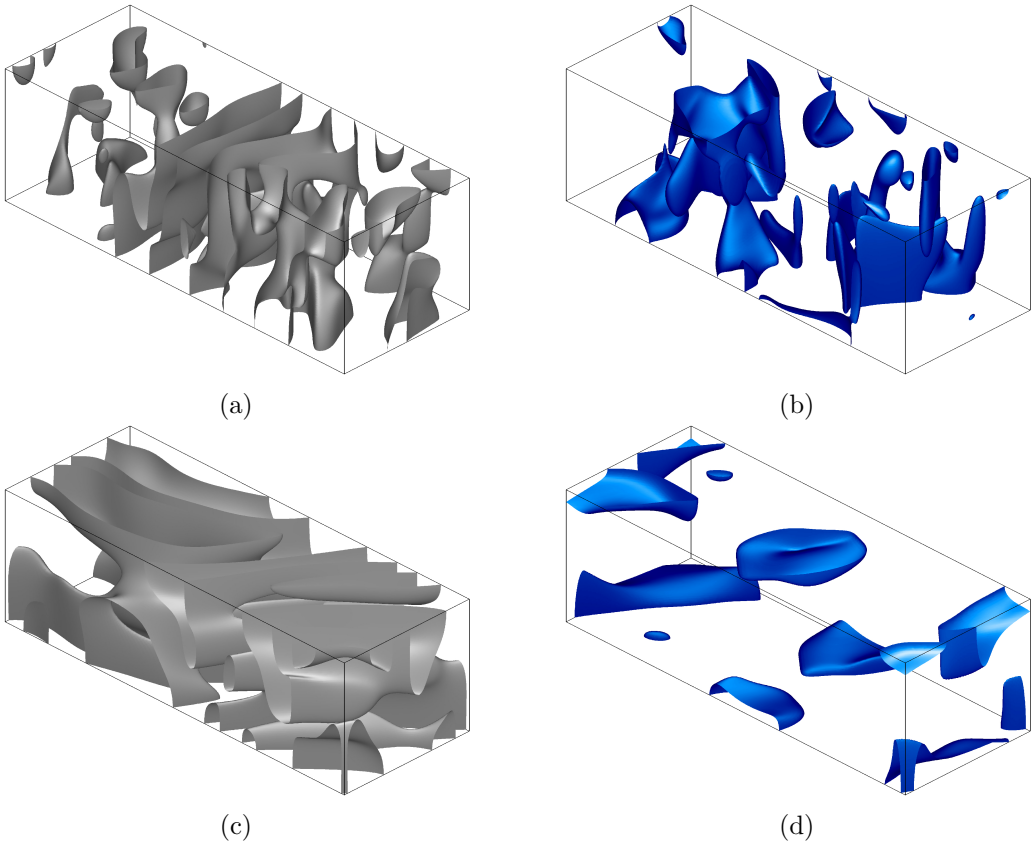


Figure 3.13 Isosurfaces of the kinetic (a), (c) and magnetic (b), (d) energy densities in the run 3×1.2 (the standard resolution) at times $t = 380.671$ (a), (b) and $t = 566.671$ (c), (d) at the levels of 15% (d), 30% (b) and 50% (a), (c) of the respective maxima of the densities. One triple-period fluid box is shown.

generation of magnetic field - as stated by the Zeldovich antidynamo theorem [Zeldovich, 1957].

The plots Fig. 3.15(a) of the magnetic Reynolds number estimates for the run 3×1.2 show that upon the advent of the quasiperiodic flow in the terminal phase, $\mathrm{Rm}_1^{'}$ and $\mathrm{Rm}_3^{'}$ on average increase. The mean $\mathrm{Rm}_3^{'}$ increases from roughly 30.5 in the initial chaotic phase (for $t < 533$) to 35.6 in the terminal phase. The mean values of $\mathrm{Rm}_1^{'}$ and $\mathrm{Rm}_3^{'}$ computed over time differ insignificantly in the terminal phase; their respective values are 32.6 and 35.7 in the double-period runs, and 31.7 and 35.7 in the triple period runs.

The kinetic energy on average is smaller in the initial chaotic phase than in the terminal phase. The travelling waves in the double-period and triple-period cells are almost identical in vigour: kinetic energy oscillates in the range $138.9 < E^v < 148.5$ with its mean around 143.6 in the double-period runs, and in the range $132.6 < E^v < 156.5$ around its mean 142.9 in the triple-period runs.

In the initial transitory chaotic regime, the mean kinetic helicity varies in the range $-18.7 < h < 22$. But, in the terminal phase it vanishes as in the generating phases G of the five runs of the double-period fluid cells.

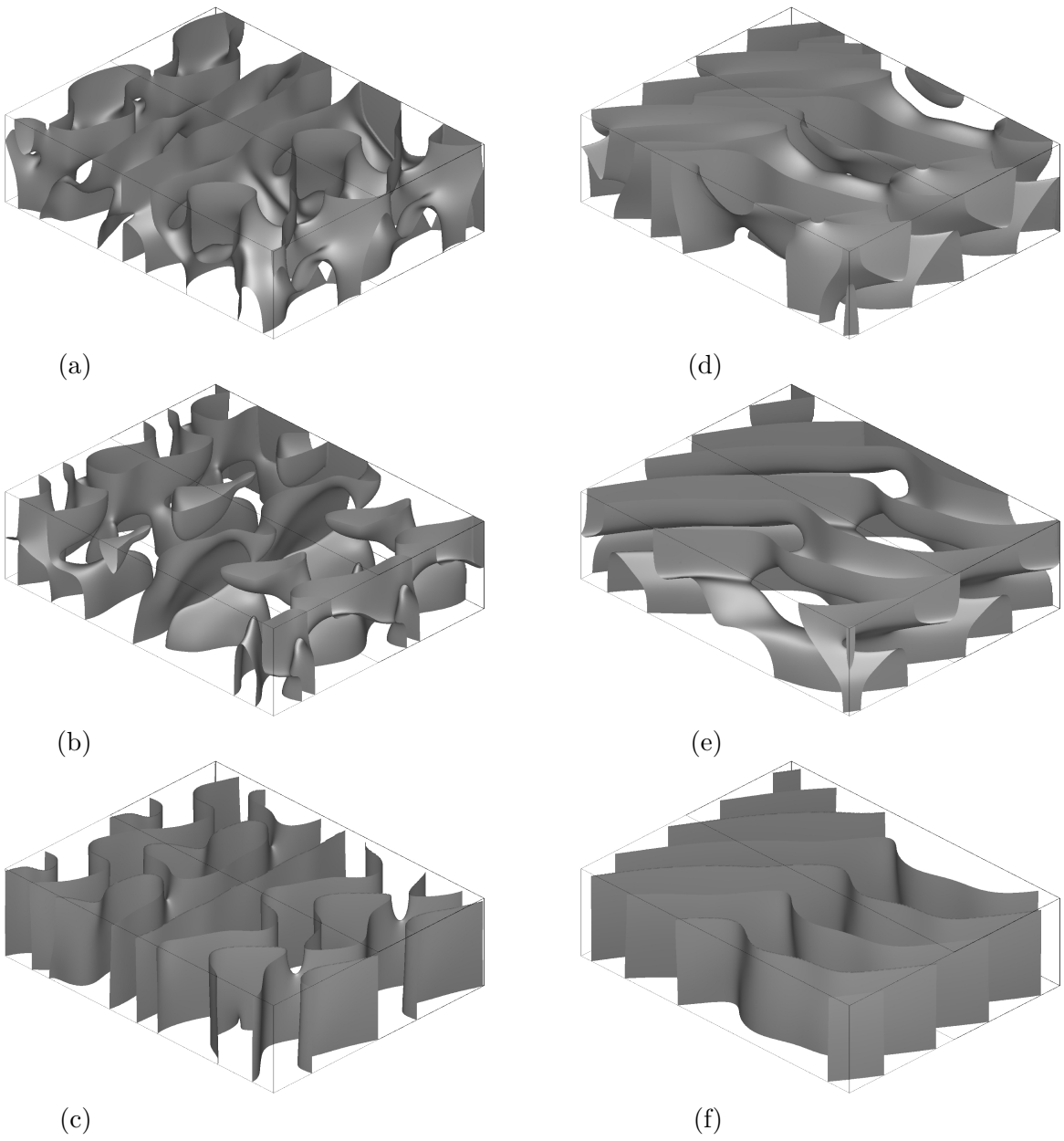


Figure 3.14 Isosurfaces of the zero-value flow components v^1 (a), (d), v^2 (b), (e), v^3 (c), (f) in the run 3x1.2 (the standard resolution) at times $t = 380.671$ (a)–(c) and $t = 566.671$ (d)–(f). Two adjacent triple-period fluid boxes are shown. Although, by the boundary conditions (3.5), $v^3 = 0$ on the horizontal boundaries, they are not shown as zero-value isosurfaces in (c) and (f), because that would block the view of the isosurfaces inside the fluid volume.

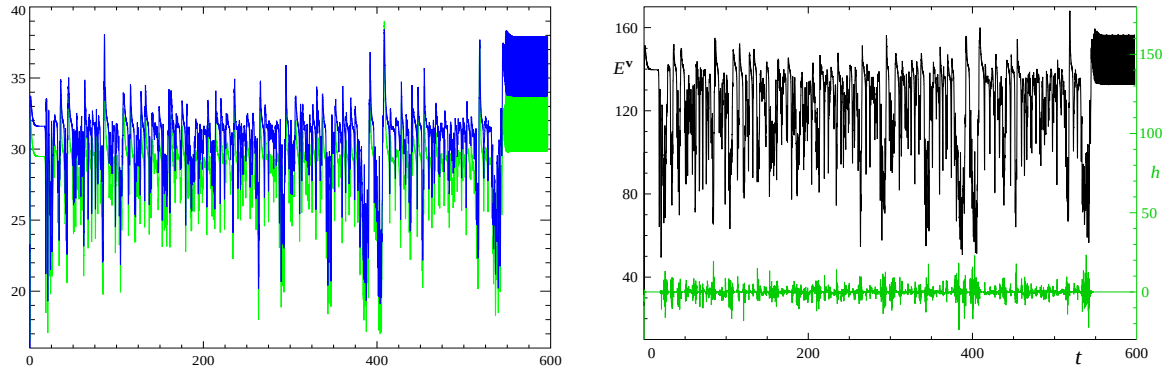


Figure 3.15 Temporal behaviour of the local in time estimates of the magnetic Reynolds number, Rm'_1 (green line) and Rm'_3 (blue line), (a), and of the kinetic helicity (green line, right vertical axis) together with the kinetic energy (black line, left vertical axis) for reference, (b), in the run 3x1.2 .

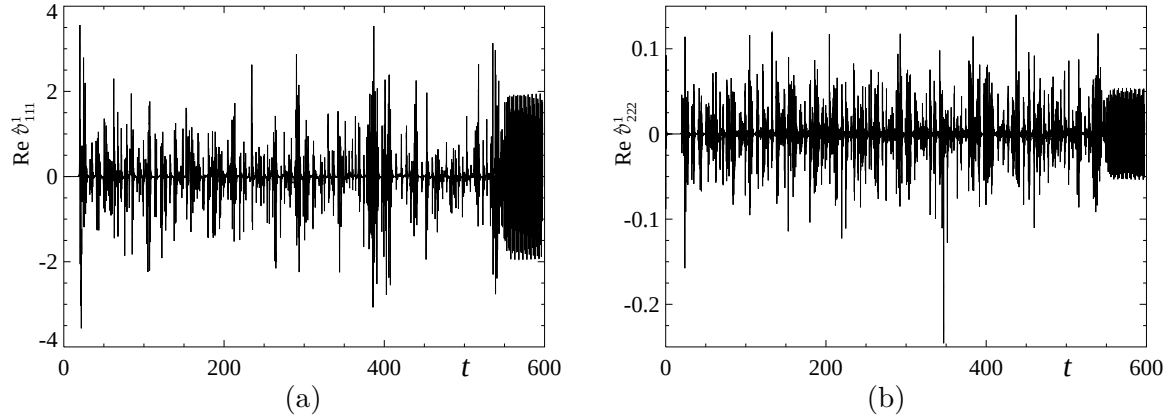


Figure 3.16 Temporal behaviour of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$ (a), and $\mathrm{Re}\,\hat{v}^1_{222}$ (b), for $0 \le t \le 600$, in the run 3x1.2 .

Despite all this analysis and small differences in values of the integral parameters the flow in the triple-period cells is non-generating. Indications are that favourable values of kinetic energy, mean kinetic helicity or magnetic Reynolds number do not guarantee that a flow can act as a dynamo and generation of magnetic field depends on the intrinsic small-scale details of a flow's structure.

The temporal behaviour of the coefficients Re $\hat{v}^1_{111}$ and Re $\hat{v}^1_{222}$ shown in Fig. 3.16 for the interval $0 \le t \le 600$ from the run 3x1.2 represents the typical behaviour of individual harmonics in the triple-period runs.

Plots of the temporal power spectra Fig. 3.17(a), (b) of the two coefficients for the same run on the interval $50 \le t \le 533$ validates that the initial behaviour is chaotic. For the subsequent phase where the flow is quasiperiodic and where the magnetic field decays exponentially, we have used data of two harmonics on the interval $75 \le t \le 975$ from the run 1x3.3 (see Fig. 3.17(c),(d)). The two adjacent peaks in the spectra plots indicate that the

flow is a travelling wave transporting a periodic orbit with two basic frequencies $f_{TW} = 0.398$ and $f_{PO} = 1.391$ – as in the five double-period fluid volumes.

We performed the same quantitative and qualitative tests that were used to establish the coincidence of the generating flows in phases G of the runs 1×2.3, 2×1.2 and 2×1.3 (see section 3.2.1) with an exception: the magnetic field decay rates were computed for the triple-period cells as to the growth rates computed for the double-period cells. The tests confirm that the flows in the terminal phase in the triple-period fluid cells are identical. Similar results were also obtained in the terminal phase in the runs for the alternative resolution.

Poincaré sections for the flows in the runs 1×3.1 and 3×1.3 are shown in Fig. 3.18. They also confirm that the flows in the final phase are travelling waves transporting periodic orbits. As it can be noticed the shape of the sections is significantly different, because the relative positions of the periodicity cells are to be adjusted in order to obtain identical maps. This task was not implemented.

3.2.3 Regimes with the periodicities 1×4/4×1

The temporal behaviour of the mean kinetic and magnetic energies in the quadruple-period runs are shown in Fig. 3.19: run, 4×1.2 for the standard, and 4×1.1 for the standard and alternative resolution. They represent the typical behaviour of the convective dynamos in the quadruple-fluid cells. The evolution of the flows is again chaotic. But it is more structured in the run 4×1.1(Fig. 3.19(c),(d)) than in 4×1.2 (Fig. 3.19(a),(b)) and it can be divided into two phases:

- a relatively vigorous flow generating a relatively large magnetic field in time intervals $15 < t < 120$ and $t > 340$.
- a moderate flow in the interval $132 < t < 212$, which fails to generate magnetic field of the same magnitude as the previous phase.

In the first phase, the kinetic energy E^v experiences relatively low-amplitude chaotic oscillations between 105.0 and 166.0 around the mean 140.5, and the magnetic energy E^b experiences oscillations between 0.7 and 14.7 around the mean 5.9. In the second phase E^v, the oscillations are around the mean 112.0 and its amplitudes are in the range $49.9 < E^v < 203.8$. Here the magnetic field has become weaker, $E^b < 0.17$ with its mean around 0.01. The high density of segments of smooth curves of the trajectory in the upper right part of the (E^v, E^b) plane occurs during the second phase of the evolution. This phase is further split into two parts - the trajectory approaches a hydrodynamic metastable state for a short time in the interval $223 < t < 242$. Here, the kinetic energy varies in the range $131.5 \leq E^v \leq 152.3$ with the largest mean, 142.9. This flow is a generating travelling wave. Its projection is a spiral-like structure in the energy plot of the (E^v, E^b) plane. A similar categorisation of phases cannot be made for the run 4×1.2 (see Fig. 3.19(e),(f)), 1×4.1 and 1×4.3, where unstructured chaotic evolutions are seen. Such a behaviour is also observed in the alternative resolution of the quadruple-period runs.

Clearly, the qualitative differences mentioned above and the behaviour observed in the runs 4×1.2, 1×4.1 and 1×4.3 fail to indicate whether the quadruple-period regimes have two

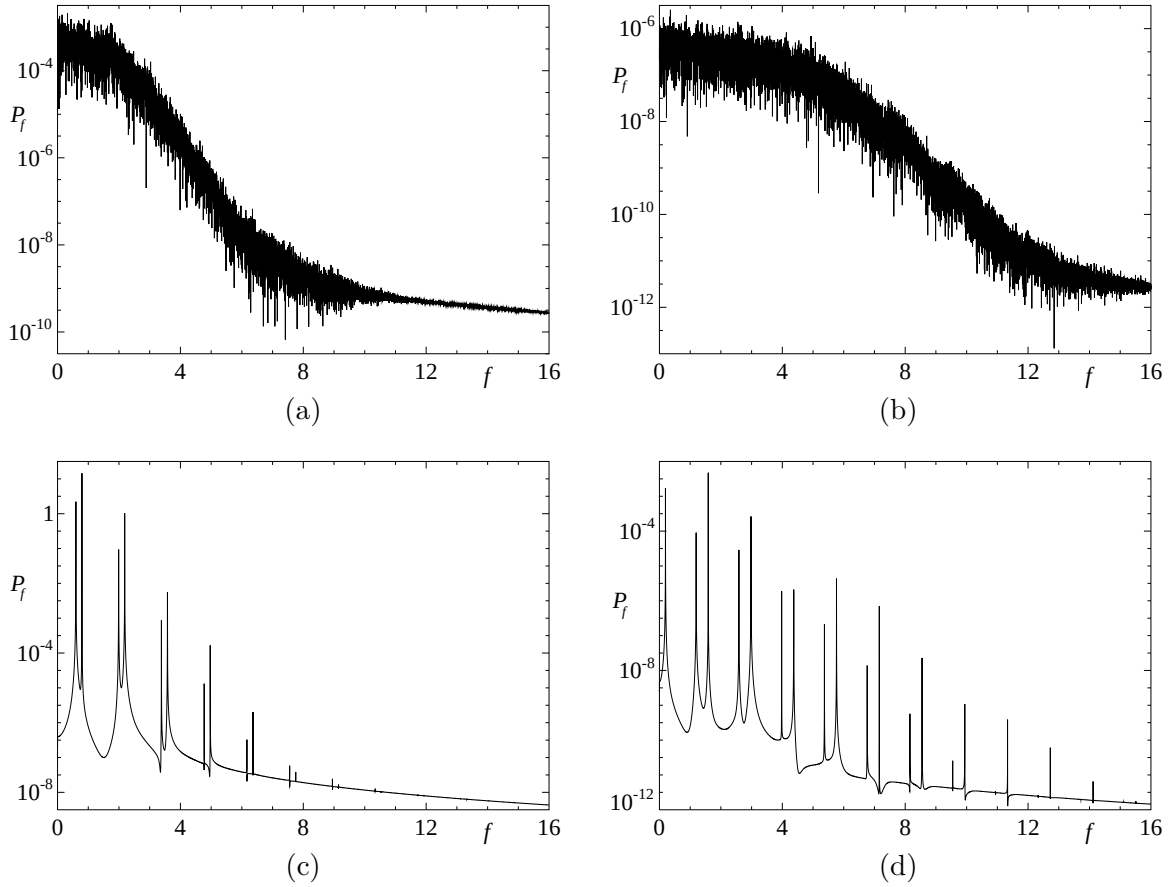


Figure 3.17 Temporal power spectra of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$ (left column) and $\mathrm{Re}\,\hat{v}^1_{222}$ (right column) at the interval $50 \le t \le 533$ in the run 3x1.2 (a), (b) and at the interval $75 \le t \le 975$ in the run 3x1.2 .

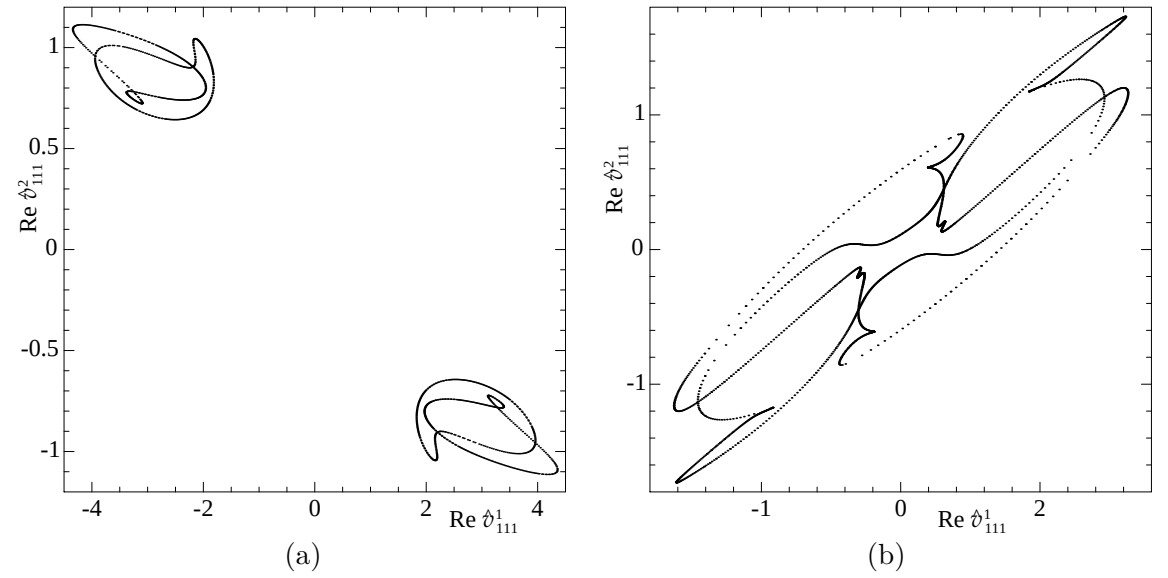


Figure 3.18 Poincaré sections of the plane ($\mathrm{Re}\,\hat{v}^1_{111}$, $\mathrm{Re}\,\hat{v}^2_{111}$) defined by the condition $\mathrm{Re}\,\hat{v}^3_{222} = 0$ for the hydrodynamic attractors in the runs 1x3.1 (constructed for the time interval $100 \le t \le 487$), (a), and 3x1.3 (for $300 \le t \le 550$), (b).

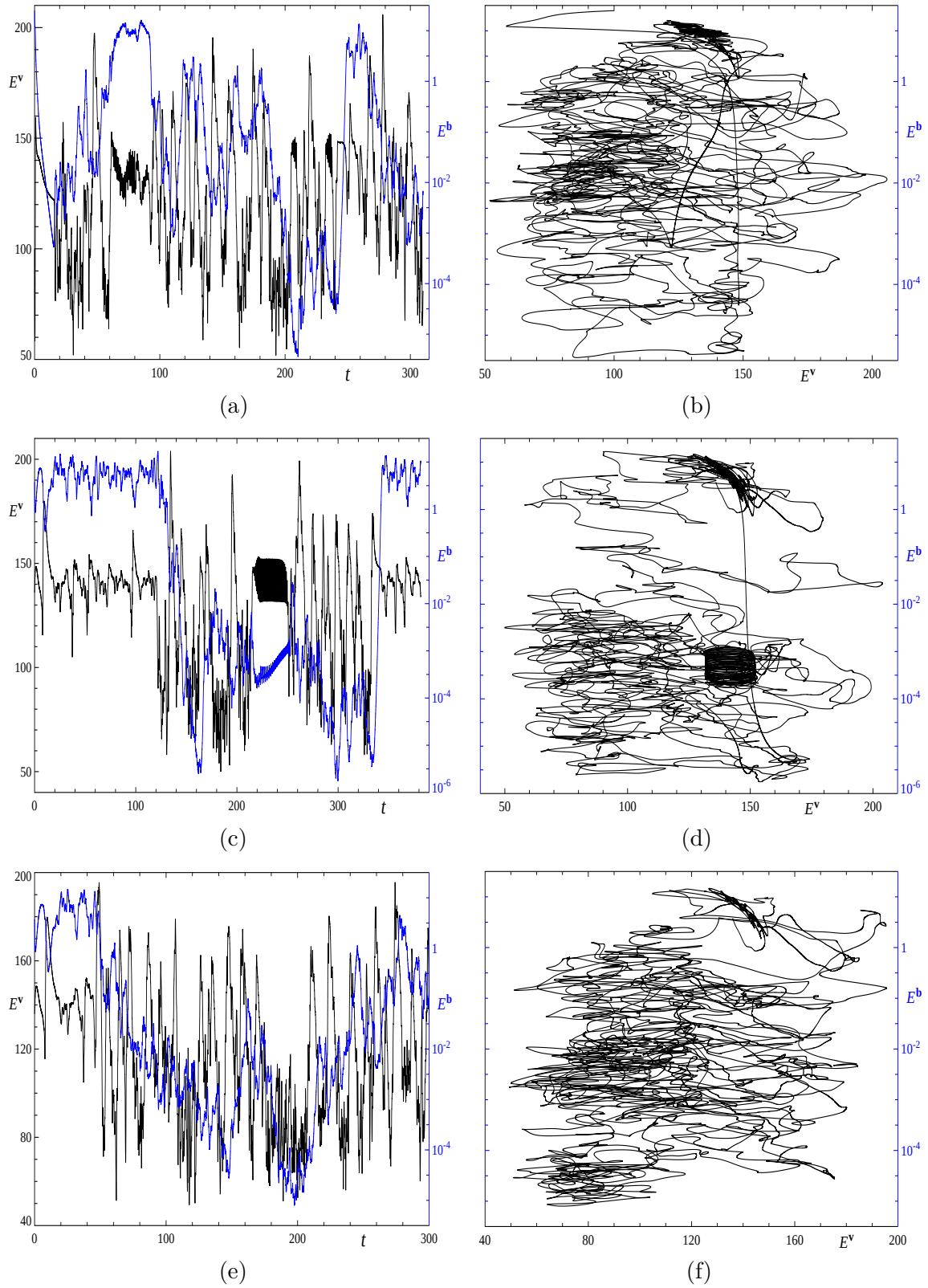


Figure 3.19 Temporal behaviour of the kinetic (left vertical axis, black line) and magnetic (right vertical axis, blue line) energies (left column), and projection of the trajectories on the plane of magnetic and kinetic energies (right column) in the quadruple-period runs 4x1.2 for the standard resolution, (a), (b), 4x1.1 for the standard resolution, (c), (d), and 4x1.1 for the alternative resolution, (e), (f).

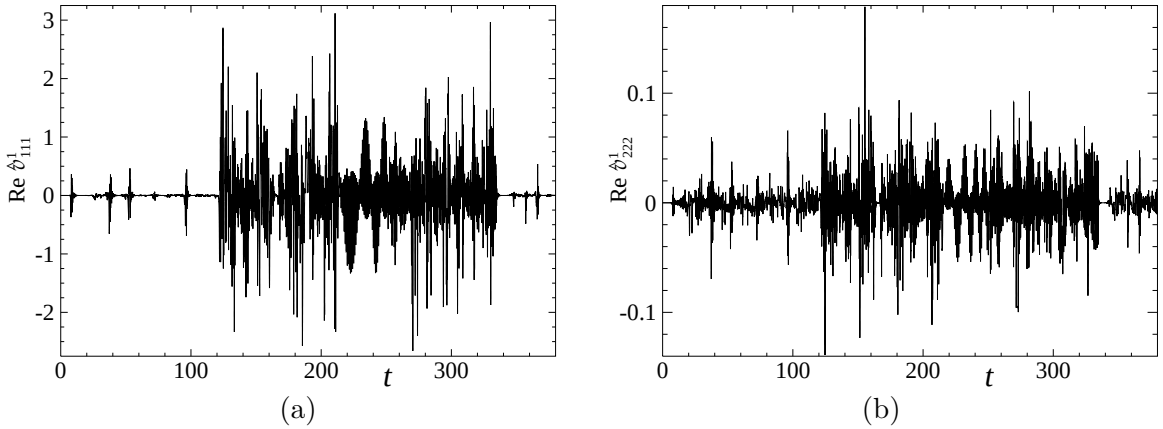


Figure 3.20 Temporal behaviour of the coefficients Re $\hat{v}^1_{111}$ (a), and Re $\hat{v}^1_{222}$ (b), in the run 4x1.1 .

or more attractors. The unique emerging attractor is chaotic and non-hydrodynamic. The magnetic field is sustained for the entire duration of the runs, which do not exceed 400 time units. The triple-period runs showed that the transitory regimes may last twice longer. So, continuing the quadruple-period runs may reveal that the computed evolutions are just transients and may reach a different attractor. A decay of the magnetic field in the terminal phase is also plausible.

A similar evolution of distinct phases is observed in the behaviour of individual harmonics (see Fig. 3.20), the magnetic Reynolds number estimates, $\mathrm{Rm}_1^{'}$ and $\mathrm{Rm}_3^{'}$ (Fig. 3.22 (a)), and the mean kinetic helicity (Fig. 3.22 (b)).

In the first phase the flow Fourier coefficient Re $\hat{v}^1_{111}$ is predominantly zero with some random perturbations; each dies out in fast exponentially decaying oscillations. In fact, the perturbations of a chaotic metastable state possess the symmetry $r\gamma^1_{2L}$, and it is not preserved during the perturbative events.

Temporal spectra of the coefficients Re $\hat{v}^1_{111}$ and Re $\hat{v}^1_{222}$ of the two phases are presented in Fig. 3.21. In the first phase the spectra can be distinguished as curves with multiple arc segments that are concave up and down and subjected to a white-noise-like perturbation of considerable amplitude. Such an intrinsic double-scale structure of the spectra is not typical for chaotic regimes in our simulations. Temporal spectra of the oscillatory regime splitting the second phase indicate that the flow is a travelling wave (as we have established $|\hat{v}^1_{111}|$ and $|\hat{v}^1_{222}|$ are time periodic functions) transporting a periodic orbit; the basic frequencies are $f_{TW} = 0.947$ and $f_{PO} = 1.816$, respectively.

The behaviour of the magnetic Reynolds number estimates is also consistent with the observed efficiency of magnetic field generation (see Fig. 3.22): in the first phase the estimates $\mathrm{Rm}_1^{'}$ and $\mathrm{Rm}_3^{'}$ are on average higher than in the second phase. Although momentary peaks are significantly higher in the second phase, the magnetic field generation is weak. The estimates for the intermediate quasiperiodic flow are higher than those encountered in the first phase, and indeed, this flow is kinematically generating. But the ranges of $\mathrm{Rm}_1^{'}$ and

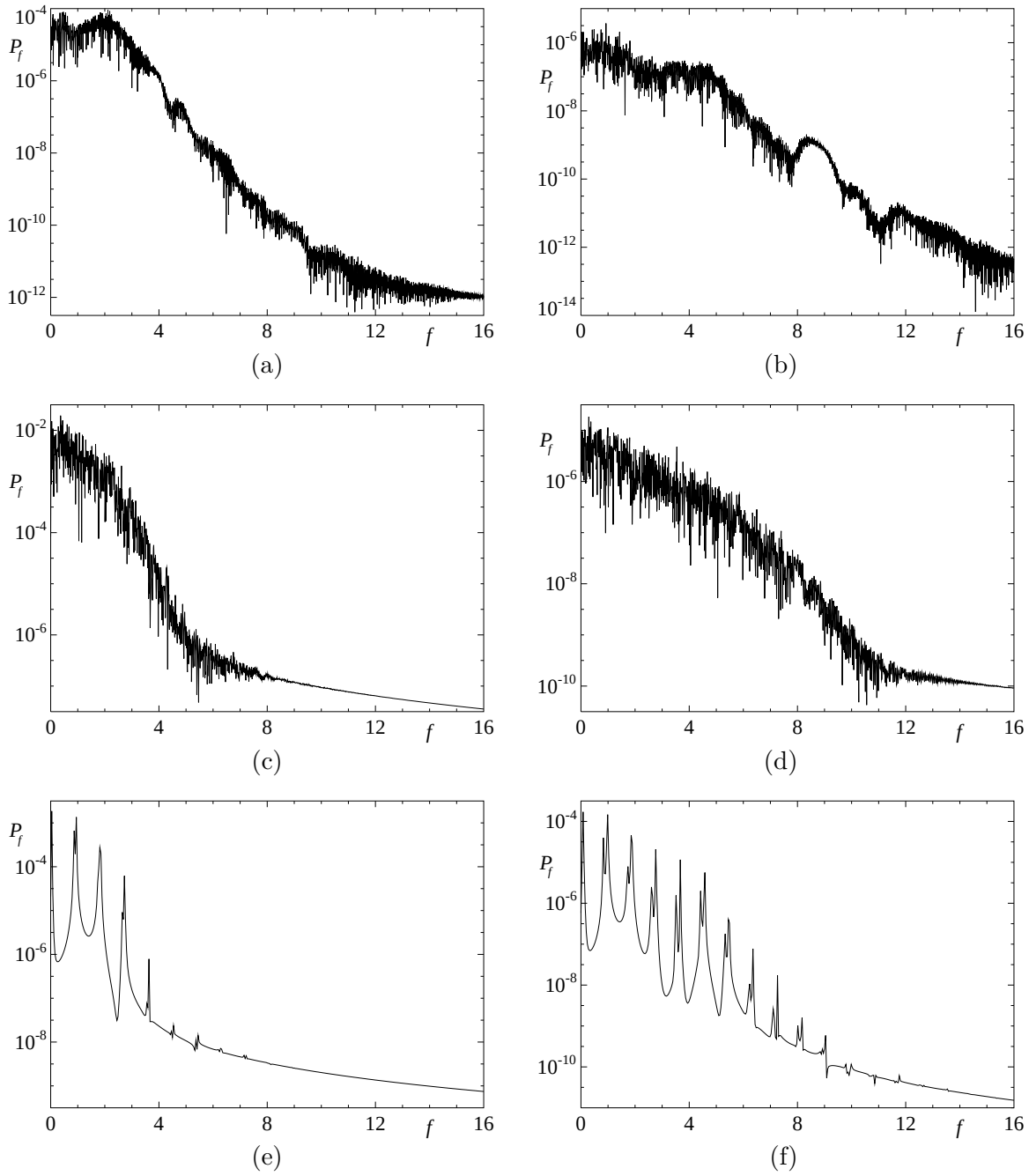
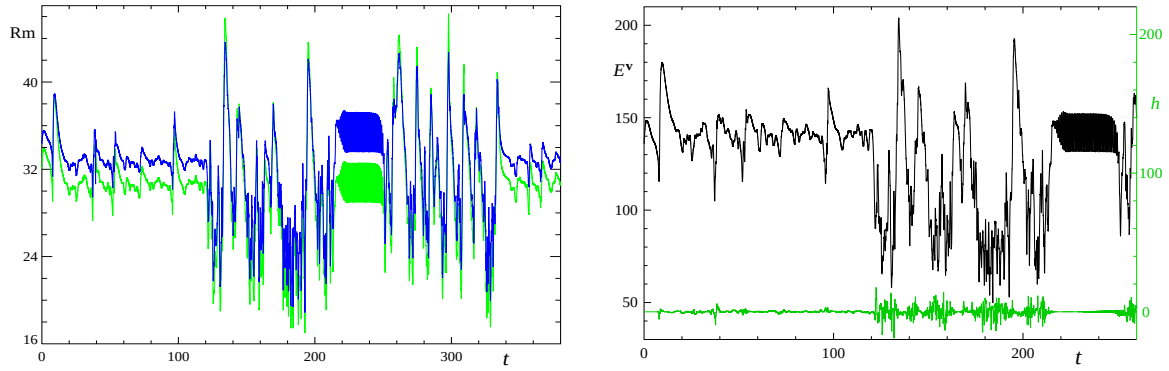
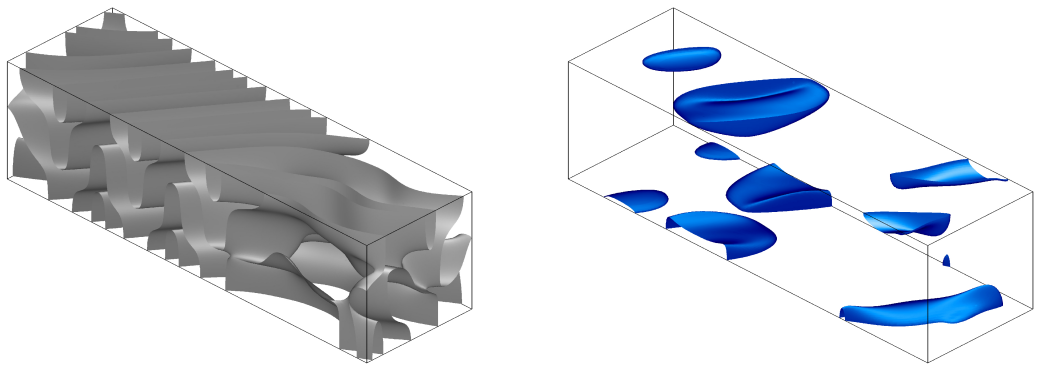


Figure 3.21 Temporal power spectra of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$ (left column) and $\mathrm{Re}\,\hat{v}^1_{222}$ (right column) at the time intervals $15 \leq t \leq 120$, (a) and (b), $132 \leq t \leq 212$, (c) and (d), and $223 \leq t \leq 242$, (e) and (f), in the run 4x1.1 for the standard resolution.



(a) (b)

Figure 3.22 Temporal behaviour of the local in time estimates of the magnetic Reynolds number, Rm_1' (green line) and Rm_3' (blue line), (a), and of the kinetic helicity (green line, right vertical axis) together with the kinetic energy (black line, left vertical axis) for reference, (b), in the run 4x1.1 .



(a) (b)

Figure 3.23 Isosurfaces of the kinetic (a) and magnetic (b) energy densities at the levels of 50% and 35% of the respective maxima of the densities in the run 4x1.1 at time $t = 222.626$. One quadruple-period fluid box is shown.

Rm_3' characterising the terminal quasiperiodic flow basically do not differ from those that characterise the terminal quasiperiodic flow in the triple-period elongated fluid cell, which is incapable of generation. We again observe that the magnetic Reynolds number values do not influence the generation of magnetic field.

Isosurfaces of the kinetic and magnetic energies Fig. 3.23 demonstrate the quasiperiodic flow depend on all the three spatial variables and does not acquire smaller spatial periods.

3.2.4 Regimes in the (M_1, M_2)-periodicity for $M_i \geq 2$

All runs in the "wide" fluid cells of the periodicity M_1xM_2 where both $M_i \geq 2$, exhibit similar behaviour. Temporal evolution of the kinetic and magnetic energies of the runs, 4x4.2 and 2x4.2 are shown in Fig. 3.24. The kinetic energy lacks a definite structure in its evolution and consists of chaotic oscillations of relatively low amplitude (see Fig. 3.24 (a)); in the

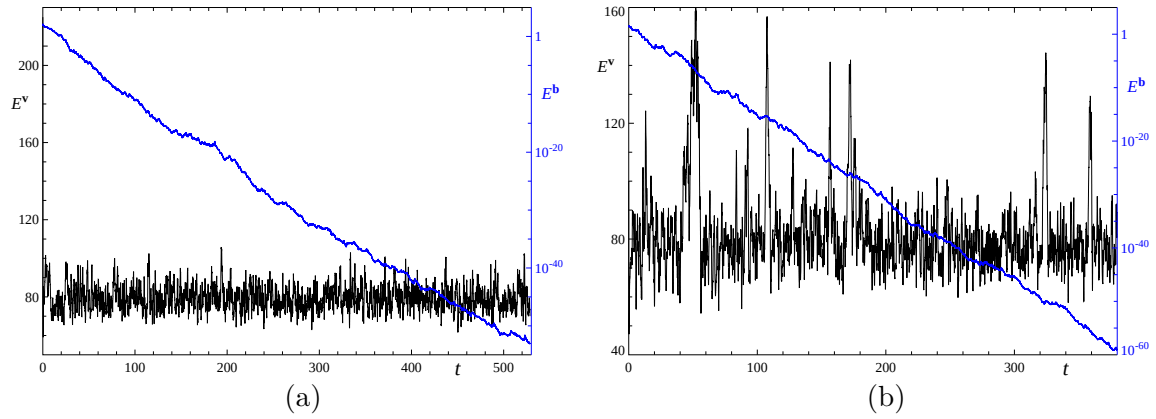


Figure 3.24 Temporal behaviour of the kinetic (left vertical axis, black line) and magnetic (right vertical axis, blue line) energies in the double-period runs 4x4.2 (a) and 2x4.2 (b).

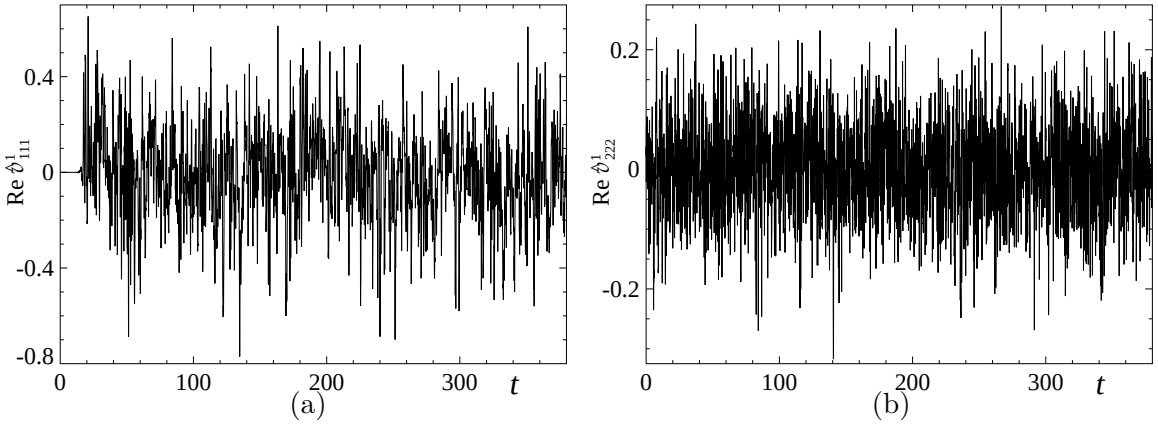


Figure 3.25 Temporal behaviour of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$, (a), and $\mathrm{Re}\,\hat{v}^1_{222}$ (b) in the run 2x4.2 .

run 4x4.2, the minimum, mean and maximum kinetic energy over time is 61.6, 79.4 and 105.7 respectively which can be superimposed upon random violent large-amplitude bursts of short duration with the energy increasing up to 160 (see Fig. 3.24(b)); in the run 2x4.2, the minimum and mean kinetic energy over time are 54.3 and 82, respectively.

The time- and space-averaged kinetic energy in all runs is roughly 80 independent of the size of the wide periodicity cell. It is noticeably smaller than the mean kinetic energy of flows in the elongated periodicity cells. It is also smaller than the temporal variation of the spatial mean (see Fig. 3.2). We interpret this as follows. According to Table 3.3 the characteristic length scales that determine the size of the hydrodynamic structures in these cells are below 0.25 . For $M_i\,L \geq 2$, the period in x_i equals $M_i\,L \geq 4\sqrt{2}$ which is more than 20 characteristic length scales. This relation suggests that the flow does not experience the constraints imposed by the periodicity condition and evolves as if they were absent. This ensures the independence of the integral flow parameters of the periodicity cell size. The relatively small kinetic energy and characteristic spatial scale length estimates in these regimes imply the smallness of the magnetic Reynolds number estimates.

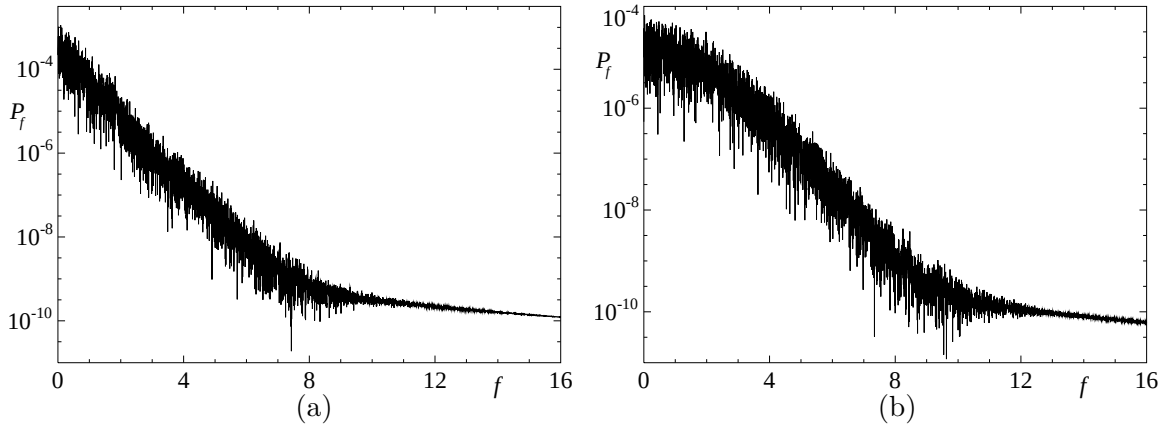


Figure 3.26 Temporal power spectra of the coefficients $\mathrm{Re}\,\hat{v}^1_{111}$ (a) and $\mathrm{Re}\,\hat{v}^1_{222}$ (b) at the time interval $40 \le t \le 381.21$ in the run 2×4.2 .

The chaotic behaviour of the kinetic energy is paralleled by a similar behaviour of individual harmonics; as typical examples, we present the plots of time dependence of the harmonics $\mathrm{Re}\,\hat{v}^1_{111}$ and $\mathrm{Re}\,\hat{v}^2_{111}$ in the run 2×4.2 (Fig. 3.25). Its chaotic nature is corroborated by the temporal power spectra and Poincaré sections (see Figs. 3.26 and 3.27, respectively). Because of the small size of the hydrodynamic structures and their chaoticity, they fail to develop a concerted action within themselves, which is needed for magnetic field generation. Besides this structural deficiency, the flow is relatively slow, implying that none of the flows in the wide cells is generating: in all runs for such fluid volumes, magnetic field exponentially decays with an occasional insignificant increase (see Fig. 3.24).

3.2.5 Videos of the kinetic and magnetic energy evolution

Visualizations of the flow and magnetic field evolutions from the runs 2×1.1, 2×2.1, 2×3.1, 4×2.1 and 4×4.1 are provided as supplementary materials for the reader. In the videos, isosurfaces of the kinetic and magnetic energy densities either of the fully simulated solution or of the perturbations (for the run 4×4.1) are shown. The video name indicates the size of the periodicity box shown in the video. On each frame, in the panel on the upper right corner plots of the spatially averaged kinetic (black line, left vertical axis) and magnetic (blue line, right vertical axis) energies as functions of time are shown. A vertical black line indicates the time of the current frame. Time is also displayed in digitial form in the information panel on left side of the video frame. In all videos except for the run 4×4.1 we show the isosurfaces in two adjacent periodicity boxes, thus providing a clearer illustration of the geometry.

It was established in [Chertovskih, 2015], that when the scale separation is high, amplitudes of a two-scale weakly nonlinear perturbation of the MHD regime $\mathrm{S}^{\mathrm{R1}}_8$, which only depends on slow spatial variables, resemble a solitary wave, moving along a straight line, gradually accelerating and increasing in magnitude until eventually blowing up in a finite time. Even if a perturbation adheres to the multiscale form studied in [Chertovskih, 2015], it is difficult to extract the amplitudes from it. Consequently, in an $M_1{\times}M_2$-periodicity box, we were

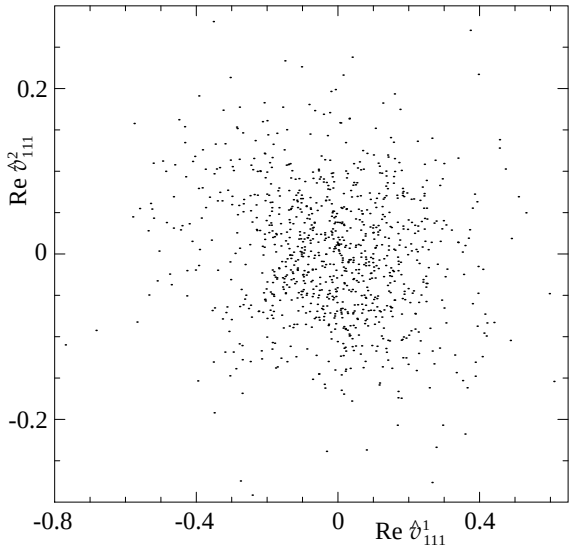


Figure 3.27 Poincaré section of the plane (Re $\hat{v}^1_{111}$, Re $\hat{v}^2_{111}$) defined by the condition Re $\hat{v}^3_{222} = 0$ for the runs 2×4.2, constructed for the time interval $40 \leq t \leq 381.21$.

considering the truncated Fourier series for the perturbed flow, where harmonics associated with the horizontal wave numbers n_i are retained only for $|n_i| < M_i$ - none of these harmonics belong to the 1×1-periodic state that we perturb and therefore, is involved in the amplitudes. However the"truncated amplitudes" fail to demonstrate the characteristic motion in the horizontal direction or an unlimited (or at least significant) growth in size similar to the amplitudes derived in [Chertovskih, 2015].

We have therefore switched to analysing videos in order to assess whether these features are present in the behaviour of the computed perturbations. The amplitudes of the two-scale weakly nonlinear perturbation investigated in [Chertovskih, 2015] evolve in slow time $T = \varepsilon^2 t$, where ε is the ratio of the small to the large spatial scales. In our direct numerical simulations, the smallest scale ratio $\varepsilon = 0.25$ characterises runs for the largest $4\times M/M\times 4$-periodicity box. Therefore we have used $\delta t = 0.01$ as the time step between subsequent frames.

A typical behaviour of kinetic and magnetic perturbations is shown in the perturbation video of 4×4.1 for the 4×4-periodicity box. Each frame presents isosurfaces of densities of energies of the hydrodynamic and magnetic perturbations at the level of 35% of the spatial maximum in time. Initially, we see the transients die out and the perturbations evolve very fast, but after $t = 0.15$ the evolution slows down considerably, and for $t > 0.5$ it becomes very slow. It again accelerates after $t = 5.5$ when the energies of the perturbations approach the maxima (see plots in the panel on the upper right section of the video), but it never is as fast as near $t = 0$. While for $t < 5.5$ the flow perturbation tends to respect the periodicity of the unperturbed steady state, subsequently it becomes completely chaotic in space (this happens to the blobs of magnetic energy at an earlier time). No traces of the behaviour, typical for the perturbation studied in [Chertovskih, 2015] are visible. We therefore conclude that even for this periodicity, characterised by the smallest in our runs, $\varepsilon = 0.25$, the perturbation is not in the asymptotic regime examined in [Chertovskih, 2015].

Another typical scenario of the perturbation behaviour is observed in the run 2×2.1 for the 2×2-periodicity box. Up to $t = 3.3$ the evolution of the flow is very slow but it is accelerating. Displacement of structures along the x_1-axis in the middle of the periodicity boxes (over a wide strip parallel to this axis) sets in and bends the isosurfaces of the energy densities; no such motion occurs close to the vertical sides of the periodicity boxes parallel to this axis. By $t = 5.5$, the isosurfaces of the kinetic energy density are in the form of rolls aligned with the diagonal of the horizontal base of the parallelepiped of periodicity. These rolls are non-generating and the magnetic field is decaying. At $t = 6.7$, they start bending and, at $t = 7.0$, a large eddy emerges. Its axis is vertical and the diameter equals the period of the unperturbed cell. The eddy efficiently "wipes out" both the flow and the magnetic field from a half of the periodicity cell volume and disintegrates. The space-averaged kinetic and magnetic energy is also significantly decreasing. By $t = 7.8$, the perturbation becomes unstructured and chaotic. A similar succession of events is observed, for instance, in the run 4×2.1. In the run 3×2.1 appearance of the "killer eddy" is less manifested. These events do not resemble what we observe in [Chertovskih, 2015].

The video of the evolution of the perturbed R_1 rolls (see [Chertovskih, 2010]) in the 2×1.1 in instructive. It shows isosurfaces of the energy densities of the flow and magnetic field in the 2×1-periodicity box at the level 12.0 and 3.0, respectively. We observe that the magnetic field evolves much faster than the kinetic energy. As in the 4×4.1 run, the evolution slows down significantly after the transients gradually die out around $t = 2.5$. Although the plots of the energies show that at this time the perturbation is still significant, the shape of the rolls at this level of kinetic energy is very close to that of the steady state R_1 subjected to perturbation. However, at $t = 7.35$, near the local minimum of 129.95 of the kinetic energy, the rolls becomes significantly distorted, they bend and are not any more parallel to the horizontal x_2-axis. A visibly fast evolution of the shape of the isosurfaces of the kinetic energy density observed in $11.5 < t < 16$, is still within the initial transient regime. After $t = 20$, during the time interval of more or less ordered cycles of magnetic field generation and decay, the magnetic energy shows a visibly time-periodic behaviour, while the kinetic energy density structure basically freezes, the cross-section of the rolls changes in shape and the rolls slightly bend only close to the local maxima of magnetic energy. This confirms our scenario describing the attractor revealed in the 2×1.1 (see section 3.2.1). No qualitative difference in the shape of isosurfaces of the kinetic or magnetic energy densities is visible in the sequences of cycles $20 < t < 50$ and $60 < t < 100$.

3.2.6 Analysis of integral parameters of the regimes

Let us define a norm in the Lebesgue space $L^P(\mathscr{C})$ denoted by $|\cdot|_p$,

$$|\mathbf{f}|_q = \Big(\frac{1}{\operatorname{vol}\mathscr{C}} \int_{\mathscr{C}} |\mathbf{f}|^q \,\mathrm{d}\mathbf{x}\Big)^{1/q} = \langle |\mathbf{f}|^q \rangle^{1/q}, \quad q \geq 1\,, \tag{3.13}$$

where $\operatorname{vol}\mathscr{C}$ is the volume of the periodicity cell $\mathscr{C}$. This norm may be regarded as non-standard because of the presence of the factor $1/\operatorname{vol}\mathscr{C}$, but it is equivalent to the usual definition.

Let $\|\cdot\|_s$ be a norm in the Sobolov Hilbert space $H^s(\mathscr{C})$ defined by

$$\|\mathbf{f}\|_s = \Big(\sum_{\mathbf{n}} |\mathbf{n}|^{2s} |\widehat{\mathbf{f}}_{\mathbf{n}}|^2\Big)^{1/2}, \quad \text{where} \quad \mathbf{f} = \sum_{\mathbf{n}\neq 0} \widehat{\mathbf{f}}_{\mathbf{n}}\, \mathrm{e}^{i\mathbf{n}\cdot\mathbf{x}}\,.$$

In [Zheligovsky, 2021], it was proven that for space-periodic magnetohydrodynamic flows and space-analytic initial conditions

$$\int_{t_0}^{T} \Big(\|\mathbf{v}\|_{q,s} + \|\mathbf{b}\|_{q,s}\Big)^{\alpha_{q,s}} \,\mathrm{d}t \;<\; \infty\,, \tag{3.14}$$

for

$$q \geq 2, \quad s \geq 3/q - 1/2, \quad T > 0\,. \tag{3.15}$$

The reader may refer to [Foias *et al.*, 1981, Gibbon, 2019] for the purely hydrodynamic case. Here, we denote

$$\alpha_{q,s} = q/(q(s+1) - 3)\,,$$

and

$$\|\mathbf{f}\|_{q,s} = |(-\nabla^2)^{s/2}\,\mathbf{f}|_q\,.$$

The present problem differs from that considered *ibid.* in two aspects: first, there is no periodicity in the vertical direction, and second, an additional equation, the heat transfer equation is involved. The boundary conditions (1.23) used here, however, are similar to those considered in [Foias *et al.*, 1981, Gibbon, 2019]. Therefore the solutions can be expanded in the sine and cosine series in the vertical direction, and the non-linearity in the heat transfer equation is analogous to the non-linearity in the Navier-Stokes and magnetic induction equations. Consequently, the conjecture that the integrals (3.14) are finite at any T seems plausible, and the above mentioned similarities allow us to use the same techniques as in [Zheligovsky, 2021].

For the parameter values specified in (3.15), the family of quantities

$$D_q^{\mathbf{v}} = \Big(\text{æ} A^{-1/q}\Big(\int_{\mathscr{C}} |\nabla\times\mathbf{v}|^q \,\mathrm{d}\mathbf{x}\Big)^{1/q}\Big)^{\alpha_{q,1}}, \quad \text{æ} = \mathscr{A}/\nu \tag{3.16}$$

was studied numerically in [Donzis *et al.*, 2013] (the rescaling factor æ was introduced in [Gibbon, 2010, 2011 and 2012]) for solutions $\mathbf{v}$ to the Navier-Stokes equation. Here, $\mathscr{A} = M_1\, M_2\, L^2$ is the area of the horizontal cross section of the flow periodicity box; in the non-dimensional case, when the layer has a unit width as it is considered here, the volume of $\mathscr{C}$ also equals $\mathscr{A}$. We analyse these quantities for the flow and the magnetic field. For the magnetic field, we replace $\mathbf{v}$ and ν in (3.16) with $\mathbf{b}$ and η.

Two possible scenarios were considered for the flow evolution in [Donzis *et al.*, 2013] and they are called regimes in [Donzis *et al.*, 2013]:

- D_q^v to decrease monotonically for all considered q at all times (scenario I); or
- at certain times this monotonicity can be broken for some q (scenario II).

By Hölder's inequality, $\langle|\mathbf{f}|^q\rangle^{1/q}$ monotonically increases on increasing $q \geq 1$ (unless $\mathbf{f}$ is constant in $\mathscr{C}$) and tends to the essential maximum of $|\mathbf{f}|$ for $q \to +\infty$, but the powers $\alpha_{q,s}$ decrease in q making scenario I possible [Donzis *et al.*, 2013]. It was numerically shown that all the four flow evolutions considered *ibid.* realise it for all the considered values $q \leq 18$. The above two scenarios can also be explored for the quantities

$$D^{\mathbf{f}}_{q,s} = \left(\varpi \langle|(-\nabla^2)^{s/2}\mathbf{f}|^q\rangle^{1/q}\right)^{\alpha_{q,s}} , \tag{3.17}$$

for q and s satisfying (3.15), where it is assumed $\mathbf{f} = \mathbf{v}$, $\mathbf{b}$ or the 6-dimensional field $(\mathbf{v}, \mathbf{b})$.

The behaviour observed *ibid.* can be interpreted as follows. Differentiating the integral (3.13) in q reveals that the inequality

$$q^2 \langle|\mathbf{f}|^q\rangle^{1-1/q} \frac{\mathrm{d}}{\mathrm{d}q}\langle|\mathbf{f}|^q\rangle^{1/q} = q\langle|\mathbf{f}|^q \ln|\mathbf{f}|\rangle - \langle|\mathbf{f}|^q\rangle \ln\langle|\mathbf{f}|^q\rangle \geq 0 , \tag{3.18}$$

holds true for any vector field $\mathbf{f}$ in $L^\infty(\mathscr{C})$ and this inequality is strict, when $\mathbf{f}$ is not constant. Now,

$$\frac{\mathrm{d}}{\mathrm{d}q} D^{\mathbf{v}}_{q,s} = D^{\mathbf{v}}_{q,s}\alpha_{q,s} q^{-2}\left(q\frac{\langle|\mathbf{f}|^q \ln|\mathbf{f}|\rangle}{\langle|\mathbf{f}|^q\rangle} - \ln\langle|\mathbf{f}|^q\rangle - 3\alpha_{q,s}\ln(\varpi|\mathbf{f}|_q)\right) , \tag{3.19}$$

for $\mathbf{f} = (-\nabla^2)^{s/2}\,\mathbf{v}$ (or for $\mathbf{f} = \nabla\times\mathbf{v}$, when $D^{\mathbf{v}}_q$ is considered). In view of (3.18), the first two terms constitute a positive quantity. Therefore, for $D^{\mathbf{v}}_{q,s}$ to grow for any $q \to +\infty$ depends on the sign and the magnitude of the last term.

It is instructive to consider the model $\mathbf{f}$ such that $|\mathbf{f}|$ is constant on a set of Lebesgue measure μ, and $\mathbf{f} = 0$ everywhere else in $\mathscr{C}$. In this case, for s satisfying (3.15), $\mathrm{d}D^{\mathbf{v}_{q,s}}/\mathrm{d}q > 0$ provided

$$|\mathbf{f}| < (\mathscr{A}/\varphi)^{(s+1)/3}/\varpi . \tag{3.20}$$

This suggests that the onset of scenario II is possible for sufficiently small $\mu/\mathscr{A}$, i.e. for sufficiently space-intermittent flows, where intense eddies are highly localised. Notably, the condition (3.20) does not explicitly involve the norm index q.

We have computed three pairs of quantities for the flow and magnetic field obtained in the computations for both resolutions: D_q and $D_{q,2}$ with $2 \leq q \leq 10$ and $D_{2,s}$ with $2 \leq s \leq 10$. Fig. 3.28 presents the temporal behaviour of D_q and $D_{q,s}$ in the run 4x1.1; In our simulations, the flow follows scenario I. That is, there is a monotonic decrease of $D^{\mathbf{v}}_q$ in q for all runs in both resolutions. We also note that at $t = 0$, the monotonicity is not broken for any of the considered quantities. The robustness of this property is remarkable, in particular, for the following reason: here we were solving a complete system (3.1) - (3.3) of equations of the thermal convective dynamo although only solutions to the Navier-Stokes equations were considered in (3.16). In (3.16) it is interpreted that fast accumulation of $D^{\mathbf{v}}_q$ for $q \geq 4$ (see Fig. 3.28(a)) indicates the presence of significant depletion, i.e., cancellation in the vector products constituting the nonlinear terms in (3.1) - (3.3) and its decrease upon removal of the gradient parts in the Navier-Stokes equation. The similarity of the graphs of $D^{\mathbf{v}}_q$ for $q \geq 4$ as functions of time can be attributed to the fast convergence of $\langle|\nabla\times\mathbf{v}|^q\rangle^{1/q}$ and

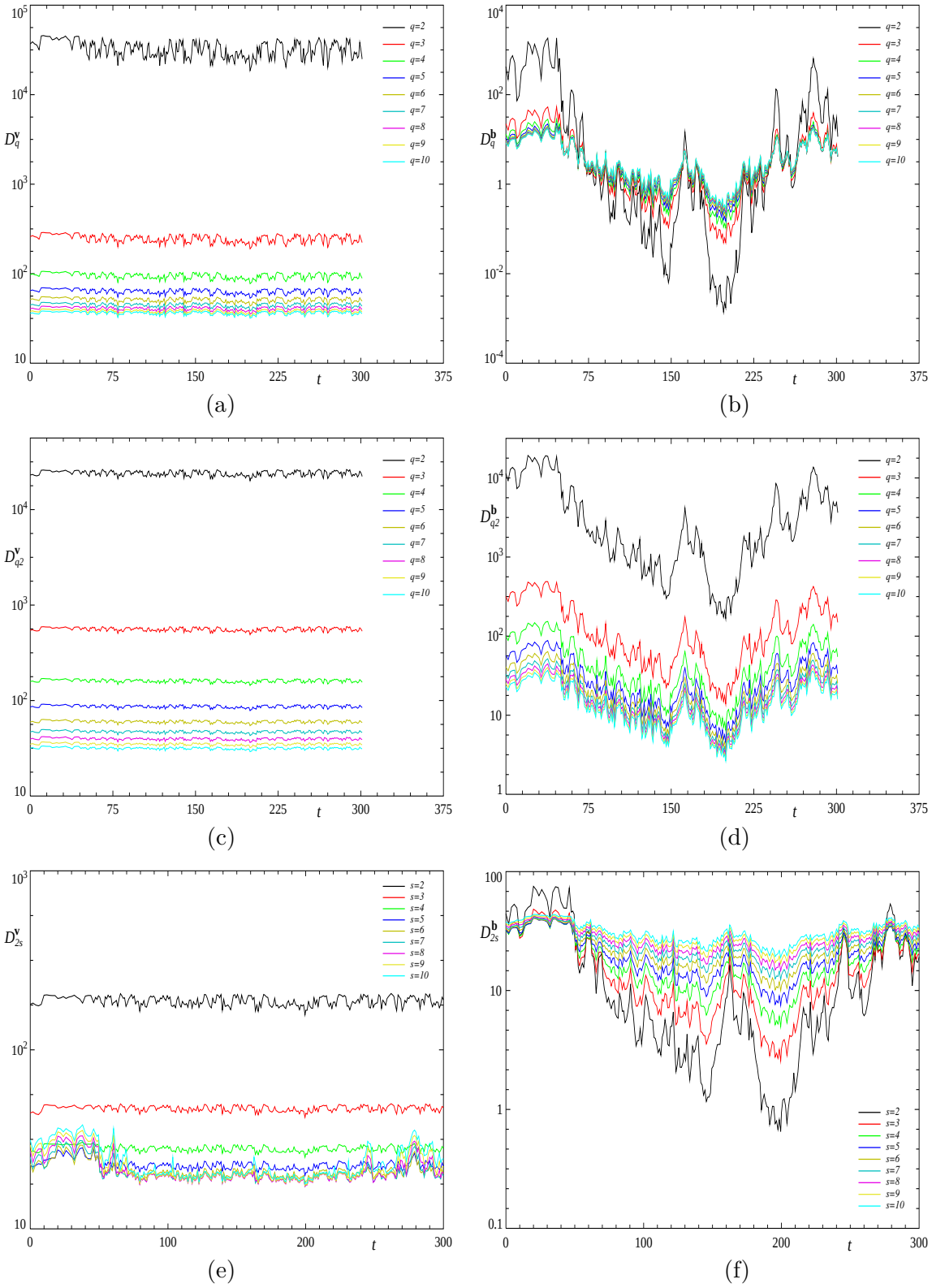


Figure 3.28 Time evolution of the quantities $D_q^{\mathbf{v}}$ (a), $D_{q,s}^{\mathbf{v}}$ (panels (c), (e)), $D_q^{\mathbf{b}}$ (b) and $D_{q,s}^{\mathbf{b}}$ (panels (d), (f)) for $s = 2$ and q varying in the range $2 \leq q \leq 10$ (panels (c), (d)), and for $q = 2$ and s in the range $1 < s \leq 10$ (panels (e), (f)) in the run 4×1.1 .

$\alpha_{q,s}$ on increasing q to the essential maxima of $|\nabla \times \mathbf{v}|$ and $1/(s+1)$, respectively. The fast convergence of $\langle|\nabla \times \mathbf{v}|^q\rangle^{1/q}$ and $\langle|\nabla \times \mathbf{b}|^q\rangle^{1/q}$ primarily characterises the distributions of values of $|\nabla \times \mathbf{v}|$ and $|\nabla \times \mathbf{b}|$. The fast-converging powers $\alpha_{q,s}$ in the definition of D_q may have been chosen arbitarily, owing to the fact that these powers are the maximum for which we can prove (3.14) [Foias, 1981, Zheligovsky, 2021] (this indicates that we can estimate depletion only roughly).

Similarly, we observe that the values of $D^{\mathbf{v}}_{q,2}$ monotonically decrease in q in all runs (see Fig. 3.28(c)). For larger values of q and/or for other s, or when computed for the magnetic field, the monotonicity of D_q or $D_{q,s}$ can partially break down or be completely inverted (at least in the considered parameter range $2 \le q \le 10$). For instance, in the run 4x1.1, $D^{\mathbf{b}}_q$ decreases in q as $D^{\mathbf{v}}_q$, for $t \le 45$, but monotonically increases in q on the time interval $80 \le t \le 160$ (see. Fig. 3.28(b)). Similarly, the quantities $D_{q,s}$ may decrease in s for small values of the parameter, but they begin to increase for larger s (e.g., for $60 \le t \le 270$, $D^{\mathbf{b}}_{q,s}$ monotonically increases in s for all $s \le 10$ for which computations were performed, see Fig. 3.28(f)). In some runs $D^{\mathbf{b}}_{q,s}$ for $q = 2$ increases monotonically in s for all considered s in the range $1 < s \le 10$ on some time intervals; by contrast, a monotonic increase of $D^{\mathbf{v}}_{q,s}$ in s is broken at all times in all runs.

3.3 Summary

The highlights in this chapter:

- Computations were performed for two resolutions: $128M_1$ x $128M_2$ x 96 (standard; (M_1, M_2) represents a periodicity-box and $1 \le M_1, M_2 \le 4$) and 256 x 256 x 257 (alternative). The alternative resolution was kept constant with respect to the periodicity box and was used for runs where the energy spectrum decay of the flow was below 7 in the standard resolution. Results from both were qualitatively checked.

- The attractors and metastable states possess a single pair of symmetry for the flow and magnetic field. They are either identical or the magnetic field additionally comprises the reverse symmetry q .

- Two distinct MHD chaotic attractors are identified in the double-period cells and an unique chaotic attractor in the quadruple-period cells. The attractors in the triple-period and in the wide-period cells are hydrodynamic.

- The characteristic lengths of wide-period cells evolves to a fraction of the period of initial state resulting in a less arduous convection and in the subsequent ceasing of magnetic field generation.

- The scale ratio for the observed perturbations of the regimes in the transitory states is insufficiently small.

- In all the runs, the computed D_q and $D_{q,2}$ for the flow decrease monotonically for $q \le 10$; However monotonicity is broken when these quantities are computed for the

magnetic field. The behaviour of $D_{q,s}$ in s is non-monotonic both for the flow and magnetic field.

Chapter 4

Conclusion

The present study was to numerically explore the thermal space-periodic convection of an electrically conducting fluid in a horizontal layer rotating about the vertical axis. In this pursuit we simulated three large-scale perturbations. The fully nonlinear perturbations of the MHD steady state identified in [Chertovskih *et al.*, 2010] and fields with randomly generated Fourier coefficients with an exponentially decay spectrum were considered as initial conditions. From these simulations the temporal evolution of small perturbations was investigated, where the horizontal periods are up to four times larger than the spatial period of the MHD state subjected to perturbation. The primary purpose is to understand how the perturbations evolve and locate any evidence of a multiscale weakly nonlinear perturbation [Chertovskih, 2015]. However their evolutions were not as anticipated: convergence to a small scale neutral stability mode of a constant amplitude or a blow-up at a finite slow time was expected. This may be because of two reasons:

- First, the scale ratio $\varepsilon = 1/4$ is probably large for the regimes that were found in [Chertovskih, 2015] by asymptotic methods.
- Second, it is possible that intermediate-scale perturbations set in, which is a common scenario for all linear stability problems in the MHD realm. Some of these problems include purely hydrodynamic perturbations of flows, magnetic modes arising in the context of the kinematic dynamo problem (that is, magnetic perturbations of flows of electrically conducting fluids), and full MHD perturbations of MHD regimes [Zheligovsky, 2020 and 2022].

We also identified attractors and metastable states in the respective regimes. Some of these findings are:

- The attractors in the double and quadruple-period elongated cells for small and large initial perturbation remain as chaotic MHD. That is, the magnetic field does not decay over time in the terminal regime.
- Two distinct MHD attractors are idenitifed in the double-period fluid cells and an unique MHD attractor in the quadraple-period fluid cells.

- The fluid cells of the 2:1 and 4:1 aspect ratios – with one of the horizontal spatial periods twice or four times larger than the period of the unperturbed steady state subjected to perturbation - have not revealed any hydrodynamic attractors with their magnetic field ultimately decaying.
- The attractors in the triple-period elongated fluid cells are classified as hydrodynamic.
- Simulations for the fluid cells whose horizontal spatial periods are at least twice larger than that of the MHD state subjected to perturbation, also have resulted in hydrodynamic attractors.

Characteristic length scales of the flow structures defined in terms of ratios (3.12) of the Sobolev norms (see [Doering and Gibbon, 1995]) were also estimated. Using these estimates and the root-mean square velocity, estimates of the magnetic Reynolds numbers were computed. The obtained values are relatively small, typically below 35.6. But, neither the Rm values, nor the mean helicity have separated the generating and non-generating (triple-period) attractors in the elongated cells.

Attractors in wide periodicity cells are also characterised by smaller estimates of Rm. It is because their characteristic length scales have become significantly shorter and they are below the period of the unpertubed state. Consequently, convection becomes less vigorous and generation of magnetic field ceases. The length scale estimates and the kinetic energy remain approximately the same for all the non-elongated fluid cells – the flow does not experience the constraints of the "boundaries" of the periodicity cells.

A parallel program was developed in Fortran for numerical integration of the convective dynamo problem forward in time. Software libraries, FFTW [Frigo and Johnson, 1998], for computing Fast Fourier Transforms, and P3DFFT [Pekurovsky, 2012] (a wrapper arround FFTW), for data manipulation in the parallel environment, were used.

We also analyzed a family of integral quantities $D_{q,s}$ as in [Donzis *et al.*, 2013]. In all our runs, D_q and $D_{q,2}$ computed for the flow decrease monotonically for all the considered values $q \leq 10$; Nevertheless this monotonicity is broken at certain times for the magnetic field. The behaviour of $D_{q,s}$ in s is non-monotonic both for the flow and the magnetic field.

Another compelling work from this study is to explain the behaviour of the kinetic helicity of the weakly nonlinear regime 2x1.1, in terms of appropriate asymptotic expansions. We intend to carry out this task in the future. From our simulation we observe that the kinetic helicity is almost regular at certain times and reversed with a similar amplitude in the subsequent or other time intervals.

There is also interest to explore the evolution of large-scale perturbations of other steady regimes ($\mathrm{S}_2^{\mathrm{R1}}, \mathrm{S}_3^{\mathrm{R1}}, \mathrm{S}_4^{\mathrm{R1}}, \mathrm{S}_6^{\mathrm{R1}}, \mathrm{S}_7^{\mathrm{R1}}, \mathrm{S}_1^{\mathrm{RD}}, \mathrm{S}_2^{\mathrm{RD}}, \mathrm{S}_3^{\mathrm{RD}}$ and $\mathrm{S}_4^{\mathrm{RD}}$) found in [Chertovskih *et al.*, 2010, Zheligovsky, 2010]. These regimes are symmetric about the vertical axis and possess additional spatial symmetries. The relevant amplitude equations were derived in [Zheligovsky, 2011] and yet to be studied numerically. So all this opens up for future work.

Bibliography

[1] Arnold V.I., Korkina E.I. The growth of a magnetic field in a three-dimensional incompressible flow. Vestnik Moscow State Univ. Ser. Math., **3**, 43–46 (1983), in Russian.

[2] Bullard, E.C., Gellman, H.: Homogeneous dynamos and terrestrial magnetism. Philos. Trans. R. Soc. Lond., **247**, 213—278 (1954).

[3] Canuto C., Hussaini M.Y., Quarteroni A., Zang T.A. Spectral Methods in fluid dynamics. Springer, NY (1988).

[4] Conte, S.D., De Boor, C. Elementary numerical analysis: an algorithmic approach. SIAM (2017).

[5] Chandrasekhar, S. Hydrodynamic and hydromagnetic stability. Oxford University Press (1961).

[6] Chertovskih R., Chimanski E.V., Rempel E.L. Route to hyperchaos in Rayleigh–Bénard convection. Europhysics Lett., **112**, 14001 (2015). [https://arxiv.org/abs/1608.04716]

[7] Chertovskih R., Gama S.M.A., Podvigina O., Zheligovsky V. Dependence of magnetic field generation by thermal convection on the rotation rate: a case study. Physica D, **239**, 1188–1209 (2010). [https://arxiv.org/abs/0908.1891].

[8] Chertovskih R., Rempel E.L., Chimanski E.V. Magnetic field generation by intermittent convection. Physics Lett. A, **381**, 3300–3306 (2017). [https://arxiv.org/abs/1608.04716]

[9] Chertovskih R., Zheligovsky V. Large-scale weakly nonlinear perturbations of convective magnetic dynamos in a rotating layer. Physica D, **313**, 99–116 (2015) [https://arxiv.org/abs/1504.06856].

[10] Cooley, J. W., Tukey, J. W. An algorithm for the machine calculation of complex Fourier series. Mathematics of computation, 19(**90**), 297-301 (1965).

[11] Cowling, T.G., The magnetic field of sunspots. Mon. Not. R. Astron. Soc., **94**, 39–48 (1934).

[12] Cowling, T.G., The dynamo maintenance of steady magnetic fields. Quart. J. Mech. Appl. Math., **10**, 129–136 (1957).

[13] Cox S.M., Matthews P.C. Exponential time differencing for stiff systems. J. Computational Physics, **176**, 430–455 (2002).

[14] Demircan A., Seehafer N. Dynamo in asymmetric square convection. Geophys. Astrophys. Fluid Dyn., **96**, 461–479 (2002).

[15] Doering C.R., Gibbon J.D. Applied analysis of the Navier–Stokes equations. Cambridge Univ. Press (1995).

[16] Donzis D.A., Gibbon J.D., Gupta A., Kerr R.M., Pandit R., Vincenzi D. Vorticity moments in four numerical simulations of the 3D Navier–Stokes equations. J. Fluid Mech., 732, 316–331 (2013). [https://arxiv.org/abs/1302.1768]

[17] Dubrulle B., Frisch U. Eddy viscosity of parity-invariant flow. Phys. Rev. A, **43**, 5355–5364 (1991).

[18] Elsasser, W.M.: Induction effects in terrestrial magnetism. Phys. Rev., **69**, 106–116 (1946).

[19] Foias C., Guillopé C., Temam R. New a priori estimates for Navier–Stokes equations in dimension 3. Comm. Partial Diff. Equations, 6, 329–359 (1981).

[20] Frigo, M., Johnson, S. G. FFTW: An adaptive software architecture for the FFT. In Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181), vol. 3, 1381-1384. IEEE (1998, May).

[21] Frisch, U. Turbulence: the legacy of A.N. Kolmogorov. Cambridge University Press (1995).

[22] Gertsenshtein S.Ya., Zheligovsky V.A., Podvigina O.M., Chertovskih R.A. On magnetic field generation by three-dimensional convective flows of conducting fluid in a rotating horizontal layer. Doklady Akademii Nauk, **417** №5, 613–615 (2007). (Engl. transl.: Gertsenshtein S.Ya., Zheligovsky V.A., Podvigina O.M., Chertovskih R.A. Generation of a magnetic field by three-dimensional convective flows of a conducting fluid in the rotating horizontal layer. Doklady Physics, **52**, 653–655 (2007).)

[23] Gertsenshtein S.Ya., Chertovskikh R.A., Generation of a magnetic field by convective flows in a rotating horizontal layer, Izvestiya Rossiiskoi Akademii Nauk, Mekhanika Zhidkosti i Gaza, 43, №2, 92—101 (2008). (Engl. transl.: Gertsenshtein S.Ya., Chertovskikh R.A. Generation of a magnetic field by convective flows in a rotating horizontal layer. Fluid Dyn., 43, 248–256 (2008).

[24] Gibbon J.D. Regularity and singularity in solutions of the three-dimensional Navier–Stokes equations. Proc. R. Soc. A, **466**, 2587–2604 (2010). [https://arxiv.org/abs/0905.0344]

[25] Gibbon J.D. A hierarchy of length scales for weak solutions of the three-dimensional Navier–Stokes equations. Comm. Math. Sci., **10**, 131–136 (2011). [https://arxiv.org/abs/1012.3604]

[26] Gibbon J.D. Conditional regularity of solutions of the three-dimensional Navier–Stokes equations and implications for intermittency. J. Math. Phys., **53**, 115608 (2012). [https://arxiv.org/abs/1108.4651]

[27] Gibbon J.D. Weak and strong solutions of the 3D Navier–Stokes equations and their relation to a chessboard of convergent inverse length scales. J. Nonlin. Sci., 29, 215–228 (2019). [https://arxiv.org/abs/1803.11518]

[28] Hochbruck, M., Ostermann, A. Exponential integrators. Acta Numerica, **19**, 209-286 (2010).

[29] Hutt, A., Beim Graben, P. Sequences by metastable attractors: interweaving dynamical systems and experimental data. Frontiers in Applied Mathematics and Statistics, 3, 11 (2017).

[30] Iserles, A. A first course in the numerical analysis of differential equations № 44. Cambridge university press (2009).

[31] Jeyabalan S.R., Chertovskih R., Gama S.M.A., Zheligovsky V. Nonlinear large-scale perturbations of a steady thermal convective dynamo regimes in a plane layer of electrically conducting fluid rotating about the vertical axis. Mathematics, **10**, № 16: 2957 (2022). [https://doi.org/10.3390/math10162957]

[32] Kosobokov V.G. On a regularity in the sequence of the Earth's magnetic field reversals. Doklady Akademii Nauk, **340** № 4, 539–542 (1995). (Engl. transl.: Kosobokov V.G. Pattern in the succession of inversions of the terrestrial magnetic field. Transactions (Doklady) of the Russian Academy of Sciences. Earth Science Sections. **345** A № 9, 128–132 (1996).)

[33] Krause F., Rädler K.-H. Mean-field magnetohydrodynamics and dynamo theory. Academic-Verlag, Berlin (1980).

[34] Larmor, J. 17. How Could a Rotating Body such as the Sun Become a Magnet?. In A Source Book in Astronomy and Astrophysics, 1900–1975, 106-107. Harvard University Press (2013).

[35] Lanotte A., Noullez A., Vergassola M., Wirth A. Large-scale dynamo by negative magnetic eddy diffusivities. Geophys. Astrophys. Fluid Dyn. 91, 131–146 (1999).

[36] Matthews P.C. Dynamo action in simple convective flows. Proc. R. Soc. Lond. A, **455**, 1829–1840 (1999).

[37] Meneguzzi M., Pouquet A. Turbulent dynamos driven by convection. J. Fluid Mech., **205**, 297–318 (1989).

[38] Moffatt, H.K. Magnetic Field Generation in Electrically Conducting Fluids. CUP: Cambridge, 1978.

[39] Müller, U., Stieglitz, R.: The Karlsruhe dynamo experiment. Nonlinear Process. Geophys., **9**, 165-–170 (2002).

[40] Neill, S. P., Hashemi, M. R. Fundamentals of ocean renewable energy: generating electricity from the sea. Academic Press (2018).

[41] Parker E.N. Hydrodynamic dynamo models. Astrophys. J. **122**, 293–314 (1955).

[42] Parker E.N. Cosmical magnetic fields: Their origin and their activity. Clarendon Press (1979).

[43] Pekurovsky, D. P3DFFT: A framework for parallel computations of Fourier transforms in three dimensions. SIAM Journal on Scientific Computing, **34**, № 4, C192-C209 (2012). [https://arxiv.org/abs/1905.02803]

[44] Peyret R. Spectral methods for incompressible viscous flow. Springer Verlag, Berlin (2002).

[45] Podvigina O.M. Magnetic field generation by convective flows in a plane layer. Eur. Phys. J. B, **50**, 639–652 (2006).

[46] Podvigina O.M. Magnetic field generation by convective flows in a plane layer: the dependence on the Prandtl number. Geophys. Astrophys. Fluid Dyn., **102**, 409–433 (2008).

[47] Rädler, K.-H., Apstein, E., Rheinhardt, M., Schüler, M.: The Karlsruhe dynamo experiment. A mean field approach. Stud. Geophys. Geodaetica 42, 224—231 (1998).

[48] Rädler K.-H. Mean-field dynamo theory: Early ideas and today's problems. In: Magnetohydrodynamics. Historical evolution and trends. Eds. Molokov S., Moreau R., Moffatt K. Fluid mechanics and its applications, **80**. Springer, 55–72 (2007).

[49] Rasskazov A., Chertovskih R., Zheligovsky V. Magnetic field generation by pointwise zero-helicity three-dimensional steady flow of incompressible electrically conducting fluid. Phys. Rev. E, **97**, 043201 (2018). [https://arxiv.org/abs/1708.08770]

[50] Roberts G.O. Spatially periodic dynamos. Phil. Trans. Roy. Soc. Lond. A, **266**, 535–558 (1970).

[51] Roberts G.O. Dynamo action of fluid motions with two-dimensional periodicity. Phil. Trans. Roy. Soc. Lond. A, **271**, 411–454 (1972).

[52] Ruelle, D., Isola, S. Chaotic evolution and strange attractors, vol. 1. Cambridge University Press (1989).

[53] Steenbeck M., Krause F., Rädler K.-H. Berechnung der mittleren Lorentz-Feldstärke $\overline{\mathbf{v} \times \mathbf{b}}$ für ein elektrisch leitendes Medium in turbulenter, durch Coriolis–Kräfte beeinfluißter Bewegung. Z. Naturforsch. 21a, 369–376 (1966). Engl. transl.: A calculation of the mean electromotive force in an electrically conducting fluid in turbulent motion, under the influence of Coriolis forces. In Roberts P.H., Stix M. The turbulent dynamo: A translation of a series of papers by F. Krause, K.-H. Rädler, and M. Steenbeck. Tech. Note NCAR-TN/IA-60, Boulder, Colorado, 29–47 (1971). http://nldr.library.ucar.edu/repository/assets/technotes/TECH-NOTE-000-000-000-045.pdf.

[54] Stieglitz, R., Müller, U.: Experimental demonstration of a homogeneous two-scale dynamo. Phys. Fluids 13, 561-–564 (2001).

[55] Stieglitz, R., Müller, U.: Experimental demonstration of a homogeneous two-scale dynamo. Magnetohydrodynamics 38, 27-–33 (2002).

[56] Tilgner, A., Busse, F.H.: Simulation of the bifurcation diagram of the Karlsruhe dynamo. Magnetohydrodynamics 38, 35—40 (2002).

[57] Vishik M.M. Periodic dynamo. In: Mathematical methods in seismology and geodynamics (Computational seismology, iss. 19). Keilis-Borok V.I., Levshin A.L. (eds.), 186–215. Nauka, Moscow (1986), in Russian. Engl. transl.: Computational seismology, **19**, 176–209. Allerton Press, NY (1987).

[58] Vishik M.M. Periodic dynamo. II. In: Numerical modelling and analysis of geophysical processes (Computational seismology, iss. 20). Keilis-Borok V.I., Levshin A.L. (eds.), 12–22. Nauka, Moscow (1987), in Russian. Engl. transl.: Computational seismology, **20**, 10–21. Allerton Press, NY (1988).

[59] Zeldovich Ya.B. The magnetic field in the two-dimensional motion of a conducting turbulent fluid. Journ. Exper. Theor. Phys. **31**, 154–156 (1956). Engl. transl.: Sov. Phys. J.E.T.P. **4**, 460–462 (1957).

[60] Zeldovich, Ya.B., Ruzmaikin, A.A.: The magnetic field in a conducting fluid in twodimensional motion. J. Exp. Theor. Phys. 78, 980–986 (1980). Engl. transl.: Sov. Phys. J.E.T.P 51, 493—497 (1980).

[61] Zheligovsky V.A. A kinematic magnetic dynamo sustained by a Beltrami flow in a sphere. Geophys. Astrophys. Fluid Dynamics, **73**, 217–254 (1993).

[62] Zheligovsky, V. A. Numerical solution of the kinematic dynamo problem for Beltrami flows in a sphere. Journal of scientific computing, **8** № 1, 41-68 (1993).

[63] Zheligovsky V.A. On the linear stability of steady space-periodic magnetohydrodynamic systems to large-scale perturbations. Physics of the Earth, № 5, 65–74 (2003) [arxiv.org/abs/nlin/0512076], in Russian. (Engl. transl.: Zheligovsky V.A. On the linear

stability of spatially periodic steady magnetohydrodynamic systems with respect to long-period perturbations. Izvestiya, Physics of the Solid Earth, **39** № 5, 409–418 (2003).)

[64] Zheligovsky V.A. Amplitude equations for weakly nonlinear two-scale perturbations of free hydromagnetic convective regimes in a rotating layer. Geophys. Astrophys. Fluid Dyn., **103**, 397–420 (2009). [https://arxiv.org/abs/0809.1195v2].

[65] Zheligovsky V. Generation of a symmetric magnetic field by thermal convection in a plane rotating layer. Magnetohydrodynamics, **46**, 3–22, (2010). [https://arxiv.org/abs/0906.5380]

[66] Zheligovsky V.A. Large-scale perturbations of magnetohydrodynamic regimes: linear and weakly nonlinear stability theory. Lecture Notes in Physics, vol. 829, Springer-Verlag, Heidelberg (2011).

[67] Zheligovsky V. Space analyticity and bounds for derivatives of solutions to the evolutionary equations of diffusive magnetohydrodynamics. Mathematics, **9**, 1789 (2021). https://doi.org/10.3390/math9151789

[68] Zheligovsky V.A., Chertovskih R.A. On kinematic generation of magnetic modes of the Bloch type. Physics of the Earth № 1, 118–132 (2020). (Engl. transl.: Zheligovsky V.A., Chertovskih R.A. On kinematic generation of the magnetic modes of Bloch type. Izvestiya, Physics of the Solid Earth, **56**, 104–117 (2020).)

[69] Zheligovsky V.A., Chertovskih R.A. Linear perturbations of the Bloch type of space-periodic magnetohydrodynamic steady states. Manuscript in preparation (2022).

[70] Zheligovsky V.A., Galloway D.J. Dynamo action in Christopherson hexagonal flow. Geophys. Astrophys. Fluid Dynamics, **88**, 277–293 (1998).

[71] Zheligovsky V.A., Podvigina O.M, Frisch U. Dynamo effect in parity-invariant flow with large and moderate separation of scales. Geophys. Astrophys. Fluid Dynamics, **95**, 227–268 (2001). [https://arxiv.org/abs/nlin/0012005]

Appendix A

Software

A.1 Code verification

In this section we describe the tests that were performed to validate the program code and how its outputs were verified against the values computed analytically and from results obtained by other researchers in their studies.

The tests were carried out in two phases: In phase 1, the evolution of the magnetic field was checked. The equations governing the fluid and temperature were excluded. In phase 2, the complete system (3.1),-(3.3) of equations of thermal convective dynamo was considered and their respective evolution was checked.

A.1.1 Linear dynamo

In phase 1, three different flows were used: first, a motionless flow, second, a divergence-free flow and third, a non divergence-free flow. In each case the system was integrated in time with its solenoidal and boundary conditions and the evolution of magnetic energy was observed. The rate of growth or decay was computed for each of the runs. The chosen initial magnetic field, $\mathbf{b}$, was small, of the order 10^{-2}. The spatial resolutions used were $48 \times 48 \times 25$ and $64 \times 64 \times 33$. Dealiasing was excluded in the computations.

- Case 1. A fluid at rest, $\mathbf{v} = 0$, with $P = 1$ and $P_m = 8$. For a steady hydrodynamic state of a system, a linear (or kinematic) dynamo problem is an eigenvalue problem. Thus,

$$\frac{P}{P_m}\nabla^2\mathbf{b} = \lambda\,\mathbf{b}\,.$$

  Here, the eigenvalues are computed as below.

$$\lambda = -\frac{P}{P_m}|\tilde{\mathbf{k}}|^2 = -\frac{P}{P_m}(\alpha^2 k_1^2 + \alpha^2 k_2^2 + \pi^2 k_3^2).$$

  The dominant eigenvalue λ_d is

$$\lambda_d = \lambda_{(1,0,0)} = \lambda_{(0,1,0)} = -\frac{1}{8}\frac{\pi^2}{2} \approx -0.617\,,$$

which concurs with the dominant eigenvalue observed in computations.

- Case 2. A divergence-free velocity field, $\mathbf{v}$,

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} 2\sin(x_2)\cos(2\pi x_3) \\ \pi\cos(3x_2)\cos(\pi x_3)/2 \\ 3\alpha_2\sin(3x_2)\sin(\pi x_3)/2 \end{pmatrix},$$

  with $P/P_m = 0.06$. The algorithm of Zheligovsky [1993] was used to compute its dominant eigenvalue λ_d, which is 0.096966. The slope of the dominant eigenvalue drawn in Figure A.1 is consistent with this aforementioned value.

- Case 3. A non-divergence-free velocity field, $\mathbf{v}$,

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = \begin{pmatrix} \cos(x_1 - x_2)\cos(3\pi x_3)/2 - \sin(2x_1 + x_2)\cos(\pi x_3)/3 - \cos(2x_1 - 2x_2)/2 \\ \sin(2x_1 - x_2)\cos(2\pi x_3)/2 + \cos(2x_1 - 2x_2)/3 + \sin(x_1) \\ \cos(x_1 + 2x_2)\sin(2\pi x_3) + \sin(3x_1 - 2x_2)\sin(\pi x_3)/2 \end{pmatrix},$$

  with $P/P_m = 0.0075$. In this test, the growth rate was determined experimentally. A line parallel to the magnetic energy was drawn – in a linear regime the magnetic energy E_b grows as $e^{2\,\mathrm{Re}(\lambda_d)\,t}$ – and its slope was estimated to be 0.12599. Using the sequential code [Chertovskih, 2010], the computed dominant eigenvalue λ_d, however, is 0.127 , which again is in agreement with our estimation.

A.1.2 Non-linear dynamo

In order to verify the non-linear section of the code, we simulated the computation performed by Podvigina [2006] of a magnetic field generation in a non-rotating layer ($Ta = 0$).

The inputs to the system: a non-transient state of a hydrodynamic attractor for $\mathbf{v}$ and θ and a small magnetic field, $\mathbf{b}$. The run was executed for the parameters

$$L = 2\sqrt{2}, \quad R = 2300, \quad P = 1, \quad P_m = 8\,,$$

with a spatial resolution $48 \times 48 \times 25$. The solenoidal and boundary conditions for $\mathbf{v}$ and $\mathbf{b}$ were unchanged. The number of processors used was 4 and its grid size was 2×2.

The initial energies of the fields were as follows:

$$E_v = 226.75, \quad E_b = 0.79 \times 10^{-6}, \quad E_t = 0.58 \times 10^{-1}\,.$$

As it was expected, the initial growth of the magnetic energy was an exponential decay ($0 < t < 10$). Then there was a magnetic field generation in the interval $10 < t < 30$. From $t > 35$, the kinetic and magnetic fields became periodic and a saturated regime was observed at subsequent times.

The computed dominant eigenvalue λ_d is 0.98 . The above results agree with the values computed by Podviginia [2006].

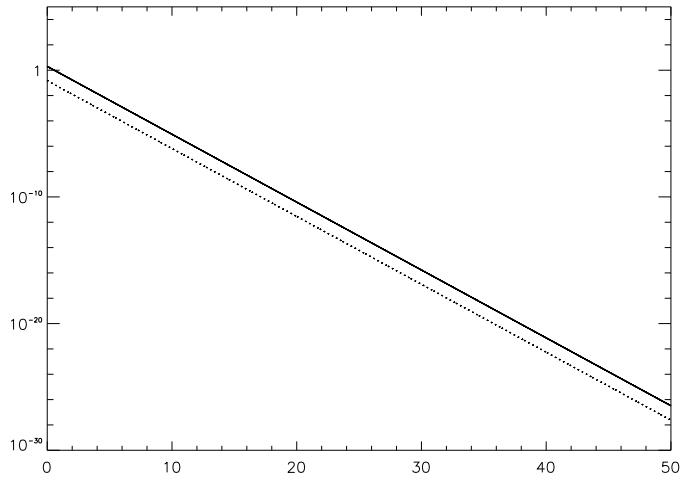


Figure A.1 Evolution of magnetic energy (solid line) for a fluid at rest and $e^{-2\times0.617\,t}$ (dotted line).

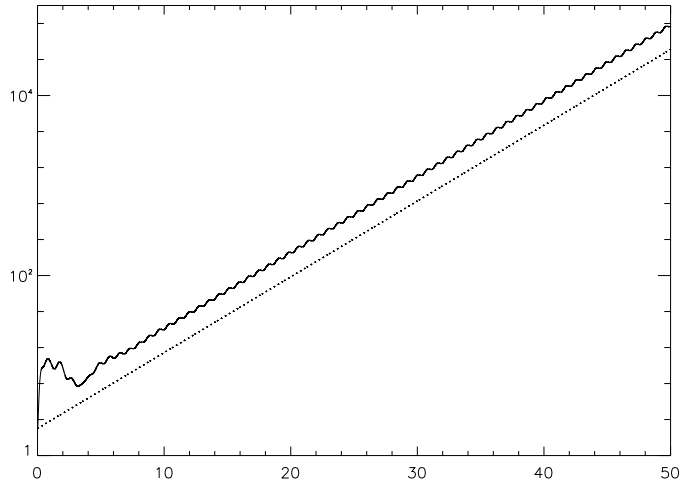


Figure A.2 Evolution of magnetic energy (solid line) of a divergence-free velocity field and $e^{2\times0.09696\,t}$ (dotted line).

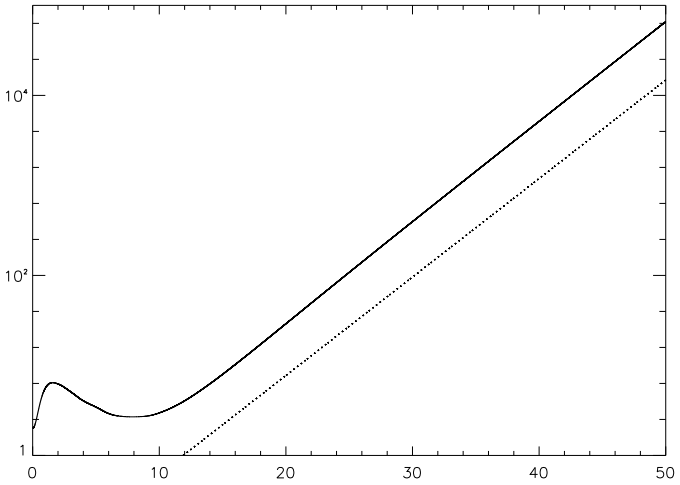


Figure A.3 Evolution of magnetic energy (solid line) of a non-divergence-free velocity field and $e^{2\times0.12599\,t}$ (dotted line).

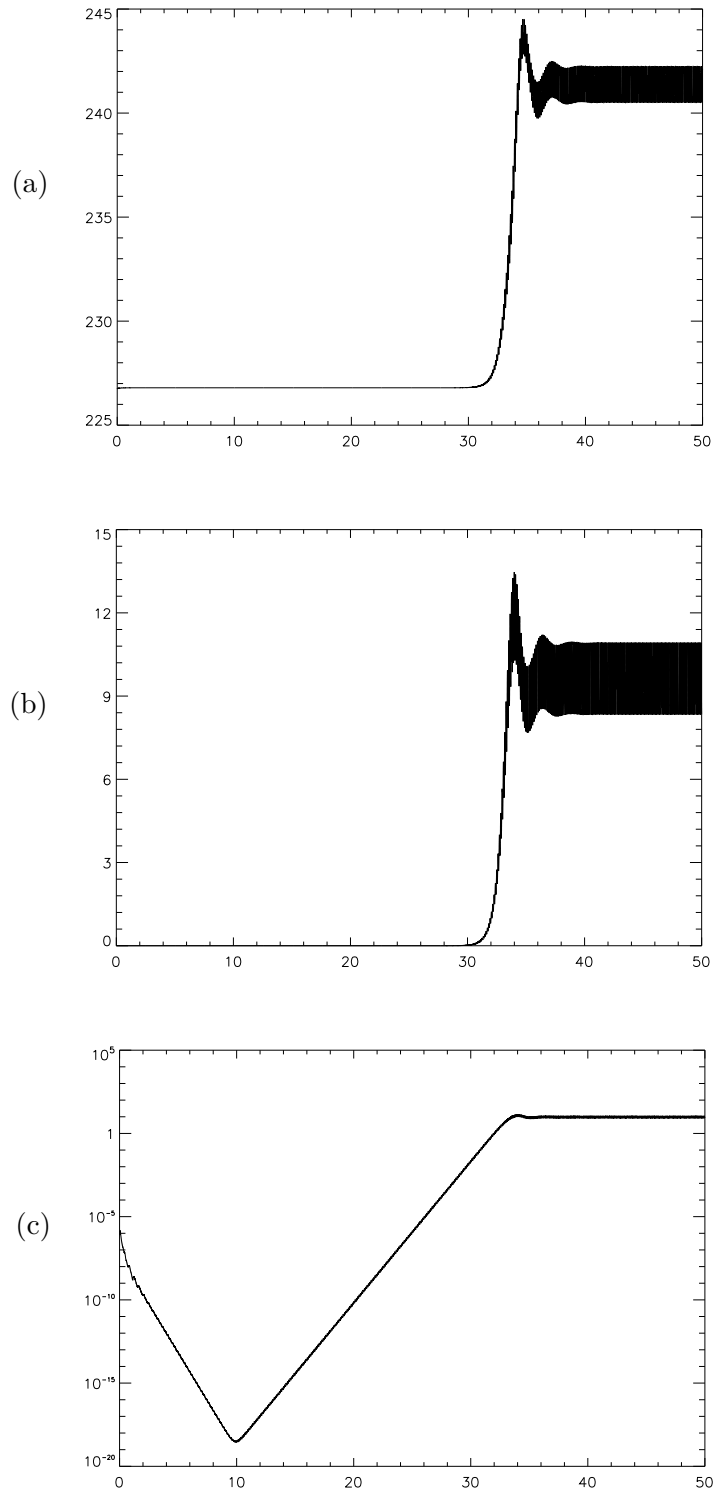


Figure A.4 The evolution of kinectic E_v and magnetic energy E_b of the MHD system for the parameters used by Podviginia [2006]. The vertical axis is in the logarithmic scale in plot (c).

A.2 Scaling of the parallel code

All computations in this work were performed on a cluster and a server. In this section we present the computer systems used, list their hardware specifications, briefly review the commands used to configure the software libraries, describe the code optimizations implemented for performance, and the tests executed to study the scalability of the parallel code.

The first computer used is the Navigator cluster from the Laboratory for Advanced Computing in Coimbra (LCA). Its specifications are mentioned below and they constitute only a small part of the available resources at LCA. A complete description of the cluster resources can be found on this link:
`https://www.uc.pt/lca/ClusterResources/Navigator/description`

We used two sets of computing nodes for the simulations in the cluster.

- 164 computing nodes (Fujitsu PRIMERGY CX400 S2)
  Processors: 2 × Intel Xeon E5-2697v2 (12-core) at 2.70 GHz
  Memory: 96 Gb ECC DDR3 LV RAM. ECC stands for error correction code, DDR3 is double data rate which is type 3 and LV refers to low voltage.
  Cache: L3, shared, 30 Mb
  Interconnect: HCA InfiniBand FDR 56 Gbps

- 28 computing nodes (Fujitsu PRIMERGY CX2550 M4)
  Processors: 2 × Intel Xeon Gold 6148 (20-core) at 2.40 Ghz
  Memory:

  – 21 nodes: 96 Gb DDR4-2666 R ECC RAM
  – 7 nodes: 384 Gb DDR4-2666 R ECC RAM

  Cache: L3, shared, 27.5 Mb
  Interconnect: EP Mellanox EDR ConnectX-4 100 Gbps

The second computer used is the cmup4 server from the Centro de Matemàtica at Universidade do Porto (CMUP). Some of its specifications are
Processors: 2 × AMD EPYC 7501 32-Core
Memory: 256 Gb RAM
Cache: L3, shared, 64 Mb

In both systems Intel compilers with MPI support were used: at LCA, version 19.0.5.281 was deployed, at CMUP, this included versions 2019.4.243 (2019), 2020.1.217 (2020) and 2021.2.0 (2021) - for each accompanying year, a recent version of the compiler was installed and used.

The libraries, FFTW 3.0 and P3DFFT 2.7.5, used in the above systems were configured and built using the below commands:

- FFTW

```
./configure CC=icc CFLAGS=-O3 F77=ifort FFLAGS=-O3 --enable-avx2
```

`CC`: This option is used to indicate the C compiler.

`CFLAGS`: This option allows for specifying flags for the C compiler.

`F77`: This option is used to specify the Fortran compiler.

`FFLAGS`: This option allows for specifying more flags for the Fortran compiler.

`--enable-avx2`: This extension enables the Single Instruction Multiple Data (SIMD) instruction set. FFTW checks only at run time if the compiler supports this particular feature. So, on a processor without the AVX support, a code can still be compiled and run.

- P3DFFT

```
./configure --enable-stride1 --enable-intel --enable-fftw
            FC=mpiifort CC=mpiicc
            CFLAGS=-O3 FCFLAGS=-O3
            --host=x86_64-unknown-linux-gnu
```

`--enable-stride1`: This option enables stride-1 data structures on the output. For more information about the stride feature, refer to section 2.6.2.2.

`--enable-intel`: This argument enables P3DFFT to be built for an intel compiler. Other options such as `-enable-gnu`, `-enable-pgi`, `-enableibm` are also available.

`--enable-fftw` : This argument enables P3DFFT to be used with FFTW.

`FC`: This option is used to specify the name of the Fortran compiler.

`FCFLAGS`: Using this option we can specify more flags for the Fortran compiler.

`--host`: This option allows for specifying the architecture of the machine.

The parallel code was tuned for performance before its execution according to the hardware in the systems. As an example the below command was executed at compilation on the Navigator cluster.

```
mpiifort -o mhd magneto.f90
         -mtune=core-avx2 -march=core-avx2
         -O3 -r8 -ipo -mcmodel=large
         -lp3dfft -lfftw3
         -assume byterecl -heap-arrays
```

The options used are:

`-march`: this flag instructs the compiler to generate a code for execution for the underlying architecture, CPU capabilities, instruction sets and quirks.

`-mtune`: this option allows for code optimization for specific processors. In our computations `core-avx2` was used.

`-O3`: It instructs the compiler to optimize the generated code for performance at the expense of compilation time and code size. This might result in an increased code size.

`-r8`: This tag doubles the default byte size for REAL and COMPLEX variables which are declared without an explicit byte size in the code.

`-ipo`: stands for Interprocedural Optimization. This option allows the compiler to analyze the code for specific optimization between files.

`-mcmodel`: enables the compiler to use a specific memory model to generate code and store data. The value `large` was used and so there was no restriction on the size of the code or data used.

`-lp3dfft`: This is a reference to the P3DFFT library for use in the code.

`-lfftw3`: This is a reference to the FFTW library.

`-assume byterecl`: the `assume` keyword tells the compiler to make certain assumptions. Here, `byterecl` allows to specify bytes or longwords (RECL units) for unformatted files without source code modification.

`-heap-arrays`: It enables the compiler to put automatic and temporary arrays created for computations on the heap instead of the stack.

An important aspect of a parallel code is to check its scalability. So we performed test runs in both systems. First we executed the code for 25 time steps for the resolution $512 \times 512 \times 97$ which is a 4×4 grid. We increased the processors in each run and measured the elapsed time in seconds – the time that elapses from the moment a parallel computation starts to the moment the last processor finishes execution. The data grid was decomposed into slabs (1D decomposition). This test also had the objective of finding the minimal number of processors needed for computation in a reasonable amount of time.

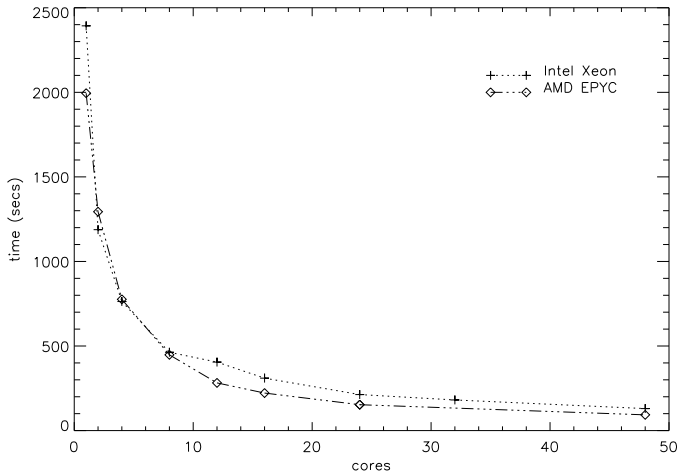
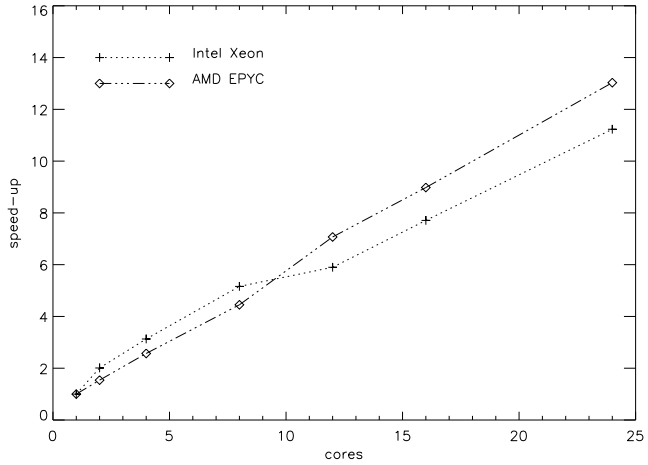


Figure A.5 Elapsed time (in seconds) of the parallel code for 25 time steps for $512 \times 512 \times 97$.



(a)

Cores	Intel	AMD
2	2.01	1.54
4	3.13	2.57
8	5.16	4.45
12	5.90	7.07
16	7.71	8.98
24	6.85	13.03

(b)

Figure A.6 Speed-ups for 25 time steps for the resolution $512 \times 512 \times 97$.

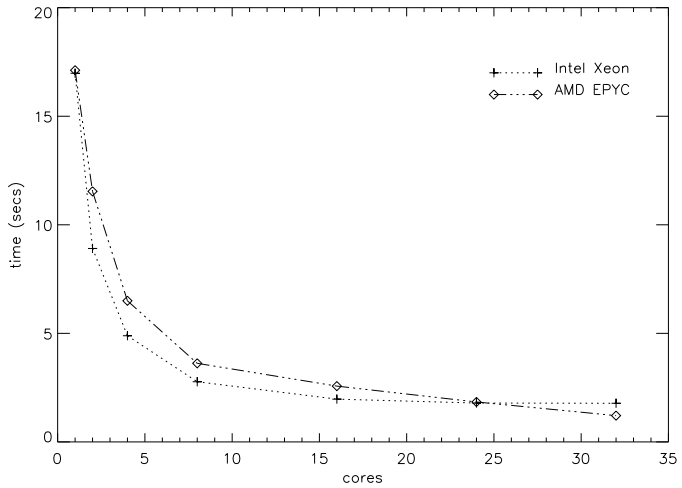
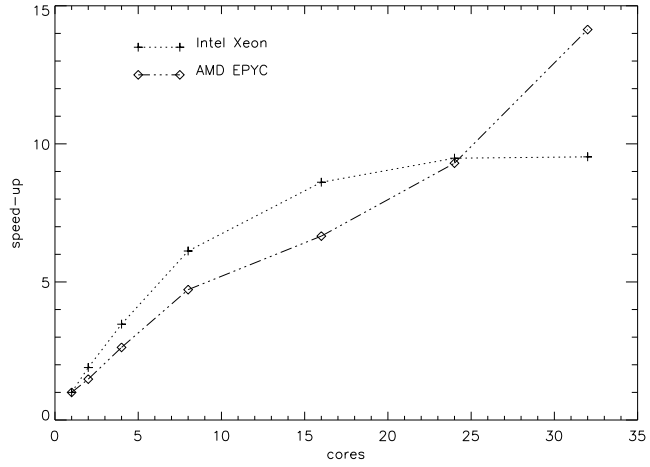


Figure A.7 Scaling of P3DFFT with forward and backward sine Fourier transforms for 10 iterations.

The test indicated that as the number of processors is increased, the computational time is reduced. We also determined that the number of processors needed for execution of the aforementioned resolution in a reasonable amount of time is either 24 or 32.

The speed-ups of the test runs were also calculated and the values are shown in Table A.6. We observe sublinear scaling in both systems. Communication between processors and load imbalance (uneven data distribution) are two probable reasons why scaling can be affected. With the use of P3DFFT in the parallel code, data distribution is minimized and therefore communication overhead could be the only reason behind this sublinear scaling. Anyway, this test indicated that more processors indeed can be a benefit in the computations.

The scaling of P3DFFT was also checked by performing just the Fourier transforms. A sample code was written to execute 10 iterations and in each iteration, a forward and



(a)

Cores	Intel	AMD
2	1.90	1.48
4	3.47	2.63
8	6.12	4.72
16	8.61	6.66
24	9.48	9.30
32	9.53	14.14

(b)

Figure A.8 Speed-ups of forward and backward sine transforms for 10 iterations

backward sine transform was computed. That is, a Fourier transform in X and Y, and a sine transform (DST-I) in Z. For each run we increased the number of processors and measured the elapsed time (see Figure A.7). Again, the resolution used was $512 \times 512 \times 97$ and slab decomposition was deployed.

The equations of hydromagnetic convection involve non-linear terms which are computed using forward and backward transforms in their respective spaces (Fourier and physical). Because of load imbalance, these operations may take a significant portion of the computing time and lead to performance degradation. The speed-ups of this test are tabulated in Table A.8. In both cases (integration of the full convective system and FFTs only), the computed values of speed-ups are close, enabling us to conclude that the scalability of the code is largely determined by the scalability of the FFTs.

A.3 Code

Below, we present the program code developed in this work.

A.3.1 magneto.f90

```
module params
    use p3dfft
    use iso_c_binding
    include 'mpif.h'
    include 'fftw3.f'

    public

    integer, parameter          :: rk = 8, writeState = 2000
    integer ierr, ndim, dims(2), nproc, procId

    integer nx, ny, nz, iproc, jproc, nxc, nyc, nzc, nxBox, nyBox
    complex(kind = rk)        :: ci

    real, parameter             :: tolerance = 1e-14
    real(kind = rk)             :: pi, alphax, alphay, P, Pm

    integer                     :: F_FLAG
    real(kind = rk)             :: TAYLOR = 675.0_rk
    real(kind = rk)             :: RAYLEIGH = 2300.0_rk

    character(*), parameter :: STATE_FOLDER = 'states/'
    character(*), parameter :: DATA_FOLDER = 'data/'
    character(*), parameter :: WISDOM_FOLDER = 'wisdom/'
    character(*), parameter :: DATA_FILE = 'mhd.dat'
    character(*), parameter :: ENERGY_SHELLS = &
    'energyShells.bin'
    character(*), parameter :: GRADS = 'gradients.dat'
    character(*), parameter :: COMM_FILE = 'comm.txt'

    character(*), parameter :: SEQ_DATA_FILE = 'fc'
    character(*), parameter :: STATE_FILE = 'datafile'
    ! FILE WITHOUT PERTURBATION
    character(*), parameter :: STEADY49_FILE = 'fc_wo'

    integer, parameter      :: WISDOM_FILE_INDEX = 100
```

```
character(*), parameter :: HARMONIC_001 = 'harmonic001.dat'
character(*), parameter :: HARMONIC_010 = 'harmonic010.dat'
character(*), parameter :: HARMONIC_100 = 'harmonic100.dat'
character(*), parameter :: HARMONIC_111 = 'harmonic111.dat'

character(*), parameter :: HARMONIC_002 = 'harmonic002.dat'
character(*), parameter :: HARMONIC_020 = 'harmonic020.dat'
character(*), parameter :: HARMONIC_200 = 'harmonic200.dat'
character(*), parameter :: HARMONIC_222 = 'harmonic222.dat'

character(*), parameter :: HARMONIC_112 = 'harmonic112.dat'
character(*), parameter :: HARMONIC_121 = 'harmonic121.dat'
character(*), parameter :: HARMONIC_211 = 'harmonic211.dat'

character(*), parameter :: HARMONIC_004 = 'harmonic004.dat'
character(*), parameter :: HARMONIC_040 = 'harmonic040.dat'
character(*), parameter :: HARMONIC_400 = 'harmonic400.dat'
character(*), parameter :: DECAY_FILE = 'decay.dat'
character(*), parameter :: PERTURB_FILE = 'perb.dat'

integer pr112, pr121, pr211, pr222, pr113, pr131, pr311, &
pr333, pr223, pr232, pr322, pr115, pr151, pr511

integer(kind = rk) sinePlan, cosinePlan, num, numx, numy, &
istride, idist

integer istart(3), iend(3), isize(3)
integer fstart(3), fend(3), fsize(3)

integer i, j, k, l, programType, kfil, neq, fileUnit

real(kind = rk), dimension(:,:,:), allocatable :: vp1, vp2, &
vp3, vcp1, vcp2, vcp3
complex(kind = rk), dimension(:,:,:), allocatable :: vf1, vf2, vf3

real(kind = rk), dimension(:,:,:), allocatable :: bp1, bp2, bp3
complex(kind = rk), dimension(:,:,:), allocatable :: bf1, bf2, bf3

real(kind = rk), dimension(:,:,:), allocatable :: tp
complex(kind = rk), dimension(:,:,:), allocatable :: tf

real(kind = rk), dimension(:), allocatable :: espv, espb, espt
```

```
real(kind = rk), dimension(:), allocatable :: kineticEnergyInShells, &
magneticEnergyInShells, temperatureInShells

real(kind = rk), dimension(:), allocatable :: env, enb, ent
real(kind = rk), dimension(:), allocatable :: vFullShells, bFullShells, &
tFullShells
complex(kind = rk) , dimension(:,:,:), allocatable :: svf1, svf2, svf3
complex(kind = rk) , dimension(:,:,:), allocatable :: sbf1, sbf2, sbf3
complex(kind = rk) , dimension(:,:,:), allocatable :: stfx

complex(kind = rk), dimension(:,:,:), allocatable :: tvf1, tvf2, tvf3, tyf
complex(kind = rk), dimension(:,:,:), allocatable :: tmvf1, tmvf2, tmvf3, tmyf
complex(kind = rk), dimension(:,:,:), allocatable :: tbf1, tbf2, tbf3
complex(kind = rk), dimension(:,:,:), allocatable :: tmbf1, tmbf2, tmbf3

complex(kind = rk), dimension(:,:,:), allocatable :: r3tmp

complex(kind = rk), dimension(:), allocatable :: arr
real(kind = rk), dimension(:), allocatable :: brr
integer, dimension(:), allocatable :: dx, dy, dzs, dc, dcy, dcz
integer(kind = rk), dimension(:), allocatable :: seedx

real(kind = rk), pointer :: buf(:), ret(:)
complex(p3dfft_type), pointer :: iftmp(:,:,:), oftmp(:,:,:)

complex(kind = rk), dimension(:), allocatable :: rvf1, rvf2, rvf3, rtf, &
msg, rbf1, rbf2, rbf3
complex(kind = rk), dimension(:), allocatable :: jindex, iindex

logical, parameter :: filter = .false.
integer status(MPI_STATUS_SIZE)
complex(kind = rk)  :: cvalue, fc
integer  :: numOfElements, source, dest, conjIndex, numProcs, tag

integer  :: sx = 96, sy = 96, sz = 49, ns, cc

contains

subroutine initWaveNumbers()
implicit none

do i = fstart(1),fend(1)
```

```
    if(i.eq.1.or.i.eq.nx/2+1)  dx(i) = 0
    if(i.gt.1.and.i.lt.nx/2+1) dx(i) = i-1
enddo

do j = fstart(2),fend(2)
    if(j.eq.1.or.j.eq.ny/2+1)  dy(j) = 0
    if(j.gt.1.and.j.lt.ny/2+1) dy(j) = j-1
    if(j.gt.ny/2+1)            dy(j) = j-1-ny
enddo

! For transform
do k = fstart(3),fend(3)
    if(k.eq.1.or.k.eq.nz)  dzs(k) = 0
    if(k.gt.1.and.k.lt.nz) dzs(k) = k-1
enddo

! For computation
dc(1) = 0
dc(nx/2+1) = nx/2
do i = 2,nx/2
    dc(i) = i-1
    dc(nx+2-i) = 1-i
enddo

dcy(1) = 0
dcy(ny/2+1) = ny/2
do i = 2,ny/2
    dcy(i) = i-1
    dcy(ny+2-i) = 1-i
enddo

do i = 1,nz
    dcz(i) = i-1
enddo
end

function divnorm(fc1,fc2,fc3)
implicit none

complex(kind = rk), dimension(:,:,:),  allocatable :: fc1,fc2,fc3
real(kind = rk)    :: divnorm
```

```
r3tmp = 0.
do i = fstart(3), fend(3)
    do j = fstart(2), fend(2)
        do k = fstart(1), fend(1)
        r3tmp(k,j,i) =  dc(i) * alphax * ci * &
        fc1(k,j,i) + dcy(j) * alphay * ci * &
        fc2(k,j,i) + dcz(k) * pi * fc3(k,j,i)
        enddo
    enddo
enddo

divnorm = maxval(abs(r3tmp))
end

function energy(fc1,fc2,fc3)

implicit none

complex(kind = rk), dimension(:,:,:), allocatable :: fc1,fc2,fc3
real(kind = rk)    :: a, energy

energy = 0.0_rk

do i = fstart(3), fend(3)
    do j = fstart(2),fend(2)
        do k = fstart(1), fend(1)
            a = 0.0_rk
            a = a + abs(fc1(k,j,i))**2 + abs(fc2(k,j,i))**2 + &
            abs(fc3(k,j,i))**2

            if(i.eq.1) a = 0.5_rk * a
            if(k.eq.1) a = 0.5_rk * a

            energy = energy + a
        enddo
    enddo
enddo
end

function perturbEnergy(fc1,fc2,fc3,st1,st2,st3,subspace)

implicit none
```

```
complex(kind = rk), dimension(:,:,:), allocatable :: fc1,fc2,fc3, &
    st1,st2,st3
real(kind = rk)    :: a, perturbEnergy
logical :: subspace
perturbEnergy = 0.0_rk

if(subspace) then
do i = 1, sx/2+1
    do j = 1,sy
        do k = 1, sz
        if((fstart(1).le.i.and.fend(1).ge.i).and. &
        (fstart(2).le.j.and.fend(2).ge.j)) then
        a = 0.0_rk
        a = abs(fc1(k,j,i) - st1(k,j,i))**2 + &
        abs(fc2(k,j,i) - st2(k,j,i))**2 + &
        abs(fc3(k,j,i) - st3(k,j,i))**2

        if(i.eq.1) a = 0.5_rk * a
        if(k.eq.1) a = 0.5_rk * a

        perturbEnergy = perturbEnergy + a
        endif
        enddo
    enddo
enddo
else
do i = fstart(3), fend(3)
    do j = fstart(2),fend(2)
        do k = fstart(1), fend(1)
        a = 0.0_rk
        a = abs(fc1(k,j,i) - st1(k,j,i))**2 + &
        abs(fc2(k,j,i) - st2(k,j,i))**2 + &
        abs(fc3(k,j,i) - st3(k,j,i))**2

        if(i.eq.1) a = 0.5_rk * a
        if(k.eq.1) a = 0.5_rk * a

        perturbEnergy = perturbEnergy + a
        enddo
    enddo
enddo
```

```
endif
end

subroutine computeEnergyInShells(fc1,fc2,fc3,esp)

implicit none

complex(kind = rk), dimension(:,:,:), allocatable :: fc1,fc2,fc3
real(kind = rk), dimension(:), allocatable :: esp
real(kind = rk)    :: a
integer            :: layer

esp = 0.0_rk

do i = fstart(3), fend(3)
    do j = fstart(2),fend(2)
        do k = fstart(1), fend(1)

            a  = dcz(k)**2 + dcy(j)**2 + dc(i)**2
            layer = sqrt(a)

            if(layer*layer.lt.a) layer = layer + 1

            a = abs(fc1(k,j,i))**2 + abs(fc2(k,j,i))**2 + &
            abs(fc3(k,j,i))**2

            if(i.eq.1) a = 0.5_rk * a
            if(k.eq.1) a = 0.5_rk * a

            if(layer.gt.0.and.layer.le.kfil) then
                esp(layer) = esp(layer) + a
            else
                esp(kfil+1) = esp(kfil+1) + a
            endif
        enddo
    enddo
enddo
end

subroutine computeGradient(fc1,fc2,fc3,grad2,grad4)

implicit none
```

```
    integer       :: i, j, k
    complex(kind = rk), dimension(:,:,:), allocatable :: fc1,fc2,fc3
    real(kind = rk)          :: a, nsq2, nsq4
    real(kind = rk), intent(out) :: grad2, grad4

    grad2 = 0.; grad4 = 0.

    do i = fstart(3), fend(3)
        do j = fstart(2),fend(2)
            do k = fstart(1), fend(1)

                a = abs(fc1(k,j,i))**2 + abs(fc2(k,j,i))**2 + &
                abs(fc3(k,j,i))**2

                if(i.eq.1) a = 0.5_rk * a
                if(k.eq.1) a = 0.5_rk * a

                nsq2 = (alphax**2 * dc(i)**2 + alphay**2 * &
                dcy(j)**2 + pi**2 * dcz(k)**2) * a
                nsq4 = (alphax**4 * dc(i)**4 + alphay**4 * &
                dcy(j)**4 + pi**4 * dcz(k)**4) * a

                grad2 = grad2 + nsq2
                grad4 = grad4 + nsq4
            enddo
        enddo
    enddo
    end
end

module temp
    use params

    ! Temporary arrays
    real(kind = rk), dimension(:,:,:), allocatable :: vfrtmp, vfctmp, &
    vfr, vfc
    complex(kind = rk), dimension(:,:,:), allocatable :: vftmp, ftmp1, &
    ftmp2, ftmp3
    real(kind = rk), dimension(:,:,:),  allocatable :: xtmp1, xtmp2, &
    xtmp3
```

```
    complex(kind = rk), dimension(:,:,:), allocatable :: vftmp1, &
    vftmp2, vftmp3
    complex(kind = rk), dimension(:,:,:), allocatable :: bftmp1, &
    bftmp2, bftmp3
    complex(kind = rk), dimension(:,:,:), allocatable :: tftmp
    complex(kind = rk), dimension(:,:,:), allocatable :: aftmp1, &
    aftmp2, aftmp3
end

module rungeKutta
    use params

    complex(kind = rk), dimension(:,:,:), allocatable :: v11, v12, v13, &
    v21, v22, v23
    complex(kind = rk), dimension(:,:,:), allocatable :: v31, v32, v33
    complex(kind = rk), dimension(:,:,:), allocatable :: k11, k12, k13, &
    k21, k22, k23
    complex(kind = rk), dimension(:,:,:), allocatable :: k31, k32, k33
    complex(kind = rk), dimension(:,:,:), allocatable :: t10, t20, t30

    real(kind = rk), dimension(:,:,:), allocatable :: cex
    real(kind = rk), dimension(:,:,:), allocatable :: c11, c21, c31
    real(kind = rk), dimension(:,:,:), allocatable :: c12, c22, c32
    real(kind = rk), dimension(:,:,:), allocatable :: c13, c23, c33
end

module wisdom
    use params

    contains
    subroutine writeChar(c, iunit)
        character c
        integer iunit
        write(iunit,321) c
    321  format(a,$)
    end

    subroutine exportWisdomToFile(iunit)
    integer iunit

    call dfftw_export_wisdom(writeChar, iunit)
    end
```

```
    subroutine readChar(ic, iunit)
        integer ic
        integer iunit
        character*256 buf
        save buf
        integer ibuf
        data ibuf/257/
        save ibuf
        if (ibuf .lt. 257) then
            ic = ichar(buf(ibuf:ibuf))
            ibuf = ibuf + 1
            return
        endif
        read(iunit,123,end = 666) buf
        ic = ichar(buf(1:1))
        ibuf = 2
        return
    666  ic = -1
        ibuf = 257
    123  format(a256)
    end

    subroutine importWisdomFromFile(isuccess, iunit)
    integer isuccess
    integer iunit

    call dfftw_import_wisdom(isuccess, readChar, iunit)
    end
end

program mhd
    use params
    use rungeKutta
    use temp
    use wisdom
    use ifport

    implicit none

    integer fstatus, idx1, idx2, idx3, res, sd
    integer(kind = rk)            :: itr, counter = 0
```

```
logical fexist, fileExist
real(kind = rk)               :: tau, now = 0.,startClock, stopClock
real(kind = rk)               :: energyInProcessor, &
totalEnergy, temperatureInProcessor, temperatureSum, &
magneticEnergyInProcessor, magneticEnergySum, &
vPerturbSum, bPerturbSum, tPerturbSum, &
v49Sum, b49Sum, t49Sum, vPerturbEnergy, bPerturbEnergy, &
tPerturbEnergy, divV, divB, divP

real(kind = rk)               :: sumGrad2, sumGrad4
complex(kind = rk)            :: pk
real(kind = rk)               :: maxvalue = 1d-15
character(len = 32)           :: filename, wisdomFile

call MPI_INIT (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD,nproc,ierr)
call MPI_COMM_RANK (MPI_COMM_WORLD,procId,ierr)

if(procId.eq.0) then
    open (unit = 3,file='stdinput.txt',status='old', &
        access='sequential',form='formatted', iostat=fstatus)
    if (fstatus.eq.0) then
    write(*, *) ' Reading from input file stdin'
    endif
    read (3,*) nx, ny, nz, ndim, nxBox, nyBox
    write (*,'(6(A,I4))') "procs=", nproc," nx=", nx, &
    " ny=", ny," nz=", nz, " ndim=", ndim

    numProcs = 0

    close(3)
    !print *, "Problem type to solve:"
    !print *, "0 - Read data(file from sequential code)"
    !print *, "1 - Read data(file from parallel code)"
    !print *, "2 - Generate hydrodynamic attractor"
    !read *, programType
    programType = 0

    ! Complete shells
    if(nz.eq.97) then
        if(nxBox.lt.2.or.nyBox.lt.2) then
            ns = 63
```

```
            else
                ns = 96
            endif
        else
            ns = 127
        endif
        cc = ns * ns
    endif

    call MPI_Bcast(nx,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
    call MPI_Bcast(ny,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
    call MPI_Bcast(nz,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
    call MPI_Bcast(ndim,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
    call MPI_Bcast(nxBox,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
    call MPI_Bcast(nyBox,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
    call MPI_Bcast(ns,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
    call MPI_Bcast(cc,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
    call MPI_Bcast(programType,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)

    ! nproc is divided into a iproc x jproc stencil
    if(ndim.eq.1) then
        dims(1) = 1
        dims(2) = nproc
    else if(ndim.eq.2) then
        inquire(file ='dims.txt', exist = fexist)
        if(fexist) then
            if(procId.eq.0) print *, 'Reading proc. grid from file dims'

            open(999,file = 'dims.txt')
            read(999,*) dims(1), dims(2)
            close(999)

            if(dims(1) * dims(2) .ne. nproc) then
                dims(2) = nproc/dims(1)
            endif
        else
            if(procId.eq.0) write(*,*) "Creating proc.grid"

            dims(1) = 0
            dims(2) = 0

            call MPI_Dims_create(nproc,2,dims,ierr)
```

```
        if(dims(1).gt.dims(2)) then
            dims(1) = dims(2)
            dims(2) = nproc/dims(1)
        endif
    endif
endif

iproc = dims(1); jproc = dims(2)

nxc = nx; nyc = ny; nzc = nz

call p3dfft_setup(dims,nx,ny,nz,MPI_COMM_WORLD,nxc,nyc,nzc,.true.)

call p3dfft_get_dims(istart,iend,isize,1)
call p3dfft_get_dims(fstart,fend,fsize,2)

print *, 'istart: ', procId, istart(1), istart(2), istart(3)
print *, 'iend: ', procId, iend(1), iend(2), iend(3)
print *, 'fstart: ', procId, fstart(1), fstart(2), fstart(3)
print *, 'fend: ', procId, fend(1), fend(2), fend(3)
print *, '================================='

! Prandtl numbers
P = 1.0_rk;
Pm = 8.0_rk

tau = 1e-3_rk
pi = 4.0_rk * atan(1.0_rk)
ci = cmplx(0.0_rk, 1.0_rk, kind = rk)
kfil = ns
itr = 1
tag = 0
neq = 7*2*(nx/2+1)*ny*nz

! Box sizes
alphax = pi/(nxBox * sqrt(2.0_rk))
alphay = pi/(nyBox * sqrt(2.0_rk))

if(procId .eq. 0) then
    print *,'Using processor grid: ',iproc,' x ',jproc
```

```
open(1, access = 'sequential', form = 'formatted', &
file = trim(adjustl(DATA_FOLDER))//DATA_FILE, &
status = 'unknown', recl = 256, position = 'append')

open(2, access = 'sequential', form = 'formatted', &
file = trim(adjustl(DATA_FOLDER))//GRADS, &
status = 'unknown', recl = 256, position = 'append')

open(31, access = 'sequential', form = 'formatted', &
file = trim(adjustl(DATA_FOLDER))//HARMONIC_100, &
status = 'unknown', recl = 512, position = 'append')

open(32, access = 'sequential', form = 'formatted', &
file = trim(adjustl(DATA_FOLDER))//HARMONIC_010, &
status = 'unknown', recl = 512, position = 'append')

open(33, access = 'sequential', form = 'formatted', &
file = trim(adjustl(DATA_FOLDER))//HARMONIC_001, &
status = 'unknown', recl = 512, position = 'append')

open(34, access = 'sequential', form = 'formatted', &
file = trim(adjustl(DATA_FOLDER))//HARMONIC_111, &
status = 'unknown', recl = 512, position = 'append')

open(41, access = 'sequential', form = 'formatted', &
file = trim(adjustl(DATA_FOLDER))//HARMONIC_200, &
status = 'unknown', recl = 512, position = 'append')

open(42, access = 'sequential', form = 'formatted', &
file = trim(adjustl(DATA_FOLDER))//HARMONIC_020, &
status = 'unknown', recl = 512, position = 'append')

open(43, access = 'sequential', form = 'formatted', &
file = trim(adjustl(DATA_FOLDER))//HARMONIC_002, &
status = 'unknown', recl = 512, position = 'append')

open(44, access = 'sequential', form = 'formatted', &
file = trim(adjustl(DATA_FOLDER))//HARMONIC_222, &
status = 'unknown', recl = 512, position = 'append')

open(45, access = 'sequential', form = 'formatted', &
file = trim(adjustl(DATA_FOLDER))//HARMONIC_211, &
```

```
    status = 'unknown', recl = 512, position = 'append')

    open(46, access = 'sequential', form = 'formatted', &
    file = trim(adjustl(DATA_FOLDER))//HARMONIC_121, &
    status = 'unknown', recl = 512, position = 'append')

    open(47, access = 'sequential', form = 'formatted', &
    file = trim(adjustl(DATA_FOLDER))//HARMONIC_112, &
    status = 'unknown', recl = 512, position = 'append')

    open(48, access = 'sequential', form = 'formatted', &
    file = trim(adjustl(DATA_FOLDER))//HARMONIC_400, &
    status = 'unknown', recl = 512, position = 'append')

    open(49, access = 'sequential', form = 'formatted', &
    file = trim(adjustl(DATA_FOLDER))//HARMONIC_040, &
    status = 'unknown', recl = 512, position = 'append')

    open(50, access = 'sequential', form = 'formatted', &
    file = trim(adjustl(DATA_FOLDER))//HARMONIC_004, &
    status = 'unknown', recl = 512, position = 'append')

    open(51, access = 'sequential', form = 'formatted', &
    file = trim(adjustl(DATA_FOLDER))//DECAY_FILE, &
    status = 'unknown', recl = 512, position = 'append')

    open(52, access = 'sequential', form = 'formatted', &
    file = trim(adjustl(DATA_FOLDER))//PERTURB_FILE, &
    status = 'unknown', recl = 512, position = 'append')

    open(37, access = 'stream', form = 'unformatted', &
    convert='little_endian', &
    file = trim(adjustl(DATA_FOLDER))//ENERGY_SHELLS, &
    status = 'unknown', position = 'append')
endif

! Allocate
call allocateArrays()

call initWaveNumbers()
call initFFTW()
```

```
if(nproc.gt.1) then
    call getProcessorIndices()

    call findProcessor(2,1,1,pr211) ! (1,0,0)
    call findProcessor(1,2,1,pr121) ! (0,1,0)
    call findProcessor(1,1,2,pr112) ! (0,0,1)
    call findProcessor(2,2,2,pr222) ! (1,1,1)

    ! Even harmonics
    call findProcessor(3,1,1,pr311) ! (2,0,0)
    call findProcessor(1,3,1,pr131) ! (0,2,0)
    call findProcessor(1,1,3,pr113) ! (0,0,2)
    call findProcessor(3,3,3,pr333) ! (2,2,2)

    call findProcessor(3,2,2,pr322) ! (2,1,1)
    call findProcessor(2,3,2,pr232) ! (1,2,1)
    call findProcessor(2,2,3,pr223) ! (1,1,2)

    call findProcessor(5,1,1,pr511) ! (4,0,0)
    call findProcessor(1,5,1,pr151) ! (0,4,0)
    call findProcessor(1,1,5,pr115) ! (0,0,4)
else
    pr112 = 0; pr121 = 0; pr211 = 0; pr222 = 0; pr113 = 0; pr131 = 0;
    pr311 = 0; pr333 = 0; pr223 = 0; pr232 = 0; pr322 = 0; pr115 = 0;
    pr151 = 0; pr511 = 0;
endif

call random_seed(size=sd)
allocate(seedx(sd))

if(programType.eq.0) then
    call readSeqDatafromFile()
    call readSteadyData()

    now = 0.
    tau = 1e-3_rk
else if(programType.eq.1) then
    call readSeqDatafromFile()
    call readSteadyData()
    tau = 5e-4_rk
else if(programType.eq.2) then
    call random_seed(get=seedx)
```

```
    do i = 1,sd
     seedx(i) = int(seedx(i))*(procId+1)
    enddo

    call random_seed(put=seedx)

    ! Random field
    call initRandomField()
    call initializeMagneticField()

    call cleanFourierCoefficients2()
    call minusGrad(vf1,vf2,vf3)
    call minusGrad(bf1,bf2,bf3)
    call cleanFourierCoefficients2()

    write(filename,'(F8.3)') now
    filename = 'state_'//trim(adjustl(filename))
    call storeDataForSeqCode(filename)
endif

call printDataToScreen()

if(procId .eq. 0) then
    open(77, access = 'sequential', form = 'formatted', &
    file = COMM_FILE, status = 'unknown', recl = 256, &
    position = 'append')

    write(77,*)'******************************************************'
    write(77,'("Nx    =  ",I4  )') nx
    write(77,'("Ny    =  ",I4  )') ny
    write(77,'("Nz    =  ",I4  )') nz
    write(77,'("Boxes =  ",I4  )') nxBox
    write(77,'("kind  =  ",I4  )') rk
    write(77,'("Ta    =  ",F8.2)') Taylor
    write(77,'("Ra    =  ",F8.2)') Rayleigh
    write(77,'("P     =  ",F8.2)') P
    write(77,'("Pm    =  ",F8.2)') Pm
    write(77,'("alph1 =  ",F10.7)') alphax
    write(77,'("alph2 =  ",F10.7)') alphay
    write(77,'("tnow  =  ",G12.5)') now
    write(77,'("tau0  =  ",G25.15)') tau
```

```
        write(77,'("E_v   =  ",G25.15E3)') totalEnergy
        write(77,'("E_b   =  ",G25.15E3)') magneticEnergySum
        write(77,'("E_t   =  ",G25.15E3)') temperatureSum
    endif

    if(programType.eq.0.or.programType.eq.2) call storeData()
    call initCex(tau)

    startClock = MPI_Wtime( )

    ! Time stepping computation
    1 continue
        counter = counter + 1

        call etdrk3(now,tau)
        call printDataToScreen()
        call storeData()

        inquire(file="./STOP", exist = fexist)
        if(fexist) then
            go to 12
        endif

        if(counter.eq.writeState) then
            counter = 0
            write(filename,'(F8.3)') now
            filename = 'state_'//trim(adjustl(filename))
            call storeDataForSeqCode(filename)
            filename = ''
        endif
    goto 1

    12 if(procId.eq.0) then
        print *, 'Computation is stopped at', now
        close(1); close(2)

        ! File close - Odd harmonics
        close(31);close(32);close(33);close(34)

        ! File close - Energy shells
        close(37)
```

```
    ! File close - Even harmonics
    close(41);close(42);close(43);close(44)
    close(45);close(46);close(47);close(48)
    close(49);close(50);close(51);close(52)
endif

stopClock = MPI_Wtime( ) - startClock
write(*,*)'Elapsed time : ', stopClock

write(filename,'(F8.3)') now
filename = 'state_'//trim(adjustl(filename))
call storeDataForSeqCode(filename)

call dfftw_destroy_plan(sinePlan)
!call dfftw_destroy_plan(cosinePlan)
call dfftw_forget_wisdom()

if(procId.eq.0) then
    deallocate(tvf1,tvf2,tvf3,tyf,tbf1,tbf2,tbf3, &
        kineticEnergyInShells,magneticEnergyInShells, &
        temperatureInShells)
endif

call MPI_BARRIER(MPI_COMM_WORLD,ierr)
call p3dfft_clean

deallocate(vp1,vp2,vp3,vf1,vf2,vf3,dx,dy,dzs,dc,dcy,dcz,&
vfr,vfc,vfrtmp,vfctmp,vftmp,r3tmp,arr,vcp1,vcp2,vcp3,&
bp1,bp2,bp3,bf1,bf2,bf3,xtmp1,xtmp2,xtmp3,ftmp1,&
ftmp2,ftmp3,k11,k12,k13,k21,k22,k23,k31,k32,k33,&
cex,aftmp1,aftmp2,aftmp3,c11,c21,c31,vftmp1,&
vftmp2,vftmp3,v11,v12,v13,v21,v22,v23,v31, &
v32,v33,tftmp,t10,t20,t30,tp,tf,c13,c23,c33,brr, &
jindex,iindex,iftmp,oftmp,rvf1,rvf2,rvf3,rtf,msg,&
rbf1,rbf2,rbf3,espv,espb,espt,seedx)

call MPI_FINALIZE(ierr)

contains

subroutine computeEnergyInFullShells()
```

```
implicit none

real(kind = rk)    :: a, av, ab, at
integer            :: layer

env=0.;enb=0.;ent=0.

do i = fstart(3), fend(3)
do j = fstart(2),fend(2)
do k = fstart(1), fend(1)

a = dc(i)**2 + dcy(j)**2 + dcz(k)**2

! Incomplete shell
if(a.gt.cc)then
    av = REAL(vf1(k,j,i))**2 + IMAG(vf1(k,j,i))**2 + &
    REAL(vf2(k,j,i))**2 + IMAG(vf2(k,j,i))**2 + &
    REAL(vf3(k,j,i))**2 + IMAG(vf3(k,j,i))**2
    ab = REAL(bf1(k,j,i))**2 + IMAG(bf1(k,j,i))**2 + &
    REAL(bf2(k,j,i))**2 + IMAG(bf2(k,j,i))**2 + &
    REAL(bf3(k,j,i))**2 + IMAG(bf3(k,j,i))**2
    at = REAL(tf(k,j,i))**2 + IMAG(tf(k,j,i))**2

    if(i.eq.1)then
        av = 0.5_rk * av
        ab = 0.5_rk * ab
        at = 0.5_rk * at
    endif
    if(k.eq.1)then
        av = 0.5_rk * av
        ab = 0.5_rk * ab
        at = 0.5_rk * at
    endif

    env(ns+1) = env(ns+1) + av
    enb(ns+1) = enb(ns+1) + ab
    ent(ns+1) = ent(ns+1) + at
else
    ! Complete shells
    layer = sqrt(a)
    if(layer*layer.lt.a) layer = layer + 1
```

```
    av = REAL(vf1(k,j,i))**2 + IMAG(vf1(k,j,i))**2 + &
    REAL(vf2(k,j,i))**2 + IMAG(vf2(k,j,i))**2 + &
    REAL(vf3(k,j,i))**2 + IMAG(vf3(k,j,i))**2
    ab = REAL(bf1(k,j,i))**2 + IMAG(bf1(k,j,i))**2 + &
    REAL(bf2(k,j,i))**2 + IMAG(bf2(k,j,i))**2 + &
    REAL(bf3(k,j,i))**2 + IMAG(bf3(k,j,i))**2
    at = REAL(tf(k,j,i))**2 + IMAG(tf(k,j,i))**2

    if(i.eq.1)then
        av = 0.5_rk * av
        ab = 0.5_rk * ab
        at = 0.5_rk * at
    endif
    if(k.eq.1)then
        av = 0.5_rk * av
        ab = 0.5_rk * ab
        at = 0.5_rk * at
    endif

    if(layer.gt.0.and.layer.le.ns) then
        env(layer) = env(layer) + av
        enb(layer) = enb(layer) + ab
        ent(layer) = ent(layer) + at
    endif
endif
enddo
enddo
enddo

end

subroutine readSeqDatafromFile()
implicit none

complex(kind = rk)  :: vf(nx/2+1,ny,nz,3),bf(nx/2+1,ny,nz,3),&
tmf(nx/2+1,ny,nz)
integer  :: fstart2, fstart3, fend2, fend3, vk, vj, vi

if(procId.eq.0) then

    if(programType.eq.0) then
    open(36, access = 'direct',form = 'unformatted', &
```

```
    convert='little_endian', &
    file = trim(adjustl(DATA_FOLDER))//SEQ_DATA_FILE, &
    status = 'old', recl = sizeof(pi) * neq)
else
    open(36, access = 'direct',form = 'unformatted', &
    convert='little_endian', &
    file = trim(adjustl(STATE_FOLDER))//STATE_FILE, &
    status = 'old', recl = sizeof(pi) * neq)
endif

print *, 'Reading direct data ....'
read(36,rec = 1) now, tau, RAYLEIGH, TAYLOR, P, Pm, alphax, alphay
read(36,rec = 2)  vf, bf,tmf
close(36)

tmvf1(:,:,:) = vf(:,:,:,1); tmvf2(:,:,:) = vf(:,:,:,2);
tmvf3(:,:,:) = vf(:,:,:,3)
tmbf1(:,:,:) = bf(:,:,:,1); tmbf2(:,:,:) = bf(:,:,:,2);
tmbf3(:,:,:) = bf(:,:,:,3)
tmyf(:,:,:) = tmf

! Stride format
do i = 1, nx/2+1
    do j = 1, ny
        do k = 1, nz
            tvf1(k,j,i) = tmvf1(i,j,k); tvf2(k,j,i) = tmvf2(i,j,k);
            tvf3(k,j,i) = tmvf3(i,j,k)
            tbf1(k,j,i) = tmbf1(i,j,k); tbf2(k,j,i) = tmbf2(i,j,k);
            tbf3(k,j,i) = tmbf3(i,j,k)
            tyf(k,j,i) = tmyf(i,j,k)
        enddo
    enddo
enddo

if(nproc.eq.1) then
    vf1(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
    tvf1(1:nz,fstart(2):fend(2),fstart(3):fend(3))
    vf2(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
    tvf2(1:nz,fstart(2):fend(2),fstart(3):fend(3))
    vf3(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
    tvf3(1:nz,fstart(2):fend(2),fstart(3):fend(3))
    bf1(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
```

```
        tbf1(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        bf2(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tbf2(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        bf3(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tbf3(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        tf(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tyf(1:nz,fstart(2):fend(2),fstart(3):fend(3))
    else
        do l = 1, nproc
            fstart2 = REAL(jindex(l)); fend2 = IMAG(jindex(l))
            fstart3 = REAL(iindex(l)); fend3 = IMAG(iindex(l))

            if(l.eq.1) then
                vf1(1:nz,fstart2:fend2,fstart3:fend3) = &
                tvf1(1:nz,fstart2:fend2,fstart3:fend3)
                vf2(1:nz,fstart2:fend2,fstart3:fend3) = &
                tvf2(1:nz,fstart2:fend2,fstart3:fend3)
                vf3(1:nz,fstart2:fend2,fstart3:fend3) = &
                tvf3(1:nz,fstart2:fend2,fstart3:fend3)
                bf1(1:nz,fstart2:fend2,fstart3:fend3) = &
                tbf1(1:nz,fstart2:fend2,fstart3:fend3)
                bf2(1:nz,fstart2:fend2,fstart3:fend3) = &
                tbf2(1:nz,fstart2:fend2,fstart3:fend3)
                bf3(1:nz,fstart2:fend2,fstart3:fend3) = &
                tbf3(1:nz,fstart2:fend2,fstart3:fend3)
                tf(1:nz,fstart2:fend2,fstart3:fend3) = &
                tyf(1:nz,fstart2:fend2,fstart3:fend3)
            else
                do i = fstart3, fend3
                    do j = fstart2, fend2
                        msg(1:nz) = tvf1(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
                        l-1,1,MPI_COMM_WORLD,ierr)

                        msg(1:nz) = tvf2(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
                        l-1,2,MPI_COMM_WORLD,ierr)

                        msg(1:nz) = tvf3(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
```

```
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
                        l-1,3,MPI_COMM_WORLD,ierr)

                        msg(1:nz) = tbf1(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
                        l-1,4,MPI_COMM_WORLD,ierr)

                        msg(1:nz) = tbf2(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
                        l-1,5,MPI_COMM_WORLD,ierr)

                        msg(1:nz) = tbf3(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
                        l-1,6,MPI_COMM_WORLD,ierr)

                        msg(1:nz) = tyf(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
                        l-1,7,MPI_COMM_WORLD,ierr)
                    enddo
                enddo
            endif
        enddo
    endif
else
    do i = fstart(3), fend(3)
        do j = fstart(2), fend(2)
            call MPI_RECV(rvf1,nz+1,MPI_DOUBLE_COMPLEX,0,1, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rvf1(nz+1)); vi = IMAG(rvf1(nz+1))
            vf1(1:nz,vj,vi) = rvf1(1:nz)

            call MPI_RECV(rvf2,nz+1,MPI_DOUBLE_COMPLEX,0,2, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rvf2(nz+1)); vi = IMAG(rvf2(nz+1))
            vf2(1:nz,vj,vi) = rvf2(1:nz)

            call MPI_RECV(rvf3,nz+1,MPI_DOUBLE_COMPLEX,0,3, &
            MPI_COMM_WORLD,status,ierr)
```

```
            vj = REAL(rvf3(nz+1)); vi = IMAG(rvf3(nz+1))
            vf3(1:nz,vj,vi) = rvf3(1:nz)

            call MPI_RECV(rbf1,nz+1,MPI_DOUBLE_COMPLEX,0,4, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rbf1(nz+1)); vi = IMAG(rbf1(nz+1))
            bf1(1:nz,vj,vi) = rbf1(1:nz)

            call MPI_RECV(rbf2,nz+1,MPI_DOUBLE_COMPLEX,0,5, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rbf2(nz+1)); vi = IMAG(rbf2(nz+1))
            bf2(1:nz,vj,vi) = rbf2(1:nz)

            call MPI_RECV(rbf3,nz+1,MPI_DOUBLE_COMPLEX,0,6, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rbf3(nz+1)); vi = IMAG(rbf3(nz+1))
            bf3(1:nz,vj,vi) = rbf3(1:nz)

            call MPI_RECV(rtf,nz+1,MPI_DOUBLE_COMPLEX,0,7, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rtf(nz+1)); vi = IMAG(rtf(nz+1))
            tf(1:nz,vj,vi) = rtf(1:nz)
        enddo
    enddo
endif

call MPI_Bcast(now,1,MPI_DOUBLE_PRECISION,0, &
MPI_COMM_WORLD,ierr)
call MPI_Bcast(tau,1,MPI_DOUBLE_PRECISION,0, &
MPI_COMM_WORLD,ierr)
call MPI_Bcast(RAYLEIGH,1,MPI_DOUBLE_PRECISION,0, &
MPI_COMM_WORLD,ierr)
call MPI_Bcast(TAYLOR,1,MPI_DOUBLE_PRECISION,0, &
MPI_COMM_WORLD,ierr)
call MPI_Bcast(P,1,MPI_DOUBLE_PRECISION,0, &
MPI_COMM_WORLD,ierr)
call MPI_Bcast(Pm,1,MPI_DOUBLE_PRECISION,0, &
MPI_COMM_WORLD,ierr)
call MPI_Bcast(alphax,1,MPI_DOUBLE_PRECISION,0, &
MPI_COMM_WORLD,ierr)
call MPI_Bcast(alphay,1,MPI_DOUBLE_PRECISION,0, &
MPI_COMM_WORLD,ierr)
```

```
end

subroutine initializeMagneticField()
implicit none

real(kind = rk)               :: bk

call random_number(bp1)
call random_number(bp2)
call random_number(bp3)

call forwardTransform(bp1,bf1,1)
call forwardTransform(bp2,bf2,1)
call forwardTransform(bp3,bf3,0)

do i = fstart(3),fend(3)
    do j = fstart(2),fend(2)
        do k = fstart(1),fend(1)
            bk = sqrt(real(dc(i)**2 + dcy(j)**2 + dcz(k)**2))
            bf1(k,j,i) = bf1(k,j,i)/2.0**bk
            bf2(k,j,i) = bf2(k,j,i)/2.0**bk
            bf3(k,j,i) = bf3(k,j,i)/2.0**bk
        enddo
    enddo
enddo

call cleanFourierCoefficients2()
call minusGrad(bf1,bf2,bf3)
call cleanFourierCoefficients2()

magneticEnergyInProcessor = energy(bf1,bf2,bf3)
call MPI_Allreduce(magneticEnergyInProcessor,magneticEnergySum,1, &
MPI_DOUBLE_PRECISION,MPI_SUM,MPI_COMM_WORLD,ierr)

! 10e1
bf1 = 5.0_rk*bf1/sqrt(magneticEnergySum)
bf2 = 5.0_rk*bf2/sqrt(magneticEnergySum)
bf3 = 5.0_rk*bf3/sqrt(magneticEnergySum)
end

subroutine initRandomField()
implicit none
```

```
real(kind = rk)               :: vk

call random_number(vp1)
call random_number(vp2)
call random_number(vp3)

call forwardTransform(vp1,vf1,1)
call forwardTransform(vp2,vf2,1)
call forwardTransform(vp3,vf3,0)

call random_number(tp)
call forwardTransform(tp,tf,0)

call cleanFourierCoefficients2()
call minusGrad(vf1,vf2,vf3)

do i = fstart(3),fend(3)
    do j = fstart(2),fend(2)
        do k = fstart(1),fend(1)
            vk = sqrt(real(dc(i)**2 + dcy(j)**2 + dcz(k)**2))
            vf1(k,j,i) = vf1(k,j,i)/2.0**vk
            vf2(k,j,i) = vf2(k,j,i)/2.0**vk
            vf3(k,j,i) = vf3(k,j,i)/2.0**vk
            tf(k,j,i) = tf(k,j,i)/2.0**vk
        enddo
    enddo
enddo

call cleanFourierCoefficients2()
call minusGrad(vf1,vf2,vf3)
call cleanFourierCoefficients2()

energyInProcessor = energy(vf1,vf2,vf3)
call MPI_Allreduce(energyInProcessor,totalEnergy,1, &
MPI_DOUBLE_PRECISION,MPI_SUM,MPI_COMM_WORLD,ierr)

! 10e2
vf1 = 10.0_rk*vf1/sqrt(totalEnergy)
vf2 = 10.0_rk*vf2/sqrt(totalEnergy)
vf3 = 10.0_rk*vf3/sqrt(totalEnergy)
```

```
aftmp2 = 0.; aftmp3 = 0.
temperatureInProcessor = energy(tf,aftmp2,aftmp3)
call MPI_Allreduce(temperatureInProcessor,temperatureSum, &
1,MPI_DOUBLE_PRECISION,MPI_SUM,MPI_COMM_WORLD,ierr)

tf = 0.1_rk * tf/sqrt(temperatureSum)
end

subroutine readSteadyData()
implicit none

complex(kind = rk)  :: svf(nx/2+1,ny,nz,3), &
    sbf(nx/2+1,ny,nz,3),stf(nx/2+1,ny,nz)
complex(kind = rk)  ::stvf1(nz,ny,nx/2+1), &
    stbf1(nz,ny,nx/2+1),stf1(nz,ny,nx/2+1), &
    stvf2(nz,ny,nx/2+1),stbf2(nz,ny,nx/2+1),&
    stvf3(nz,ny,nx/2+1),stbf3(nz,ny,nx/2+1)

integer  :: fstart2, fstart3, fend2, fend3, vk, vj, vi

if(procId.eq.0) then

    open(36, access = 'direct',form = 'unformatted', convert='little_endian', &
    file = trim(adjustl(DATA_FOLDER))//STEADY49_FILE, &
    status = 'old', recl = sizeof(pi)*7*2*(nx/2+1)*ny*nz)

    print *, 'Reading direct data(steady) ....'
    read(36,rec = 1) snow, stau, sRAYLEIGH, sTAYLOR, sP, sPm, &
    salphax, salphay
    read(36,rec = 2)  svf, sbf,stf
    close(36)

    stvf1 = 0; stvf2 = 0; stvf3 = 0;
    stbf1 = 0; stbf2 = 0; stbf3 = 0;
    stf1 = 0
    ! Stride format
    do i = 1, nx/2+1
        do j = 1, ny
            do k = 1, nz
                stvf1(k,j,i) = svf(i,j,k,1); stvf2(k,j,i) = svf(i,j,k,2);
                stvf3(k,j,i) = svf(i,j,k,3)
                stbf1(k,j,i) = sbf(i,j,k,1); stbf2(k,j,i) = sbf(i,j,k,2);
```

```
            stbf3(k,j,i) = sbf(i,j,k,3)

            stf1(k,j,i) = stf(i,j,k)
        enddo
    enddo
enddo

svf1 = 0; svf2 = 0; svf3 = 0
sbf1 = 0; sbf2 = 0; sbf3 = 0
stfx = 0

if(nproc.eq.1) then
    svf1 = stvf1; svf2 = stvf2; svf3 = stvf3
    sbf1 = stbf1; sbf2 = stbf2; sbf3 = stbf3
    stfx = stf1
else
    do l = 1, nproc
        fstart2 = REAL(jindex(l));
        fend2 = IMAG(jindex(l))
        fstart3 = REAL(iindex(l));
        fend3 = IMAG(iindex(l))

        if(l.eq.1) then
            svf1(1:nz,fstart2:fend2,fstart3:fend3) = &
            stvf1(1:nz,fstart2:fend2,fstart3:fend3)
            svf2(1:nz,fstart2:fend2,fstart3:fend3) = &
            stvf2(1:nz,fstart2:fend2,fstart3:fend3)
            svf3(1:nz,fstart2:fend2,fstart3:fend3) = &
            stvf3(1:nz,fstart2:fend2,fstart3:fend3)
            sbf1(1:nz,fstart2:fend2,fstart3:fend3) = &
            stbf1(1:nz,fstart2:fend2,fstart3:fend3)
            sbf2(1:nz,fstart2:fend2,fstart3:fend3) = &
            stbf2(1:nz,fstart2:fend2,fstart3:fend3)
            sbf3(1:nz,fstart2:fend2,fstart3:fend3) = &
            stbf3(1:nz,fstart2:fend2,fstart3:fend3)
            stfx(1:nz,fstart2:fend2,fstart3:fend3) = &
            stf1(1:nz,fstart2:fend2,fstart3:fend3)
        else
            do i = fstart3, fend3
                do j = fstart2, fend2
                    msg(1:nz) = stvf1(1:nz,j,i);
                    msg(nz+1) = cmplx(j,i)
```

```
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,1, &
                        MPI_COMM_WORLD,ierr)

                        msg(1:nz) = stvf2(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,2, &
                        MPI_COMM_WORLD,ierr)

                        msg(1:nz) = stvf3(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,3, &
                        MPI_COMM_WORLD,ierr)

                        msg(1:nz) = stbf1(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,4, &
                        MPI_COMM_WORLD,ierr)

                        msg(1:nz) = stbf2(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,5, &
                        MPI_COMM_WORLD,ierr)

                        msg(1:nz) = stbf3(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,6, &
                        MPI_COMM_WORLD,ierr)

                        msg(1:nz) = stf1(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,7, &
                        MPI_COMM_WORLD,ierr)
                    enddo
                enddo
            endif
        enddo
    endif
else
    do i = fstart(3), fend(3)
        do j = fstart(2), fend(2)
            call MPI_RECV(rvf1,nz+1,MPI_DOUBLE_COMPLEX,0,1, &
            MPI_COMM_WORLD,status,ierr)
```

```
              vj = REAL(rvf1(nz+1)); vi = IMAG(rvf1(nz+1))
              svf1(1:nz,vj,vi) = rvf1(1:nz)

              call MPI_RECV(rvf2,nz+1,MPI_DOUBLE_COMPLEX,0,2, &
              MPI_COMM_WORLD,status,ierr)
              vj = REAL(rvf2(nz+1)); vi = IMAG(rvf2(nz+1))
              svf2(1:nz,vj,vi) = rvf2(1:nz)

              call MPI_RECV(rvf3,nz+1,MPI_DOUBLE_COMPLEX,0,3, &
              MPI_COMM_WORLD,status,ierr)
              vj = REAL(rvf3(nz+1)); vi = IMAG(rvf3(nz+1))
              svf3(1:nz,vj,vi) = rvf3(1:nz)

              call MPI_RECV(rbf1,nz+1,MPI_DOUBLE_COMPLEX,0,4, &
              MPI_COMM_WORLD,status,ierr)
              vj = REAL(rbf1(nz+1)); vi = IMAG(rbf1(nz+1))
              sbf1(1:nz,vj,vi) = rbf1(1:nz)

              call MPI_RECV(rbf2,nz+1,MPI_DOUBLE_COMPLEX,0,5, &
              MPI_COMM_WORLD,status,ierr)
              vj = REAL(rbf2(nz+1)); vi = IMAG(rbf2(nz+1))
              sbf2(1:nz,vj,vi) = rbf2(1:nz)

              call MPI_RECV(rbf3,nz+1,MPI_DOUBLE_COMPLEX,0,6, &
              MPI_COMM_WORLD,status,ierr)
              vj = REAL(rbf3(nz+1)); vi = IMAG(rbf3(nz+1))
              sbf3(1:nz,vj,vi) = rbf3(1:nz)

            call MPI_RECV(rtf,nz+1,MPI_DOUBLE_COMPLEX,0,7, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rtf(nz+1)); vi = IMAG(rtf(nz+1))
            stfx(1:nz,vj,vi) = rtf(1:nz)
          enddo
      enddo
  endif
  end

  subroutine readDatafromFile()
  implicit none

  real(kind = rk)    :: rc1,ic1,rc2,ic2,rc3,ic3,rc4,ic4,rc5,ic5,rc6,ic6,rc7,ic7
  integer  :: fstart2, fstart3, fend2, fend3, vk, vj, vi
```

```
if(procId.eq.0) then

    open(25, access = 'direct',form = 'unformatted', &
    file = trim(adjustl(STATE_FOLDER))//STATE_FILE, &
    status = 'old', convert = 'little_endian', &
    recl = sizeof(pi)* neq)

    print *, 'Reading....'

    read(25, rec = 1) now, tau, RAYLEIGH, TAYLOR, P, Pm, alphax, alphay
    read(25, rec = 2) tvf1,tvf2,tvf3,tbf1,tbf2,tbf3,tyf
    close(25)

    if(nproc.eq.1) then
        vf1(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tvf1(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        vf2(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tvf2(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        vf3(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tvf3(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        bf1(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tbf1(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        bf2(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tbf2(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        bf3(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tbf3(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        tf(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tyf(1:nz,fstart(2):fend(2),fstart(3):fend(3))
    else
        do l = 1, nproc
            fstart2 = REAL(jindex(l));
            fend2 = IMAG(jindex(l))
            fstart3 = REAL(iindex(l));
            fend3 = IMAG(iindex(l))

            if(l.eq.1) then
                vf1(1:nz,fstart2:fend2,fstart3:fend3) = &
                tvf1(1:nz,fstart2:fend2,fstart3:fend3)
                vf2(1:nz,fstart2:fend2,fstart3:fend3) = &
                tvf2(1:nz,fstart2:fend2,fstart3:fend3)
                vf3(1:nz,fstart2:fend2,fstart3:fend3) = &
```

```
        tvf3(1:nz,fstart2:fend2,fstart3:fend3)
        bf1(1:nz,fstart2:fend2,fstart3:fend3) = &
        tbf1(1:nz,fstart2:fend2,fstart3:fend3)
        bf2(1:nz,fstart2:fend2,fstart3:fend3) = &
        tbf2(1:nz,fstart2:fend2,fstart3:fend3)
        bf3(1:nz,fstart2:fend2,fstart3:fend3) = &
        tbf3(1:nz,fstart2:fend2,fstart3:fend3)
        tf(1:nz,fstart2:fend2,fstart3:fend3) = &
        tyf(1:nz,fstart2:fend2,fstart3:fend3)
    else
        do i = fstart3, fend3
            do j = fstart2, fend2
                msg(1:nz) = tvf1(1:nz,j,i);
                msg(nz+1) = cmplx(j,i)
                call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,1, &
                MPI_COMM_WORLD,ierr)

                msg(1:nz) = tvf2(1:nz,j,i);
                msg(nz+1) = cmplx(j,i)
                call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,2, &
                MPI_COMM_WORLD,ierr)

                msg(1:nz) = tvf3(1:nz,j,i);
                msg(nz+1) = cmplx(j,i)
                call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,3, &
                MPI_COMM_WORLD,ierr)

                msg(1:nz) = tbf1(1:nz,j,i);
                msg(nz+1) = cmplx(j,i)
                call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,4, &
                MPI_COMM_WORLD,ierr)

                msg(1:nz) = tbf2(1:nz,j,i);
                msg(nz+1) = cmplx(j,i)
                call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,5, &
                MPI_COMM_WORLD,ierr)

                msg(1:nz) = tbf3(1:nz,j,i);
                msg(nz+1) = cmplx(j,i)
                call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,6, &
                MPI_COMM_WORLD,ierr)
```

```
                        msg(1:nz) = tyf(1:nz,j,i);
                        msg(nz+1) = cmplx(j,i)
                        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,l-1,7, &
                        MPI_COMM_WORLD,ierr)
                    enddo
                enddo
            endif
        enddo
    endif
else
    do i = fstart(3), fend(3)
        do j = fstart(2), fend(2)
            call MPI_RECV(rvf1,nz+1,MPI_DOUBLE_COMPLEX,0,1, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rvf1(nz+1)); vi = IMAG(rvf1(nz+1))
            vf1(1:nz,vj,vi) = rvf1(1:nz)

            call MPI_RECV(rvf2,nz+1,MPI_DOUBLE_COMPLEX,0,2, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rvf2(nz+1)); vi = IMAG(rvf2(nz+1))
            vf2(1:nz,vj,vi) = rvf2(1:nz)

            call MPI_RECV(rvf3,nz+1,MPI_DOUBLE_COMPLEX,0,3, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rvf3(nz+1)); vi = IMAG(rvf3(nz+1))
            vf3(1:nz,vj,vi) = rvf3(1:nz)

            call MPI_RECV(rbf1,nz+1,MPI_DOUBLE_COMPLEX,0,4, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rbf1(nz+1)); vi = IMAG(rbf1(nz+1))
            bf1(1:nz,vj,vi) = rbf1(1:nz)

            call MPI_RECV(rbf2,nz+1,MPI_DOUBLE_COMPLEX,0,5, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rbf2(nz+1)); vi = IMAG(rbf2(nz+1))
            bf2(1:nz,vj,vi) = rbf2(1:nz)

            call MPI_RECV(rbf3,nz+1,MPI_DOUBLE_COMPLEX,0,6, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rbf3(nz+1)); vi = IMAG(rbf3(nz+1))
            bf3(1:nz,vj,vi) = rbf3(1:nz)
```

```
            call MPI_RECV(rtf,nz+1,MPI_DOUBLE_COMPLEX,0,7,&
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rtf(nz+1)); vi = IMAG(rtf(nz+1))
            tf(1:nz,vj,vi) = rtf(1:nz)
        enddo
    enddo
endif

call MPI_Bcast(now,1,MPI_DOUBLE_PRECISION,0, &
    MPI_COMM_WORLD,ierr)
call MPI_Bcast(tau,1,MPI_DOUBLE_PRECISION,0, &
    MPI_COMM_WORLD,ierr)
call MPI_Bcast(RAYLEIGH,1,MPI_DOUBLE_PRECISION,0, &
    MPI_COMM_WORLD,ierr)
call MPI_Bcast(TAYLOR,1,MPI_DOUBLE_PRECISION,0, &
    MPI_COMM_WORLD,ierr)
call MPI_Bcast(P,1,MPI_DOUBLE_PRECISION,0, &
    MPI_COMM_WORLD,ierr)
call MPI_Bcast(Pm,1,MPI_DOUBLE_PRECISION,0, &
    MPI_COMM_WORLD,ierr)
call MPI_Bcast(alphax,1,MPI_DOUBLE_PRECISION,0, &
    MPI_COMM_WORLD,ierr)
call MPI_Bcast(alphay,1,MPI_DOUBLE_PRECISION,0, &
    MPI_COMM_WORLD,ierr)
end

subroutine gatherEnergyInShells()
implicit none

real(kind = rk), dimension(:), allocatable :: vshells, bshells, tshells

! Shell energy
call computeEnergyInShells(vf1,vf2,vf3,espv)
call computeEnergyInShells(bf1,bf2,bf3,espb)
aftmp2 = 0.; aftmp3 = 0.
call computeEnergyInShells(tf,aftmp2,aftmp3,espt)

if(procId.eq.0) then

allocate(vshells(1:kfil+1)); allocate(bshells(1:kfil+1));
allocate(tshells(1:kfil+1))
kineticEnergyInShells = 0.; magneticEnergyInShells = 0.;
```

```
temperatureInShells = 0.

!
kineticEnergyInShells = espv
magneticEnergyInShells = espb
temperatureInShells = espt

if(nproc.gt.1) then
    do j = 2, nproc
        call MPI_RECV(vshells,kfil+1,MPI_DOUBLE_PRECISION,&
        MPI_ANY_SOURCE,20,MPI_COMM_WORLD,status,ierr)

        call MPI_RECV(bshells,kfil+1,MPI_DOUBLE_PRECISION,&
        MPI_ANY_SOURCE,21,MPI_COMM_WORLD,status,ierr)

        call MPI_RECV(tshells,kfil+1,MPI_DOUBLE_PRECISION,&
        MPI_ANY_SOURCE,22,MPI_COMM_WORLD,status,ierr)

        kineticEnergyInShells = kineticEnergyInShells + vshells
        magneticEnergyInShells = magneticEnergyInShells + bshells
        temperatureInShells = temperatureInShells + tshells
    enddo
endif
else
call MPI_SEND(espv,kfil+1,MPI_DOUBLE_PRECISION,0,20,MPI_COMM_WORLD,ierr)
call MPI_SEND(espb,kfil+1,MPI_DOUBLE_PRECISION,0,21,MPI_COMM_WORLD,ierr)
call MPI_SEND(espt,kfil+1,MPI_DOUBLE_PRECISION,0,22,MPI_COMM_WORLD,ierr)
endif
end

subroutine gatherEnergyInFullShells()
implicit none

real(kind = rk), dimension(:), allocatable :: vshells, bshells, tshells
real(kind = rk)  :: zv1, zb1, zt1, zv2, zb2, zt2, etav, &
etab, etat, maxEnergy, ev, eb, et
real ::  maxval

! Shell energy
call computeEnergyInFullShells()

if(procId.eq.0) then
```

```
allocate(vshells(1:ns+1)); allocate(bshells(1:ns+1));
allocate(tshells(1:ns+1))
vFullShells = 0.; bFullShells = 0.; tFullShells = 0.

vFullShells = env
bFullShells = enb
tFullShells = ent

if(nproc.gt.1) then
do j = 2, nproc
    vshells = 0.; bshells = 0.; tshells = 0.
    call MPI_RECV(vshells,ns+1,MPI_DOUBLE_PRECISION, &
    MPI_ANY_SOURCE,120,MPI_COMM_WORLD,status,ierr)

    call MPI_RECV(bshells,ns+1,MPI_DOUBLE_PRECISION, &
    MPI_ANY_SOURCE,121,MPI_COMM_WORLD,status,ierr)

    call MPI_RECV(tshells,ns+1,MPI_DOUBLE_PRECISION, &
    MPI_ANY_SOURCE,122,MPI_COMM_WORLD,status,ierr)

    vFullShells = vFullShells + vshells
    bFullShells = bFullShells + bshells
    tFullShells = tFullShells + tshells
enddo
endif

deallocate(vshells,bshells,tshells)

ev = maxval(vFullShells(1:3))
zv1 = log10(ev/vFullShells(ns))
maxEnergy = maxval(vFullShells(ns-5:ns+1))
zv2 = log10(ev/maxEnergy)
etav = vFullShells(ns+1)

eb = maxval(bFullShells(1:3))
zb1 = log10(eb/bFullShells(ns))
maxEnergy = maxval(bFullShells(ns-5:ns+1))
zb2 = log10(eb/maxEnergy)
etab = bFullShells(ns+1)

et = maxval(tFullShells(1:3))
```

```
    zt1 = log10(et/tFullShells(ns))
    maxEnergy = maxval(tFullShells(ns-5:ns+1))
    zt2 = log10(et/maxEnergy)
    etat = tFullShells(ns+1)

    write(51,'(G17.10,9G25.15E3)')now,zv1,zv2,etav,zb1,zb2,etab, &
    zt1,zt2,etat

else
    call MPI_SEND(env,ns+1,MPI_DOUBLE_PRECISION,0,120,MPI_COMM_WORLD,ierr)
    call MPI_SEND(enb,ns+1,MPI_DOUBLE_PRECISION,0,121,MPI_COMM_WORLD,ierr)
    call MPI_SEND(ent,ns+1,MPI_DOUBLE_PRECISION,0,122,MPI_COMM_WORLD,ierr)
endif
end

subroutine storeDataForSeqCode(filename)
implicit none

complex(kind = rk)  :: vf(nx/2+1,ny,nz,3), &
    bf(nx/2+1,ny,nz,3),tmf(nx/2+1,ny,nz)
integer fstart2, fstart3, fend2, fend3, vi, vj, vk, r
character(len = 32)   :: filename, backupCmdArg

do i = 1, nx/2+1
    do j = 1, ny

        if(nproc.gt.1) then
            do l = 1, nproc
            fstart2 = REAL(jindex(l));
            fend2 = IMAG(jindex(l))
            fstart3 = REAL(iindex(l));
            fend3 = IMAG(iindex(l))

            if(j.ge.fstart2.and.j.le.fend2.and.&
            i.ge.fstart3.and.i.le.fend3) then
                source = l-1
                go to 26
            endif
            enddo
        else
            source = procId
        endif
```

```
26 if(procId.eq.0) then
    if(source.eq.0) then
        vf(i,j,1:nz,1) = vf1(1:nz,j,i)
        vf(i,j,1:nz,2) = vf2(1:nz,j,i)
        vf(i,j,1:nz,3) = vf3(1:nz,j,i)

        bf(i,j,1:nz,1) = bf1(1:nz,j,i)
        bf(i,j,1:nz,2) = bf2(1:nz,j,i)
        bf(i,j,1:nz,3) = bf3(1:nz,j,i)

        tmf(i,j,1:nz) = tf(1:nz,j,i)
    else
    call MPI_RECV(rvf1,nz+1,MPI_DOUBLE_COMPLEX, &
    MPI_ANY_SOURCE,1,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rvf1(nz+1)); vi = IMAG(rvf1(nz+1))
    vf(vi,vj,1:nz,1) = rvf1(1:nz)

    call MPI_RECV(rvf2,nz+1,MPI_DOUBLE_COMPLEX, &
    MPI_ANY_SOURCE,2,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rvf2(nz+1)); vi = IMAG(rvf2(nz+1))
    vf(vi,vj,1:nz,2) = rvf2(1:nz)

    call MPI_RECV(rvf3,nz+1,MPI_DOUBLE_COMPLEX, &
    MPI_ANY_SOURCE,3,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rvf3(nz+1)); vi = IMAG(rvf3(nz+1))
    vf(vi,vj,1:nz,3) = rvf3(1:nz)

    call MPI_RECV(rbf1,nz+1,MPI_DOUBLE_COMPLEX, &
    MPI_ANY_SOURCE,4,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rbf1(nz+1)); vi = IMAG(rbf1(nz+1))
    bf(vi,vj,1:nz,1) = rbf1(1:nz)

    call MPI_RECV(rbf2,nz+1,MPI_DOUBLE_COMPLEX, &
    MPI_ANY_SOURCE,5,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rbf2(nz+1)); vi = IMAG(rbf2(nz+1))
    bf(vi,vj,1:nz,2) = rbf2(1:nz)

    call MPI_RECV(rbf3,nz+1,MPI_DOUBLE_COMPLEX, &
    MPI_ANY_SOURCE,6,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rbf3(nz+1)); vi = IMAG(rbf3(nz+1))
    bf(vi,vj,1:nz,3) = rbf3(1:nz)
```

```
            call MPI_RECV(rtf,nz+1,MPI_DOUBLE_COMPLEX, &
            MPI_ANY_SOURCE,7,MPI_COMM_WORLD,status,ierr)
            vj = REAL(rtf(nz+1)); vi = IMAG(rtf(nz+1))
            tmf(vi,vj,1:nz) = rtf(1:nz)
            endif
        else if(procId.eq.source) then
            msg(1:nz) = vf1(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,1, &
            MPI_COMM_WORLD,ierr)
            msg(1:nz) = vf2(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,2, &
            MPI_COMM_WORLD,ierr)
            msg(1:nz) = vf3(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,3, &
            MPI_COMM_WORLD,ierr)

            msg(1:nz) = bf1(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,4, &
            MPI_COMM_WORLD,ierr)
            msg(1:nz) = bf2(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,5, &
            MPI_COMM_WORLD,ierr)
            msg(1:nz) = bf3(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,6, &
            MPI_COMM_WORLD,ierr)

            msg(1:nz) = tf(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,7, &
            MPI_COMM_WORLD,ierr)
        endif
    enddo
enddo

if(procId.eq.0) then
```

```
    filename = trim(adjustl(STATE_FOLDER))//filename
    open(26, access = 'direct',form = 'unformatted', &
    file = filename, status = 'unknown', &
    recl = sizeof(pi) * neq)
    write(26,rec = 1) now, tau, RAYLEIGH, TAYLOR, P, Pm, alphax, alphay
    write(26,rec = 2) vf,bf,tmf
    close(26)
    print *, 'Data stored in file for sequential code - ', filename

    write(backupCmdArg,'(F8.3)') now
    r = SYSTEM('./backup.sh ' // backupCmdArg)
endif

call MPI_BARRIER(MPI_COMM_WORLD,ierr)
end

subroutine storeDataToFile(filename)
implicit none

character(len = 32)   :: filename, backupCmdArg
integer fstart2, fstart3, fend2, fend3, vi, vj, vk, r

do i = 1, nx/2+1
    do j = 1, ny

        if(nproc.gt.1) then
            do l = 1, nproc
                fstart2 = REAL(jindex(l));
                fend2 = IMAG(jindex(l))
                fstart3 = REAL(iindex(l));
                fend3 = IMAG(iindex(l))

                if(j.ge.fstart2.and.j.le.fend2.and.&
                i.ge.fstart3.and.i.le.fend3) then
                    source = l-1
                    go to 24
                endif
            enddo
        else
            source = procId
        endif
```

```
24 if(procId.eq.0) then
    if(source.eq.0) then
        tvf1(1:nz,j,i) = vf1(1:nz,j,i)
        tvf2(1:nz,j,i) = vf2(1:nz,j,i)
        tvf3(1:nz,j,i) = vf3(1:nz,j,i)

        tbf1(1:nz,j,i) = bf1(1:nz,j,i)
        tbf2(1:nz,j,i) = bf2(1:nz,j,i)
        tbf3(1:nz,j,i) = bf3(1:nz,j,i)

        tyf(1:nz,j,i) = tf(1:nz,j,i)
    else
    call MPI_RECV(rvf1,nz+1,MPI_DOUBLE_COMPLEX, &
    MPI_ANY_SOURCE,1,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rvf1(nz+1)); vi = IMAG(rvf1(nz+1))
    tvf1(1:nz,vj,vi) = rvf1(1:nz)

    call MPI_RECV(rvf2,nz+1,MPI_DOUBLE_COMPLEX, &
    MPI_ANY_SOURCE,2,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rvf2(nz+1)); vi = IMAG(rvf2(nz+1))
    tvf2(1:nz,vj,vi) = rvf2(1:nz)

    call MPI_RECV(rvf3,nz+1,MPI_DOUBLE_COMPLEX, &
    MPI_ANY_SOURCE,3,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rvf3(nz+1)); vi = IMAG(rvf3(nz+1))
    tvf3(1:nz,vj,vi) = rvf3(1:nz)

    call MPI_RECV(rbf1,nz+1,MPI_DOUBLE_COMPLEX, &
    MPI_ANY_SOURCE,4,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rbf1(nz+1)); vi = IMAG(rbf1(nz+1))
    tbf1(1:nz,vj,vi) = rbf1(1:nz)

    call MPI_RECV(rbf2,nz+1,MPI_DOUBLE_COMPLEX, &
    MPI_ANY_SOURCE,5,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rbf2(nz+1)); vi = IMAG(rbf2(nz+1))
    tbf2(1:nz,vj,vi) = rbf2(1:nz)

    call MPI_RECV(rbf3,nz+1,MPI_DOUBLE_COMPLEX, &
    MPI_ANY_SOURCE,6,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rbf3(nz+1)); vi = IMAG(rbf3(nz+1))
    tbf3(1:nz,vj,vi) = rbf3(1:nz)
```

```
            call MPI_RECV(rtf,nz+1,MPI_DOUBLE_COMPLEX, &
            MPI_ANY_SOURCE,7,MPI_COMM_WORLD,status,ierr)
            vj = REAL(rtf(nz+1)); vi = IMAG(rtf(nz+1))
            tyf(1:nz,vj,vi) = rtf(1:nz)
            endif
        else if(procId.eq.source) then

        msg(1:nz) = vf1(1:nz,j,i);
        msg(nz+1) = cmplx(j,i)
        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,&
        0,1,MPI_COMM_WORLD,ierr)
        msg(1:nz) = vf2(1:nz,j,i); msg(nz+1) = cmplx(j,i)
        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,&
        0,2,MPI_COMM_WORLD,ierr)
        msg(1:nz) = vf3(1:nz,j,i); msg(nz+1) = cmplx(j,i)
        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,&
        0,3,MPI_COMM_WORLD,ierr)

        msg(1:nz) = bf1(1:nz,j,i); msg(nz+1) = cmplx(j,i)
        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,&
        0,4,MPI_COMM_WORLD,ierr)
        msg(1:nz) = bf2(1:nz,j,i); msg(nz+1) = cmplx(j,i)
        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,&
        0,5,MPI_COMM_WORLD,ierr)
        msg(1:nz) = bf3(1:nz,j,i); msg(nz+1) = cmplx(j,i)
        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,6, &
            MPI_COMM_WORLD,ierr)

        msg(1:nz) = tf(1:nz,j,i); msg(nz+1) = cmplx(j,i)
        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,&
        0,7,MPI_COMM_WORLD,ierr)

        endif
    enddo
enddo

if(procId.eq.0) then
    filename = trim(adjustl(STATE_FOLDER))//filename
    open(24, access = 'direct',form = 'unformatted', &
    file = filename, status = 'unknown',recl = sizeof(pi) * neq)
    write(24,rec = 1) now, tau, RAYLEIGH, TAYLOR, P, Pm, alphax, alphay
    write(24,rec = 2) tvf1,tvf2,tvf3,tbf1,tbf2,tbf3,tyf
```

```
        close(24)
        print *, 'Data stored in file - ', filename

        write(backupCmdArg,'(F8.3)') now
        r = SYSTEM('./backup.sh ' // backupCmdArg)
    endif

    call MPI_BARRIER(MPI_COMM_WORLD,ierr)
    end

    subroutine getProcessorIndices()
    implicit none

    complex(kind = rk) ::  temp

    if(procId.eq.0) then
        3 call MPI_RECV(temp,1,MPI_DOUBLE_COMPLEX, &
        MPI_ANY_SOURCE,MPI_ANY_TAG,MPI_COMM_WORLD,status,ierr)

        jindex(status(MPI_SOURCE)+1) = temp
        numProcs = numProcs + 1

        if(numProcs.lt.nproc-1) then
            go to 3
        else
            jindex(1) = cmplx(fstart(2), fend(2))
            numProcs = 0
        endif
    else
        temp = cmplx(fstart(2), fend(2))
        call MPI_SEND(temp,1,MPI_DOUBLE_COMPLEX,0,procId,MPI_COMM_WORLD,ierr)
    endif

    call MPI_Bcast(jindex,nproc,MPI_DOUBLE_COMPLEX,0,MPI_COMM_WORLD,ierr)

    !
    if(procId.eq.0) then
        4   call MPI_RECV(temp,1,MPI_DOUBLE_COMPLEX,&
        MPI_ANY_SOURCE,MPI_ANY_TAG,MPI_COMM_WORLD,status,ierr)

        iindex(status(MPI_SOURCE)+1) = temp
        numProcs = numProcs + 1
```

```
    if(numProcs.lt.nproc-1) then
        go to 4
    else
        iindex(1) = cmplx(fstart(3),fend(3))
        numProcs = 0
    endif
else
    temp = cmplx(fstart(3),fend(3))
    call MPI_SEND(temp,1,MPI_DOUBLE_COMPLEX,0,procId,MPI_COMM_WORLD,ierr)
endif

call MPI_Bcast(iindex,nproc,MPI_DOUBLE_COMPLEX,0,MPI_COMM_WORLD,ierr)
end

subroutine printDataToScreen()
implicit none

real(kind = rk) :: grad2, grad4

energyInProcessor = energy(vf1,vf2,vf3)

call MPI_Reduce(energyInProcessor,totalEnergy,1,&
MPI_DOUBLE_PRECISION,MPI_SUM,0,MPI_COMM_WORLD,ierr)

aftmp2 = 0.; aftmp3 = 0.
temperatureInProcessor = energy(tf,aftmp2,aftmp3)
call MPI_Reduce(temperatureInProcessor,temperatureSum,1, &
MPI_DOUBLE_PRECISION,MPI_SUM,0,MPI_COMM_WORLD,ierr)

magneticEnergyInProcessor = energy(bf1,bf2,bf3)
call MPI_Reduce(magneticEnergyInProcessor, &
magneticEnergySum,1,MPI_DOUBLE_PRECISION,MPI_SUM,0, &
MPI_COMM_WORLD,ierr)

vPerturbEnergy = perturbEnergy(vf1,vf2,vf3,svf1,svf2, &
    svf3,.FALSE.)
call MPI_Reduce(vPerturbEnergy,vPerturbSum,1,&
MPI_DOUBLE_PRECISION,MPI_SUM,0,MPI_COMM_WORLD,ierr)

bPerturbEnergy = perturbEnergy(bf1,bf2,bf3,sbf1,sbf2, &
    sbf3,.FALSE.)
```

```
call MPI_Reduce(bPerturbEnergy,bPerturbSum,1,&
MPI_DOUBLE_PRECISION,MPI_SUM,0,MPI_COMM_WORLD,ierr)

tPerturbEnergy = perturbEnergy(tf,aftmp2,aftmp3,stfx, &
    aftmp2,aftmp3,.FALSE.)
call MPI_Reduce(tPerturbEnergy,tPerturbSum,1, &
MPI_DOUBLE_PRECISION,MPI_SUM,0,MPI_COMM_WORLD,ierr)

! PERTURBATION IN STEADY SUBSPACE
vPerturbEnergy = perturbEnergy(vf1,vf2,vf3,svf1,svf2, &
    svf3,.TRUE.)
call MPI_Reduce(vPerturbEnergy,v49Sum,1,&
MPI_DOUBLE_PRECISION,MPI_SUM,0,MPI_COMM_WORLD,ierr)

bPerturbEnergy = perturbEnergy(bf1,bf2,bf3,sbf1,sbf2, &
    sbf3,.TRUE.)
call MPI_Reduce(bPerturbEnergy,b49Sum,1,&
MPI_DOUBLE_PRECISION,MPI_SUM,0,MPI_COMM_WORLD,ierr)

tPerturbEnergy = perturbEnergy(tf,aftmp2,aftmp3,stfx, &
    aftmp2,aftmp3,.TRUE.)
call MPI_Reduce(tPerturbEnergy,t49Sum,1,&
MPI_DOUBLE_PRECISION,MPI_SUM,0,MPI_COMM_WORLD,ierr)

! Divergence
divP = divnorm(vf1,vf2,vf3)
call MPI_Reduce(divP,divV,1,MPI_DOUBLE_PRECISION,MPI_MAX, &
                0,MPI_COMM_WORLD,ierr)

divP = divnorm(bf1,bf2,bf3)
call MPI_Reduce(divP,divB,1,MPI_DOUBLE_PRECISION,MPI_MAX, &
                0,MPI_COMM_WORLD,ierr)

if(procId.eq.0) then
    write(*,'(I6,G17.10,3G25.15E3)') itr, now, totalEnergy, magneticEnergySum, &
    temperatureSum
endif

grad2 = 0.; grad4 =0.
call computeGradient(vf1,vf2,vf3,grad2,grad4)

call MPI_Reduce(grad2,sumGrad2,1,MPI_DOUBLE_PRECISION, &
```

```
    MPI_SUM,0,MPI_COMM_WORLD,ierr)
call MPI_Reduce(grad4,sumGrad4,1,MPI_DOUBLE_PRECISION, &
    MPI_SUM,0,MPI_COMM_WORLD,ierr)
end

subroutine storeData()
implicit none

call gatherEnergyInShells()
call gatherEnergyInFullShells()

! Odd harmonics
call storeHarmonic(31,2,1,1,pr211) ! (1,0,0)
call storeHarmonic(32,1,2,1,pr121) ! (0,1,0)
call storeHarmonic(33,1,1,2,pr112) ! (0,0,1)
call storeHarmonic(34,2,2,2,pr222) ! (1,1,1)

! Even harmonics
call storeHarmonic(41,3,1,1,pr311) ! (2,0,0)
call storeHarmonic(42,1,3,1,pr131) ! (0,2,0)
call storeHarmonic(43,1,1,3,pr113) ! (0,0,2)
call storeHarmonic(44,3,3,3,pr333) ! (2,2,2)

call storeHarmonic(45,3,2,2,pr322) ! (2,1,1)
call storeHarmonic(46,2,3,2,pr232) ! (1,2,1)
call storeHarmonic(47,2,2,3,pr223) ! (1,1,2)

call storeHarmonic(48,5,1,1,pr511) ! (4,0,0)
call storeHarmonic(49,1,5,1,pr151) ! (0,4,0)
call storeHarmonic(50,1,1,5,pr115) ! (0,0,4)

if(procId.eq.0) then
    write(1,'(G17.10,5G25.15E3)') now, totalEnergy, &
    magneticEnergySum, temperatureSum, divV, divB
    write(2,'(G17.10,3G25.15E3)') now, totalEnergy, sumGrad2, sumGrad4

    write(37) now, kineticEnergyInShells, &
    magneticEnergyInShells, temperatureInShells
    write(52,'(G17.10,6G25.15E3)') now, vPerturbSum, &
    bPerturbSum, tPerturbSum, v49Sum, b49Sum, t49Sum
endif
end
```

```
subroutine findProcessor(hi,hj,hk,processor)
implicit none

integer hk, hj, hi
integer  fstart2, fstart3, fend2, fend3, processor

do l = 1, nproc
    fstart2 = REAL(jindex(l)); fend2 = IMAG(jindex(l))
    fstart3 = REAL(iindex(l)); fend3 = IMAG(iindex(l))

    if(hj.ge.fstart2.and.hj.le.fend2.and.&
    hi.ge.fstart3.and.hi.le.fend3) then
        processor = l-1

        return
    endif
enddo
end

subroutine storeHarmonic(unit,hi,hj,hk,processor)
implicit none

complex(kind = rk), dimension(6) :: scf, rcf

! Indices for the harmonics
integer hk, hj, hi, unit, processor

if(nproc.gt.1) then
    if(procId.eq.0) then
        if(processor.eq.0) then
            write(unit,'(G17.10,12G25.15E3)') now, &
            REAL(vf1(hk,hj,hi)),IMAG(vf1(hk,hj,hi)),&
            REAL(vf2(hk,hj,hi)),IMAG(vf2(hk,hj,hi)),&
            REAL(vf3(hk,hj,hi)),IMAG(vf3(hk,hj,hi)),&
            REAL(bf1(hk,hj,hi)),IMAG(bf1(hk,hj,hi)),&
            REAL(bf2(hk,hj,hi)),IMAG(bf2(hk,hj,hi)),&
            REAL(bf3(hk,hj,hi)),IMAG(bf3(hk,hj,hi))
        else
            call MPI_RECV(rcf,6,MPI_DOUBLE_COMPLEX,processor,51, &
            MPI_COMM_WORLD,status,ierr)
```

```
            write(unit,'(G17.10,12G25.15E3)') now, &
            REAL(rcf(1)),IMAG(rcf(1)), &
            REAL(rcf(2)),IMAG(rcf(2)),REAL(rcf(3)),IMAG(rcf(3)),&
            REAL(rcf(4)),IMAG(rcf(4)),REAL(rcf(5)),IMAG(rcf(5)),&
            REAL(rcf(6)),IMAG(rcf(6))
        endif
    else if(procId.eq.processor) then
        scf(1) = vf1(hk,hj,hi); scf(2) = vf2(hk,hj,hi);
        scf(3) = vf3(hk,hj,hi)
        scf(4) = bf1(hk,hj,hi); scf(5) = bf2(hk,hj,hi);
        scf(6) = bf3(hk,hj,hi)

        call MPI_SEND(scf,6,MPI_DOUBLE_COMPLEX,0,51,MPI_COMM_WORLD,ierr)
    endif
else
    write(unit,'(G17.10,12G25.15E3)') now, &
    REAL(vf1(hk,hj,hi)),IMAG(vf1(hk,hj,hi)),&
    REAL(vf2(hk,hj,hi)),IMAG(vf2(hk,hj,hi)),&
    REAL(vf3(hk,hj,hi)),IMAG(vf3(hk,hj,hi)),&
    REAL(bf1(hk,hj,hi)),IMAG(bf1(hk,hj,hi)),&
    REAL(bf2(hk,hj,hi)),IMAG(bf2(hk,hj,hi)),&
    REAL(bf3(hk,hj,hi)),IMAG(bf3(hk,hj,hi))
endif
end

subroutine getMaxFourierCoefficient(fc)
implicit none

complex(kind = rk), dimension(:,:,:), allocatable :: fc
real(kind = rk) :: tmp

do i = fstart(3),fend(3)
    do j = fstart(2),fend(2)
        do k = fstart(1),fend(1)
            tmp = abs(fc(k,j,i))
            if(maxvalue.lt.tmp) then
                maxvalue = abs(fc(k,j,i))
                idx1 = dx(i); idx2 = dy(j); idx3 = dzs(k)
            endif
        enddo
    enddo
enddo
```

```
end

subroutine initializeVelocityField()
implicit none

do  i = fstart(3), fend(3)
    do j = fstart(2), fend(2)
        do k = fstart(1), fend(1)
            if(k.eq.2.and.j.eq.1.and.i.eq.2) then
                vf2(k,j,i) = cmplx(1,2)
            endif
        enddo
    enddo
enddo
end

subroutine initializeTemperatureProfile()
implicit none

do  i = fstart(3), fend(3)
    do j = fstart(2), fend(2)
        do k = fstart(1), fend(1)
            tf(k,j,i) = 0.
        enddo
    enddo
enddo
end

subroutine allocateArrays()
implicit none

! Arrays
! Velocity field
allocate (vp1(istart(1):iend(1),istart(2):iend(2), &
    istart(3):iend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array vp1'
endif
allocate (vp2(istart(1):iend(1),istart(2):iend(2), &
    istart(3):iend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array vp2'
```

```
endif
allocate (vp3(istart(1):iend(1),istart(2):iend(2), &
    istart(3):iend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array vp3'
endif

allocate(vf1(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array vf1'
endif
allocate(vf2(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array vf2'
endif
allocate(vf3(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array vf3'
endif

! Magnetic field
allocate (bp1(istart(1):iend(1),istart(2):iend(2), &
    istart(3):iend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array bp1'
endif
allocate (bp2(istart(1):iend(1),istart(2):iend(2), &
    istart(3):iend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array bp2'
endif
allocate (bp3(istart(1):iend(1),istart(2):iend(2), &
    istart(3):iend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array bp3'
endif
allocate(bf1(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
```

```
    print *,'Error ',ierr,' allocating array bf1'
endif
allocate(bf2(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array bf2'
endif
allocate(bf3(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array bf3'
endif

! Temporary arrays
allocate (vftmp(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array vftmp'
endif

allocate (vfrtmp(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array vfrtmp'
endif

allocate (vfctmp(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array vfctmp'
endif

allocate (vfr(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array vfr'
endif

allocate (vfc(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array vfc'
```

```
endif

allocate (r3tmp(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array r3tmp'
endif

allocate (vcp1(istart(1):iend(1),istart(2):iend(2), &
    istart(3):iend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array vcp1'
endif
allocate (vcp2(istart(1):iend(1),istart(2):iend(2), &
    istart(3):iend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array vcp2'
endif
allocate (vcp3(istart(1):iend(1),istart(2):iend(2), &
    istart(3):iend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array vcp3'
endif

!Temperature profile
allocate (tp(istart(1):iend(1),istart(2):iend(2), &
    istart(3):iend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array tp'
endif
allocate(tf(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array tf'
endif

allocate(xtmp1(istart(1):iend(1),istart(2):iend(2), &
    istart(3):iend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array xtmp1'
endif
allocate(xtmp2(istart(1):iend(1),istart(2):iend(2), &
```

```
    istart(3):iend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array xtmp2'
endif
allocate(xtmp3(istart(1):iend(1),istart(2):iend(2), &
    istart(3):iend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array xtmp3'
endif

allocate (ftmp1(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array ftmp1'
endif
allocate (ftmp2(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array ftmp2'
endif
allocate (ftmp3(fstart(1):fend(1),fstart(2):fend(2),&
    fstart(3):fend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array ftmp3'
endif

allocate (tftmp(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr .ne. 0) then
    print *,'Error ',ierr,' allocating array tftmp'
endif

allocate(aftmp1(fstart(1):fend(1),fstart(2):fend(2),&
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array aftmp1'
endif
allocate(aftmp2(fstart(1):fend(1),fstart(2):fend(2),&
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array aftmp2'
endif
```

```
allocate(aftmp3(fstart(1):fend(1),fstart(2):fend(2),&
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array aftmp3'
endif

allocate(vftmp1(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array vftmp1'
endif
allocate(vftmp2(fstart(1):fend(1),fstart(2):fend(2),&
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array vftmp2'
endif
allocate(vftmp3(fstart(1):fend(1),fstart(2):fend(2),&
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array vftmp3'
endif

allocate(bftmp1(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array bftmp1'
endif
allocate(bftmp2(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array bftmp2'
endif
allocate(bftmp3(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array bftmp3'
endif

! Arrays for RungeKutta & ETDRK3
allocate(v11(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
```

```
    print *,'Error ',ierr,' allocating array v11'
endif
allocate(v12(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array v12'
endif
    allocate(v13(fstart(1):fend(1),fstart(2):fend(2), &
        fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array v13'
endif
allocate(v21(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array v21'
endif
    allocate(v22(fstart(1):fend(1),fstart(2):fend(2), &
        fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array v22'
endif
allocate(v23(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array v23'
endif
allocate(v31(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array v31'
endif
allocate(v32(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array v32'
endif
allocate(v33(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array v33'
endif
```

```
allocate(k11(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array k11'
endif
allocate(k12(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array k12'
endif
    allocate(k13(fstart(1):fend(1),fstart(2):fend(2), &
        fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array k13'
endif
allocate(k21(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array k21'
endif
    allocate(k22(fstart(1):fend(1),fstart(2):fend(2), &
        fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array k22'
endif
allocate(k23(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array k23'
endif
allocate(k31(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array k31'
endif
allocate(k32(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array k32'
endif
allocate(k33(fstart(1):fend(1),fstart(2):fend(2), &
```

```
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array k33'
endif

allocate(t10(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array t10'
endif

allocate(t20(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array t20'
endif

allocate(t30(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array t30'
endif

! Arrays for exponential time differencing
allocate(cex(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array cex'
endif

allocate(c11(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array c11'
endif
allocate(c21(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array c21'
endif
allocate(c31(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
```

```
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array c31'
endif

allocate(c12(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array c12'
endif
allocate(c22(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array c22'
endif
allocate(c32(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array c32'
endif

allocate(c13(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array c13'
endif
allocate(c23(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array c23'
endif
allocate(c33(fstart(1):fend(1),fstart(2):fend(2), &
    fstart(3):fend(3)), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array c33'
endif

allocate(svf1(1:nz,1:ny,1:nx/2+1), stat=ierr)
if(ierr.ne.0) then
        print *,'Error ',ierr,' allocating array svf1'
endif
allocate(svf2(1:nz,1:ny,1:nx/2+1), stat=ierr)
if(ierr.ne.0) then
```

```
        print *,'Error ',ierr,' allocating array svf2'
endif
allocate(svf3(1:nz,1:ny,1:nx/2+1), stat=ierr)
if(ierr.ne.0) then
        print *,'Error ',ierr,' allocating array svf3'
endif

allocate(sbf1(1:nz,1:ny,1:nx/2+1), stat=ierr)
if(ierr.ne.0) then
        print *,'Error ',ierr,' allocating array sbf1'
endif
allocate(sbf2(1:nz,1:ny,1:nx/2+1), stat=ierr)
if(ierr.ne.0) then
        print *,'Error ',ierr,' allocating array sbf2'
endif
allocate(sbf3(1:nz,1:ny,1:nx/2+1), stat=ierr)
if(ierr.ne.0) then
        print *,'Error ',ierr,' allocating array sbf3'
endif

allocate(stfx(1:nz,1:ny,1:nx/2+1), stat=ierr)
if(ierr.ne.0) then
        print *,'Error ',ierr,' allocating array stfx'
endif

if(procId.eq.0) then
    allocate(tvf1(1:nz,1:ny,1:nx/2+1), stat=ierr)
    if(ierr.ne.0) then
        print *,'Error ',ierr,' allocating array tvf1'
    endif
    allocate(tvf2(1:nz,1:ny,1:nx/2+1), stat=ierr)
    if(ierr.ne.0) then
        print *,'Error ',ierr,' allocating array tvf2'
    endif
    allocate(tvf3(1:nz,1:ny,1:nx/2+1), stat=ierr)
    if(ierr.ne.0) then
        print *,'Error ',ierr,' allocating array tvf3'
    endif

    allocate(tbf1(1:nz,1:ny,1:nx/2+1), stat=ierr)
    if(ierr.ne.0) then
        print *,'Error ',ierr,' allocating array tbf1'
```

```
endif
allocate(tbf2(1:nz,1:ny,1:nx/2+1), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array tbf2'
endif
allocate(tbf3(1:nz,1:ny,1:nx/2+1), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array tbf3'
endif

allocate(tyf(1:nz,1:ny,1:nx/2+1), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array tyf'
endif

allocate(tmvf1(1:nx/2+1,1:ny,1:nz), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array tmvf1'
endif
allocate(tmvf2(1:nx/2+1,1:ny,1:nz), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array tmvf2'
endif
allocate(tmvf3(1:nx/2+1,1:ny,1:nz), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array tmvf3'
endif

allocate(tmbf1(1:nx/2+1,1:ny,1:nz), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array tmbf1'
endif
allocate(tmbf2(1:nx/2+1,1:ny,1:nz), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array tmbf2'
endif
allocate(tmbf3(1:nx/2+1,1:ny,1:nz), stat=ierr)
if(ierr.ne.0) then
    print *,'Error ',ierr,' allocating array tmbf3'
endif

allocate(tmyf(1:nx/2+1,1:ny,1:nz), stat=ierr)
```

```
    if(ierr.ne.0) then
        print *,'Error ',ierr,' allocating array tmyf'
    endif

    allocate(kineticEnergyInShells(1:kfil+1))
    allocate(magneticEnergyInShells(1:kfil+1))
    allocate(temperatureInShells(1:kfil+1))

    allocate(vFullShells(1:ns+1))
    allocate(bFullShells(1:ns+1))
    allocate(tFullShells(1:ns+1))
endif

! Pointer arrays
allocate(iftmp(fsize(1)-2,fsize(2),fsize(3)),stat=ierr)
if(ierr.ne.0) print *,'Error ',ierr,' allocating iftmp'
allocate(oftmp(fsize(1)-2,fsize(2),fsize(3)),stat=ierr)
if(ierr.ne.0) print *,'Error ',ierr,' allocating oftmp'

! Arrays for wave numbers
allocate(dx(fstart(1):fend(1)))
allocate(dy(fstart(2):fend(2)))
allocate(dzs(fstart(3):fend(3)))

allocate(dc(nx))
allocate(dcy(ny))
allocate(dcz(nz))
allocate(arr(1:3))
allocate(brr(1:3))

allocate(espv(1:kfil+1))!includes incomplete shell
allocate(espb(1:kfil+1))
allocate(espt(1:kfil+1))

allocate(env(1:ns+1))
allocate(enb(1:ns+1))
allocate(ent(1:ns+1))

allocate(jindex(nproc))
allocate(iindex(nproc))

allocate(rvf1(1:nz+1)); allocate(rvf2(1:nz+1));
```

```
allocate(rvf3(1:nz+1))
allocate(rbf1(1:nz+1)); allocate(rbf2(1:nz+1));
allocate(rbf3(1:nz+1))
allocate(msg(1:nz+1));allocate(rtf(1:nz+1))
end

subroutine printFourierElements(fc)
implicit none

complex(kind = rk), dimension(:,:,:),  allocatable :: fc

do i = fstart(3), fend(3)
    do j = fstart(2), fend(2)
        do k = fstart(1), fend(1)
            if(abs(fc(k,j,i)).gt.tolerance) then
                write(*,*) dx(k), dy(j), dzs(i), fc(k,j,i)
            endif
        enddo
    enddo
enddo
end

subroutine forwardTransform(in,out,type)
implicit none

real(kind = rk), dimension(:,:,:),  allocatable :: in
complex(kind = rk), dimension(:,:,:),  allocatable :: out
integer, intent(in) :: type

call MPI_BARRIER(MPI_COMM_WORLD, ierr)

if(type.eq.1) then
    ! Cosine
    call p3dfft_ftran_r2c(in, out,'ffc')
else
    ! Sine
    call computeForwardSineTransform(in, out)
endif

out = out/real(nx * ny * (nz-1))
end
```

```
subroutine backwardTransform(in,out,type)
implicit none

complex(kind = rk), dimension(:,:,:),  allocatable :: in
real(kind = rk), dimension(:,:,:),  allocatable :: out
integer type

call MPI_BARRIER(MPI_COMM_WORLD, ierr)

if(type.eq.1) then
    ! Cosine
    call p3dfft_btran_c2r(in, out,'cff')
else
    ! Sine
    call computeBackwardSineTransform(in, out)
endif

out = out/2.0_rk
end

subroutine computeBackwardSineTransform(in,out)
implicit none

complex(kind = rk), dimension(:,:,:),  allocatable :: in
real(kind = rk), dimension(:,:,:),  allocatable :: out

!Real
call dfftw_execute_r2r(sinePlan, &
    REAL(in(2:nz-1,fstart(2):fend(2),fstart(3):fend(3))), &
    vfrtmp(2:nz-1,fstart(2):fend(2),fstart(3):fend(3)))
vfrtmp(1,fstart(2):fend(2),fstart(3):fend(3)) = 0.0_rk
vfrtmp(nz,fstart(2):fend(2),fstart(3):fend(3)) = 0.0_rk

! Complex
call dfftw_execute_r2r(sinePlan, &
    IMAG(in(2:nz-1,fstart(2):fend(2),fstart(3):fend(3))), &
    vfctmp(2:nz-1,fstart(2):fend(2),fstart(3):fend(3)))
vfctmp(1,fstart(2):fend(2),fstart(3):fend(3)) = 0.0_rk
vfctmp(nz,fstart(2):fend(2),fstart(3):fend(3)) = 0.0_rk

vftmp = cmplx(0.0_rk,1.0_rk) * vfctmp + vfrtmp
```

```
call p3dfft_btran_c2r(vftmp,out,'nff')
end

subroutine computeBackwardCosineTransform(in,out)
implicit none

complex(kind = rk), dimension(:,:,:),  allocatable :: in
real(kind = rk), dimension(:,:,:),  allocatable :: out

!Real
call dfftw_execute_r2r(cosinePlan,REAL(in),vfrtmp)

! Complex
call dfftw_execute_r2r(cosinePlan,IMAG(in),vfctmp)

vftmp = cmplx(0.0_rk,1.0_rk) * vfctmp + vfrtmp

call p3dfft_btran_c2r(vftmp,out,'nff')
end

subroutine computeForwardSineTransform(in,out)
implicit none

real(kind = rk), dimension(:,:,:),  allocatable :: in
complex(kind = rk), dimension(:,:,:),  allocatable :: out

call p3dfft_ftran_r2c(in,vftmp,'ffn')

! Real part
call dfftw_execute_r2r(sineplan, &
REAL(vftmp(2:nz-1,fstart(2):fend(2),fstart(3):fend(3))),&
    vfr(2:nz-1,fstart(2):fend(2),fstart(3):fend(3)))
vfr(1,fstart(2):fend(2),fstart(3):fend(3)) = 0.0_rk
vfr(nz,fstart(2):fend(2),fstart(3):fend(3)) = 0.0_rk

! Imaginary part
call dfftw_execute_r2r(sinePlan, &
IMAG(vftmp(2:nz-1,fstart(2):fend(2),fstart(3):fend(3))),&
    vfc(2:nz-1,fstart(2):fend(2),fstart(3):fend(3)))
vfc(1,fstart(2):fend(2),fstart(3):fend(3)) = 0.0_rk
vfc(nz,fstart(2):fend(2),fstart(3):fend(3)) = 0.0_rk
```

```
out = cmplx(0.0_rk,1.0_rk) * vfc + vfr
end

subroutine computeForwardCosineTransform(in,out)
implicit none

real(kind = rk), dimension(:,:,:),  allocatable :: in
complex(kind = rk), dimension(:,:,:),  allocatable :: out

call p3dfft_ftran_r2c(in,vftmp,'ffn')

! Real part
call dfftw_execute_r2r(cosineplan,REAL(vftmp),vfr)
! Imaginary part
call dfftw_execute_r2r(cosinePlan,IMAG(vftmp),vfc)

out = cmplx(0.0_rk,1.0_rk) * vfc + vfr
end

subroutine initFFTW()
implicit none

integer success

numx = fend(3) - fstart(3) + 1
numy = fend(2) - fstart(2) + 1
num = numx * numy

fileUnit = WISDOM_FILE_INDEX + procId

write(wisdomFile, *) procId
wisdomFile = 'wisdom'//trim(adjustl(wisdomFile))

open(fileUnit, access = 'sequential', form = 'formatted', &
file = WISDOM_FOLDER//wisdomFile, status = 'unknown', recl = 256)

if(programType.eq.1) then
    call importWisdomFromFile(success,fileUnit)

    F_FLAG =  FFTW_PATIENT + FFTW_WISDOM_ONLY
else
    F_FLAG = FFTW_PATIENT
```

```
endif

istride = 1
idist = nz-2

call dfftw_plan_many_r2r(sinePlan, 1, nz-2, num, &
vfrtmp(2:nz-1,fstart(2):fend(2),fstart(3):fend(3)), nz-2, istride, idist, &
vfr(2:nz-1,fstart(2):fend(2),fstart(3):fend(3)), nz-2, istride, idist, &
FFTW_RODFT00, F_FLAG)

if(programType.eq.0.or.programType.eq.2) then
    call exportWisdomToFile(fileUnit)
endif

close(fileUnit)
end

subroutine minusGrad(fc1,fc2,fc3)
implicit none

complex(p3dfft_type), dimension(:,:,:),  allocatable :: fc1, fc2, fc3
complex(p3dfft_type) :: pk
real(p3dfft_type)    :: qk
integer(p3dfft_type) :: index

do i = fstart(3), fend(3)
    do j = fstart(2), fend(2)
        do k = fstart(1), fend(1)
            index = i + j + k
            if(index.gt.3) then

            qk = ((alphax * dc(i))**2 + (alphay * dcy(j))**2 &
                    + (pi * dcz(k))**2)

            pk = (dc(i) * alphax * ci * fc1(k,j,i) +  &
                dcy(j) * alphay * ci * fc2(k,j,i) +  &
                dcz(k) * pi * fc3(k,j,i))/qk

            arr(1) = -pk * dc(i) * alphax * ci
            arr(2) = -pk * dcy(j) * alphay * ci
            arr(3) = pk * dcz(k) * pi
```

```
            fc1(k,j,i) = fc1(k,j,i) - arr(1)
            fc2(k,j,i) = fc2(k,j,i) - arr(2)
            fc3(k,j,i) = fc3(k,j,i) - arr(3)
            endif
        enddo
    enddo
enddo

if(fstart(1).eq.1.and.fstart(2).eq.1) then
    fc1(1,1,fstart(3):fend(3)) = 0.
endif
if(fstart(1).eq.1.and.fstart(3).eq.1) then
    fc2(1,fstart(2):fend(2),1) = 0.
endif
if(fstart(2).eq.1.and.fstart(3).eq.1) then
    fc3(fstart(1):fend(1),1,1) = 0.
endif
end

subroutine checkVelocityField()
implicit none

call MPI_BARRIER(MPI_COMM_WORLD,ierr)
call backwardTransform(vf1,vp1,1)
call backwardTransform(vf2,vp2,1)
call backwardTransform(vf3,vp3,0)

call checkPhysicalField(vcp1, vp1)
call checkPhysicalField(vcp2, vp2)
call checkPhysicalField(vcp3, vp3)
end

subroutine checkPhysicalField(fp,fcp)
implicit none

real(kind = rk), dimension(:,:,:),  allocatable :: fp, fcp
real(kind = rk) nglob, sendData, recvData

sendData = 0.0d0
recvData = 0.0d0
do k = istart(3),iend(3)
    do j = istart(2),iend(2)
```

```
        do i = istart(1),iend(1)
            if(sendData.lt.abs(fcp(i,j,k)-fp(i,j,k))) then
                sendData = abs(fcp(i,j,k)-fp(i,j,k))
            endif
        enddo
    enddo
enddo

call MPI_Reduce(sendData,recvData,1,MPI_DOUBLE_PRECISION, &
    MPI_MAX,0,MPI_COMM_WORLD,ierr)

if(procId.eq.0) then
    nglob = dble(nx) * dble(ny) * dble(nz-1)
    if(recvData.gt.1e-14 * nglob * 0.25) then
        print *,'Results are incorrect'
    else
        print *,'Results are correct'
    endif
    write(*,*) 'max diff =', recvData
endif
end

subroutine printPhysicalElements(fp)
implicit none

real(kind = rk), dimension(:,:,:),  allocatable :: fp
integer :: count = 1

do k = istart(3), iend(3)
    do j = istart(2), iend(2)
        do i = istart(1), iend(1)
            if(fp(i,j,k).gt.tolerance) then
                print *, count, i, j, k, fp(i,j,k)
                count = count + 1
            endif
        enddo
    enddo
enddo
end

subroutine computeNonlinearTermsForNavierStokes(ivf1,ivf2,ivf3)
implicit none
```

```
complex(p3dfft_type), dimension(:,:,:), allocatable :: ivf1,ivf2,ivf3

call axrota(vf1,vf2,vf3,vp1,vp2,vp3,ivf1,ivf2,ivf3)
call axrota(bf1,bf2,bf3,bp1,bp2,bp3,ftmp1,ftmp2,ftmp3)
ivf1 = ivf1 - ftmp1
ivf2 = ivf2 - ftmp2
ivf3 = ivf3 - ftmp3

do i = fstart(3),fend(3)
    do j = fstart(2),fend(2)
        do k = fstart(1),fend(1)
        ivf1(k,j,i) = ivf1(k,j,i) + P * sqrt(TAYLOR) * &
            vf2(k,j,i)
        ivf2(k,j,i) = ivf2(k,j,i) - P * sqrt(TAYLOR) * &
            vf1(k,j,i)
        ivf3(k,j,i) = ivf3(k,j,i) + P * RAYLEIGH * &
            tf(k,j,i)
        enddo
    enddo
enddo

call minusGrad(ivf1,ivf2,ivf3)
end

subroutine axrota(af1,af2,af3,ap1,ap2,ap3,rotf1,rotf2,rotf3)
implicit none

complex(p3dfft_type), dimension(:,:,:), allocatable :: af1,af2,af3
complex(p3dfft_type), dimension(:,:,:), allocatable :: rotf1,rotf2,rotf3
real(p3dfft_type), dimension(:,:,:), allocatable :: ap1,ap2,ap3

call computeRot(af1,af2,af3,rotf1,rotf2,rotf3,1)

! sine - sine - cosine
call backwardTransform(rotf1,xtmp1,0)
call backwardTransform(rotf2,xtmp2,0)
call backwardTransform(rotf3,xtmp3,1)

do k = istart(3),iend(3)
    do j = istart(2),iend(2)
        do i = istart(1),iend(1)
```

```
            brr(1) = ap2(i,j,k) * xtmp3(i,j,k) - ap3(i,j,k) * xtmp2(i,j,k)
            brr(2) = ap3(i,j,k) * xtmp1(i,j,k) - ap1(i,j,k) * xtmp3(i,j,k)
            brr(3) = ap1(i,j,k) * xtmp2(i,j,k) - ap2(i,j,k) * xtmp1(i,j,k)

            xtmp1(i,j,k) = brr(1)
            xtmp2(i,j,k) = brr(2)
            xtmp3(i,j,k) = brr(3)
        enddo
    enddo
enddo

! cosine - cosine - sine
call forwardTransform(xtmp1,rotf1,1)
call forwardTransform(xtmp2,rotf2,1)
call forwardTransform(xtmp3,rotf3,0)
end

! fctmp, rvxb in Fourier space
subroutine computeRot(fctmp1,fctmp2,fctmp3,rvxb1,rvxb2,rvxb3,type)
implicit none

complex(p3dfft_type), dimension(:,:,:), allocatable :: rvxb1,rvxb2,rvxb3
complex(p3dfft_type), dimension(:,:,:), allocatable :: fctmp1,fctmp2,fctmp3
integer type

if(type.eq.0) then
    ! sine - sine - cosine
    do i = fstart(3), fend(3)
        do j = fstart(2), fend(2)
            do k = fstart(1), fend(1)
            rvxb1(k,j,i) = ci * dcy(j) * alphay * &
            fctmp3(k,j,i) - dcz(k) * pi * fctmp2(k,j,i)
            rvxb2(k,j,i) = dcz(k) * pi * fctmp1(k,j,i) - &
            ci * dc(i) * alphax * fctmp3(k,j,i)
            rvxb3(k,j,i) = ci * dc(i) * alphax * &
            fctmp2(k,j,i) - ci * dcy(j) * alphay * &
                fctmp1(k,j,i)
            enddo
        enddo
    enddo
else if(type.eq.1) then
    ! cosine - cosine - sine
```

```
    do i = fstart(3), fend(3)
        do j = fstart(2), fend(2)
            do k = fstart(1), fend(1)
            rvxb1(k,j,i) = &
            ci * dcy(j) * alphay * fctmp3(k,j,i) + &
            dcz(k) * pi * fctmp2(k,j,i)
            rvxb2(k,j,i) = &
            - dcz(k) * pi * fctmp1(k,j,i) - &
            ci * dc(i) * alphax * fctmp3(k,j,i)
            rvxb3(k,j,i) = &
            ci * dc(i) * alphax * fctmp2(k,j,i) - &
            ci * dcy(j) * alphay * fctmp1(k,j,i)
            enddo
        enddo
    enddo
endif
end

subroutine cleanFourierCoefficients2()
implicit none

complex(kind = rk)  :: vf(nx/2+1,ny,nz,3),bf(nx/2+1,ny,nz,3), &
tmf(nx/2+1,ny,nz)
integer fstart2, fstart3, fend2, fend3, vi, vj, vk

! ******************** DISTRIBUTE ********************** !
do i = 1, nx/2+1
    do j = 1, ny

        if(nproc.gt.1) then
            do l = 1, nproc
                fstart2 = REAL(jindex(l));
                fend2 = IMAG(jindex(l))
                fstart3 = REAL(iindex(l));
                fend3 = IMAG(iindex(l))

                if(j.ge.fstart2.and.j.le.fend2.and.&
                i.ge.fstart3.and.i.le.fend3) then
                    source = l-1
                    go to 27
                endif
            enddo
```

```
else
    source = procId
endif

27 if(procId.eq.0) then
    if(source.eq.0) then
        vf(i,j,1:nz,1) = vf1(1:nz,j,i)
        vf(i,j,1:nz,2) = vf2(1:nz,j,i)
        vf(i,j,1:nz,3) = vf3(1:nz,j,i)

        bf(i,j,1:nz,1) = bf1(1:nz,j,i)
        bf(i,j,1:nz,2) = bf2(1:nz,j,i)
        bf(i,j,1:nz,3) = bf3(1:nz,j,i)

        tmf(i,j,1:nz) = tf(1:nz,j,i)
    else
    call MPI_RECV(rvf1,nz+1,MPI_DOUBLE_COMPLEX,&
    MPI_ANY_SOURCE,1,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rvf1(nz+1)); vi = IMAG(rvf1(nz+1))
    vf(vi,vj,1:nz,1) = rvf1(1:nz)

    call MPI_RECV(rvf2,nz+1,MPI_DOUBLE_COMPLEX,&
    MPI_ANY_SOURCE,2,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rvf2(nz+1)); vi = IMAG(rvf2(nz+1))
    vf(vi,vj,1:nz,2) = rvf2(1:nz)

    call MPI_RECV(rvf3,nz+1,MPI_DOUBLE_COMPLEX,&
    MPI_ANY_SOURCE,3,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rvf3(nz+1)); vi = IMAG(rvf3(nz+1))
    vf(vi,vj,1:nz,3) = rvf3(1:nz)

    call MPI_RECV(rbf1,nz+1,MPI_DOUBLE_COMPLEX,&
    MPI_ANY_SOURCE,4,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rbf1(nz+1)); vi = IMAG(rbf1(nz+1))
    bf(vi,vj,1:nz,1) = rbf1(1:nz)

    call MPI_RECV(rbf2,nz+1,MPI_DOUBLE_COMPLEX,&
    MPI_ANY_SOURCE,5,MPI_COMM_WORLD,status,ierr)
    vj = REAL(rbf2(nz+1)); vi = IMAG(rbf2(nz+1))
    bf(vi,vj,1:nz,2) = rbf2(1:nz)

    call MPI_RECV(rbf3,nz+1,MPI_DOUBLE_COMPLEX,&
```

```
            MPI_ANY_SOURCE,6,MPI_COMM_WORLD,status,ierr)
            vj = REAL(rbf3(nz+1)); vi = IMAG(rbf3(nz+1))
            bf(vi,vj,1:nz,3) = rbf3(1:nz)

            call MPI_RECV(rtf,nz+1,MPI_DOUBLE_COMPLEX,&
            MPI_ANY_SOURCE,7,MPI_COMM_WORLD,status,ierr)
            vj = REAL(rtf(nz+1)); vi = IMAG(rtf(nz+1))
            tmf(vi,vj,1:nz) = rtf(1:nz)
            endif
        else if(procId.eq.source) then
            msg(1:nz) = vf1(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,1,&
            MPI_COMM_WORLD,ierr)
            msg(1:nz) = vf2(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,2,&
            MPI_COMM_WORLD,ierr)
            msg(1:nz) = vf3(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,3,&
            MPI_COMM_WORLD,ierr)

            msg(1:nz) = bf1(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,4,&
            MPI_COMM_WORLD,ierr)
            msg(1:nz) = bf2(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,5,&
            MPI_COMM_WORLD,ierr)
            msg(1:nz) = bf3(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,6,&
            MPI_COMM_WORLD,ierr)

            msg(1:nz) = tf(1:nz,j,i);
            msg(nz+1) = cmplx(j,i)
            call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX,0,7,&
            MPI_COMM_WORLD,ierr)
        endif
    enddo
```

```
enddo

if(procId.eq.0) then

    call updateFourierCoefficients2(vf,bf,tmf)

    tmvf1(:,:,:) = vf(:,:,:,1); tmvf2(:,:,:) = vf(:,:,:,2);
    tmvf3(:,:,:) = vf(:,:,:,3)
    tmbf1(:,:,:) = bf(:,:,:,1); tmbf2(:,:,:) = bf(:,:,:,2);
    tmbf3(:,:,:) = bf(:,:,:,3)
    tmyf(:,:,:) = tmf

    ! Stride format
    do i = 1, nx/2+1
        do j = 1, ny
        tvf1(1:nz,j,i) = tmvf1(i,j,1:nz); tvf2(1:nz,j,i) = tmvf2(i,j,1:nz);
        tvf3(1:nz,j,i) = tmvf3(i,j,1:nz)
        tbf1(1:nz,j,i) = tmbf1(i,j,1:nz); tbf2(1:nz,j,i) = tmbf2(i,j,1:nz);
        tbf3(1:nz,j,i) = tmbf3(i,j,1:nz)
        tyf(1:nz,j,i) = tmyf(i,j,1:nz)
        enddo
    enddo

    if(nproc.eq.1) then
        vf1(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tvf1(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        vf2(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tvf2(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        vf3(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tvf3(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        bf1(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tbf1(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        bf2(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tbf2(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        bf3(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tbf3(1:nz,fstart(2):fend(2),fstart(3):fend(3))
        tf(1:nz,fstart(2):fend(2),fstart(3):fend(3)) = &
        tyf(1:nz,fstart(2):fend(2),fstart(3):fend(3))
    else
        do l = 1, nproc
        fstart2 = REAL(jindex(l));
        fend2 = IMAG(jindex(l))
```

```
fstart3 = REAL(iindex(l));
fend3 = IMAG(iindex(l))

if(l.eq.1) then
    vf1(1:nz,fstart2:fend2,fstart3:fend3) = &
        tvf1(1:nz,fstart2:fend2,fstart3:fend3)
    vf2(1:nz,fstart2:fend2,fstart3:fend3) = &
        tvf2(1:nz,fstart2:fend2,fstart3:fend3)
    vf3(1:nz,fstart2:fend2,fstart3:fend3) = &
        tvf3(1:nz,fstart2:fend2,fstart3:fend3)
    bf1(1:nz,fstart2:fend2,fstart3:fend3) = &
        tbf1(1:nz,fstart2:fend2,fstart3:fend3)
    bf2(1:nz,fstart2:fend2,fstart3:fend3) = &
        tbf2(1:nz,fstart2:fend2,fstart3:fend3)
    bf3(1:nz,fstart2:fend2,fstart3:fend3) = &
        tbf3(1:nz,fstart2:fend2,fstart3:fend3)
    tf(1:nz,fstart2:fend2,fstart3:fend3) = &
        tyf(1:nz,fstart2:fend2,fstart3:fend3)
else
    do i = fstart3, fend3
        do j = fstart2, fend2
        msg(1:nz) = tvf1(1:nz,j,i);
        msg(nz+1) = cmplx(j,i)
        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
            l-1,1,MPI_COMM_WORLD,ierr)

        msg(1:nz) = tvf2(1:nz,j,i);
        msg(nz+1) = cmplx(j,i)
        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
            l-1,2,MPI_COMM_WORLD,ierr)

        msg(1:nz) = tvf3(1:nz,j,i);
        msg(nz+1) = cmplx(j,i)
        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
            l-1,3,MPI_COMM_WORLD,ierr)

        msg(1:nz) = tbf1(1:nz,j,i);
        msg(nz+1) = cmplx(j,i)
        call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
            l-1,4,MPI_COMM_WORLD,ierr)

        msg(1:nz) = tbf2(1:nz,j,i);
```

```
                msg(nz+1) = cmplx(j,i)
                call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
                    l-1,5,MPI_COMM_WORLD,ierr)

                msg(1:nz) = tbf3(1:nz,j,i);
                msg(nz+1) = cmplx(j,i)
                call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
                    l-1,6,MPI_COMM_WORLD,ierr)

                msg(1:nz) = tyf(1:nz,j,i);
                msg(nz+1) = cmplx(j,i)
                call MPI_SEND(msg,nz+1,MPI_DOUBLE_COMPLEX, &
                    l-1,7,MPI_COMM_WORLD,ierr)
                enddo
            enddo
        endif
        enddo
    endif
else
    do i = fstart(3), fend(3)
        do j = fstart(2), fend(2)
            call MPI_RECV(rvf1,nz+1,MPI_DOUBLE_COMPLEX,0,1, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rvf1(nz+1)); vi = IMAG(rvf1(nz+1))
            vf1(1:nz,vj,vi) = rvf1(1:nz)

            call MPI_RECV(rvf2,nz+1,MPI_DOUBLE_COMPLEX,0,2, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rvf2(nz+1)); vi = IMAG(rvf2(nz+1))
            vf2(1:nz,vj,vi) = rvf2(1:nz)

            call MPI_RECV(rvf3,nz+1,MPI_DOUBLE_COMPLEX,0,3, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rvf3(nz+1)); vi = IMAG(rvf3(nz+1))
            vf3(1:nz,vj,vi) = rvf3(1:nz)

            call MPI_RECV(rbf1,nz+1,MPI_DOUBLE_COMPLEX,0,4, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rbf1(nz+1)); vi = IMAG(rbf1(nz+1))
            bf1(1:nz,vj,vi) = rbf1(1:nz)

            call MPI_RECV(rbf2,nz+1,MPI_DOUBLE_COMPLEX,0,5, &
```

```
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rbf2(nz+1)); vi = IMAG(rbf2(nz+1))
            bf2(1:nz,vj,vi) = rbf2(1:nz)

            call MPI_RECV(rbf3,nz+1,MPI_DOUBLE_COMPLEX,0,6, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rbf3(nz+1)); vi = IMAG(rbf3(nz+1))
            bf3(1:nz,vj,vi) = rbf3(1:nz)

            call MPI_RECV(rtf,nz+1,MPI_DOUBLE_COMPLEX,0,7, &
            MPI_COMM_WORLD,status,ierr)
            vj = REAL(rtf(nz+1)); vi = IMAG(rtf(nz+1))
            tf(1:nz,vj,vi) = rtf(1:nz)
        enddo
    enddo
endif
end

subroutine updateFourierCoefficients2(vf,bf,tmf)
implicit none

complex(kind = rk)  :: vf(nx/2+1,ny,nz,3), &
    bf(nx/2+1,ny,nz,3),tmf(nx/2+1,ny,nz)
integer :: k,j

do k = 1,nz
    do j = 2,ny/2
    vf(1,j,k,:) = &
    cmplx( 0.5_rk * (real(vf(1,j,k,:),kind = rk) + &
    real(vf(1,ny-j+2,k,:),kind = rk)), &
    0.5_rk * (imag(vf(1,j,k,:)) - imag(vf(1,ny-j+2,k,:))),kind = rk)
    vf(1,ny-j+2,k,:) = conjg(vf(1,j,k,:))

    bf(1,j,k,:) = &
    cmplx(0.5_rk * (real(bf(1,j,k,:),kind = rk) + &
    real(bf(1,ny-j+2,k,:),kind = rk)), &
    0.5_rk * (imag(bf(1,j,k,:)) - imag(bf(1,ny-j+2,k,:))),kind = rk)
    bf(1,ny-j+2,k,:) = conjg(bf(1,j,k,:))

    tmf(1,j,k) = &
    cmplx(0.5_rk * (real(tmf(1,j,k),kind = rk) + &
    real(tmf(1,ny-j+2,k),kind = rk)), &
```

```
    0.5_rk * (imag(tmf(1,j,k)) - imag(tmf(1,ny-j+2,k))),kind = rk)
    tmf(1,ny-j+2,k) = conjg(tmf(1,j,k))
    enddo
enddo

vf(1,1,:,:) = real(vf(1,1,:,:),kind = rk)
bf(1,1,:,:) = real(bf(1,1,:,:),kind = rk)
tmf(1,1,:) = real(tmf(1,1,:),kind = rk)

vf(:,:,1,3) = 0.0_rk
bf(:,:,1,3) = 0.0_rk
tmf(:,:,1) = 0.0_rk

vf(1,1,1,:) = 0.0_rk
bf(1,1,1,:) = 0.0_rk
tmf(1,1,1) = 0.0_rk
end

subroutine etdrk3(now,tau)
implicit none

real(p3dfft_type)    :: now, tau, cb, cv, ct
integer(p3dfft_type) :: index

! u_n
vftmp1 = vf1; vftmp2 = vf2; vftmp3 = vf3
tftmp = tf
bftmp1 = bf1; bftmp2 = bf2; bftmp3 = bf3

! F(u_n)
call computeNonLinearTerms(v11,v12,v13,k11,k12,k13,t10)

! a_n
do i = fstart(3), fend(3)
    do j = fstart(2), fend(2)
        do k = fstart(1), fend(1)
        index = i + j + k
        if(index.gt.3) then
            ct = cex(k,j,i)
            cv = P * ct
            cb = cv/Pm
```

```
            v21(k,j,i) = vftmp1(k,j,i) * &
            exp(0.5_rk * cv * tau) + v11(k,j,i) * &
            (exp(0.5_rk * cv * tau) - 1.0_rk)/cv
            v22(k,j,i) = vftmp2(k,j,i) * &
            exp(0.5_rk * cv * tau) + v12(k,j,i) * &
            (exp(0.5_rk * cv * tau) - 1.0_rk)/cv
            v23(k,j,i) = vftmp3(k,j,i) * &
            exp(0.5_rk * cv * tau) + v13(k,j,i) * &
            (exp(0.5_rk * cv * tau) - 1.0_rk)/cv

            k21(k,j,i) = bftmp1(k,j,i) * &
            exp(0.5_rk * cb * tau) + k11(k,j,i) * &
            (exp(0.5_rk * cb * tau)-1.0_rk)/cb
            k22(k,j,i) = bftmp2(k,j,i) * &
            exp(0.5_rk * cb * tau) + k12(k,j,i) * &
            (exp(0.5_rk * cb * tau)-1.0_rk)/cb
            k23(k,j,i) = bftmp3(k,j,i) * &
            exp(0.5_rk * cb * tau) + k13(k,j,i) * &
            (exp(0.5_rk * cb * tau)-1.0_rk)/cb

            t20(k,j,i) = tftmp(k,j,i) * &
            exp(0.5_rk * ct * tau) + t10(k,j,i) * &
            (exp(0.5_rk * ct * tau) - 1.0_rk)/ct
        endif
        enddo
    enddo
enddo

! F(a_n)
vf1 = v21; vf2 = v22; vf3 = v23
tf = t20
bf1 = k21; bf2 = k22; bf3 = k23

call computeNonLinearTerms(v21,v22,v23,k21,k22,k23,t20)

!b_n
do i = fstart(3), fend(3)
    do j = fstart(2), fend(2)
        do k = fstart(1), fend(1)
        index = i + j + k
        if(index.gt.3) then
            ct = cex(k,j,i)
```

```
            cv = P * ct
            cb = cv/Pm

            v31(k,j,i) = vftmp1(k,j,i) * &
            exp(cv * tau) + (2.0_rk * v21(k,j,i) - &
            v11(k,j,i)) * (exp(cv * tau) - 1.0_rk)/cv
            v32(k,j,i) = vftmp2(k,j,i) * &
            exp(cv * tau) + (2.0_rk * v22(k,j,i) - &
            v12(k,j,i)) * (exp(cv * tau) - 1.0_rk)/cv
            v33(k,j,i) = vftmp3(k,j,i) * &
            exp(cv * tau) + (2.0_rk * v23(k,j,i) - &
            v13(k,j,i)) * (exp(cv * tau) - 1.0_rk)/cv

            k31(k,j,i) = bftmp1(k,j,i) * &
            exp(cb * tau) + (2.0_rk * k21(k,j,i) - &
            k11(k,j,i)) * (exp(cb * tau) - 1.0_rk)/cb
            k32(k,j,i) = bftmp2(k,j,i) * &
            exp(cb * tau) + (2.0_rk * k22(k,j,i) - &
            k12(k,j,i)) * (exp(cb * tau) - 1.0_rk)/cb
            k33(k,j,i) = bftmp3(k,j,i) * &
            exp(cb * tau) + (2.0_rk * k23(k,j,i) - &
            k13(k,j,i)) * (exp(cb * tau) - 1.0_rk)/cb

            t30(k,j,i) = tftmp(k,j,i) * &
            exp(ct * tau) + (2.0_rk * t20(k,j,i) - &
            t10(k,j,i)) * (exp(ct * tau) - 1.0_rk)/ct
        endif
        enddo
    enddo
enddo

!F(b_n)
vf1 = v31; vf2 = v32; vf3 = v33
tf = t30
bf1 = k31; bf2 = k32; bf3 = k33

call computeNonLinearTerms(v31,v32,v33,k31,k32,k33,t30)

! u_n+1
do i = fstart(3), fend(3)
    do j = fstart(2), fend(2)
        do k = fstart(1), fend(1)
```

```
            index = i + j + k
            if(index.gt.3) then
                ct = cex(k,j,i)
                cv = P * ct
                cb = cv/Pm

                vf1(k,j,i) = vftmp1(k,j,i) * &
                exp(cv * tau) + v11(k,j,i) * c11(k,j,i) + &
                4.0_rk * v21(k,j,i) * c21(k,j,i) + v31(k,j,i) * c31(k,j,i)
                vf2(k,j,i) = vftmp2(k,j,i) * &
                exp(cv * tau) + v12(k,j,i) * c11(k,j,i) + &
                4.0_rk * v22(k,j,i) * c21(k,j,i) + v32(k,j,i)  * c31(k,j,i)
                vf3(k,j,i) = vftmp3(k,j,i) * &
                exp(cv * tau) + v13(k,j,i) * c11(k,j,i) + &
                4.0_rk * v23(k,j,i) * c21(k,j,i) + v33(k,j,i) * c31(k,j,i)

                bf1(k,j,i) = bftmp1(k,j,i) * &
                exp(cb * tau) + k11(k,j,i) * c12(k,j,i) + &
                4.0_rk * k21(k,j,i) * c22(k,j,i) + k31(k,j,i) * c32(k,j,i)
                bf2(k,j,i) = bftmp2(k,j,i) * &
                exp(cb * tau) + k12(k,j,i) * c12(k,j,i) + &
                4.0_rk * k22(k,j,i) * c22(k,j,i) + k32(k,j,i)  * c32(k,j,i)
                bf3(k,j,i) = bftmp3(k,j,i) * &
                exp(cb * tau) + k13(k,j,i) * c12(k,j,i) + &
                4.0_rk * k23(k,j,i) * c22(k,j,i) + k33(k,j,i) * c32(k,j,i)

                tf(k,j,i) = tftmp(k,j,i) * &
                exp(ct * tau) + t10(k,j,i) * c13(k,j,i) + &
                4.0_rk * t20(k,j,i) * c23(k,j,i) + t30(k,j,i) * c33(k,j,i)
            endif
            enddo
        enddo
    enddo

    call minusGrad(vf1,vf2,vf3)
    call minusGrad(bf1,bf2,bf3)

    ! Clean the arrays here
    call cleanFourierCoefficients2()

    itr = itr + 1
    now = now + tau
```

```
end

subroutine initCex(tau)
implicit none

real(kind = rk) :: tau
real(kind = 16)   :: cv, cb, ct, qtau

qtau = tau

do i = fstart(3), fend(3)
    do j = fstart(2), fend(2)
        do k = fstart(1), fend(1)

        cex(k,j,i) = -((alphax * dc(i))**2 + &
        (alphay * dcy(j))**2 + (pi * dcz(k))**2)
        ct = cex(k,j,i)
        cv = P * ct
        cb = cv/Pm

        ! velocity
        c11(k,j,i) = (-4.0_16 - qtau * cv + exp(cv * &
        qtau) * (4.0_16 - 3.0_16 * qtau * cv + qtau * &
        qtau * cv * cv))/(qtau * qtau * cv * cv * cv)
        c21(k,j,i) = (2.0_16 + qtau * cv + &
        exp(cv * qtau) * (-2.0_16 + qtau * cv)) / &
        (qtau * qtau * cv * cv * cv)
        c31(k,j,i) = (-4.0_16 - 3.0_16 * qtau * cv - &
        qtau * qtau * cv * cv + exp(qtau * cv) * &
        (4.0_16 - qtau * cv)) / &
        (qtau * qtau * cv * cv * cv)

        c12(k,j,i) = (-4.0_16 - qtau * cb + exp(cb * &
        qtau) * (4.0_16 - 3.0_16 * qtau * cb + qtau * &
        qtau * cb * cb))/(qtau * qtau * cb * cb * cb)
        c22(k,j,i) = (2.0_16 + qtau * cb + &
        exp(cb * qtau) * (-2.0_16 + qtau * cb)) / &
        (qtau * qtau * cb * cb * cb)
        c32(k,j,i) = (-4.0_16 - 3.0_16 * qtau * cb - &
        qtau * qtau * cb * cb + exp(qtau * cb) * &
        (4.0_16 - qtau * cb))/ &
        (qtau * qtau * cb * cb * cb)
```

```
        ! Temperature profile
        c13(k,j,i) = (-4.0_16 - qtau * ct + exp(ct * &
        qtau) * (4.0_16 - 3.0_16 * qtau * ct + qtau * &
        qtau * ct * ct))/(qtau * qtau * ct * ct * ct)
        c23(k,j,i) = (2.0_16 + qtau * ct + &
        exp(ct * qtau) * (-2.0_16 + qtau * ct)) / &
        (qtau * qtau * ct * ct * ct)
        c33(k,j,i) = (-4.0_16 - 3.0_16 * qtau * ct - &
        qtau * qtau * ct * ct + exp(qtau * ct) * &
        (4.0_16 - qtau * ct))/(qtau * qtau * ct * ct * ct)
        enddo
    enddo
enddo
end

subroutine computeNonlinearTermsForHeatTransfer(itf)
implicit none

complex(p3dfft_type), dimension(:,:,:), allocatable :: itf

do i = fstart(3),fend(3)
    do j = fstart(2),fend(2)
        do k = fstart(1),fend(1)
        aftmp1(k,j,i) = tf(k,j,i) * ci * dc(i) * alphax
        aftmp2(k,j,i) = tf(k,j,i) * ci * dcy(j) * alphay
        aftmp3(k,j,i) = tf(k,j,i) * dcz(k) * pi
        enddo
    enddo
enddo

! sine - sine - cosine
call backwardTransform(aftmp1,xtmp1,0)
call backwardTransform(aftmp2,xtmp2,0)
call backwardTransform(aftmp3,xtmp3,1)

xtmp1 = vp1 * xtmp1 + vp2 * xtmp2 + vp3 * xtmp3

! sine - sine - sine
call forwardTransform(xtmp1,aftmp1,0)

itf = -aftmp1 + vf3
```

```
end

subroutine computeNonLinearTerms(dvf1,dvf2,dvf3,dbf1,dbf2,dbf3,dtf)
implicit none

complex(p3dfft_type), dimension(:,:,:), allocatable :: dvf1,dvf2,dvf3
complex(p3dfft_type), dimension(:,:,:), allocatable :: dbf1,dbf2,dbf3
complex(p3dfft_type), dimension(:,:,:), allocatable :: dtf

! cosine - cosine - sine
call backwardTransform(vf1,vp1,1)
call backwardTransform(vf2,vp2,1)
call backwardTransform(vf3,vp3,0)

! cosine - cosine - sine
call backwardTransform(bf1,bp1,1)
call backwardTransform(bf2,bp2,1)
call backwardTransform(bf3,bp3,0)

call computeNonlinearTermsForNavierStokes(dvf1,dvf2,dvf3)
call computeNonlinearTermsForHeatTransfer(dtf)
call computeMagneticInduction(dbf1,dbf2,dbf3)
end

subroutine computeMagneticInduction(ibf1,ibf2,ibf3)
implicit none

complex(p3dfft_type), dimension(:,:,:),  allocatable :: ibf1,ibf2,ibf3

xtmp1 = 0.; xtmp2 = 0.; xtmp3 = 0.
do k = istart(3),iend(3)
    do j = istart(2),iend(2)
        do i = istart(1),iend(1)
        xtmp1(i,j,k) = vp2(i,j,k) * bp3(i,j,k) - vp3(i,j,k) * bp2(i,j,k)
        xtmp2(i,j,k) = vp3(i,j,k) * bp1(i,j,k) - vp1(i,j,k) * bp3(i,j,k)
        xtmp3(i,j,k) = vp1(i,j,k) * bp2(i,j,k) - vp2(i,j,k) * bp1(i,j,k)
        enddo
    enddo
enddo

! sine - sine - cosine
call forwardTransform(xtmp1,ftmp1,0)
```

```
call forwardTransform(xtmp2,ftmp2,0)
call forwardTransform(xtmp3,ftmp3,1)

call computeRot(ftmp1,ftmp2,ftmp3,ibf1,ibf2,ibf3,0)
end

! Dealiasing
subroutine filterCoefficients(fvf1,fvf2,fvf3,ftf)
implicit none

complex(kind = rk), dimension(:,:,:), allocatable :: ftf
complex(kind = rk), dimension(:,:,:), allocatable :: fvf1,fvf2,fvf3

do k = fstart(3),fend(3)
    if(k.ge.kfil+1.and.k.le.nx/2+1) then
        ! 1
        fvf1(fstart(1):fend(1),fstart(2):fend(2),k) = 0.
        fvf2(fstart(1):fend(1),fstart(2):fend(2),k) = 0.
        fvf3(fstart(1):fend(1),fstart(2):fend(2),k) = 0.

        ftf(fstart(1):fend(1),fstart(2):fend(2),k) = 0.
    endif
enddo

do k = fstart(2),fend(2)
    if(k.ge.kfil+1.and.k.le.nx/2+1) then
        ! 2
        fvf1(fstart(1):fend(1),k,fstart(3):fend(3)) = 0.
        fvf2(fstart(1):fend(1),k,fstart(3):fend(3)) = 0.
        fvf3(fstart(1):fend(1),k,fstart(3):fend(3)) = 0.

        ftf(fstart(1):fend(1),k,fstart(3):fend(3)) = 0.
    endif
enddo

do k = fstart(1),fend(1)
    if(k.ge.kfil+1.and.k.le.nz)then
        ! 3
        fvf1(k,fstart(2):fend(2),fstart(3):fend(3)) = 0.
        fvf2(k,fstart(2):fend(2),fstart(3):fend(3)) = 0.
        fvf3(k,fstart(2):fend(2),fstart(3):fend(3)) = 0.
```

```
            ftf(k,fstart(2):fend(2),fstart(3):fend(3)) = 0.
        endif
    enddo

    do k = fstart(2),fend(2)
        if(k.ge.nx/2+2.and.k.le.nx-kfil+1)then
            ! 4
            fvf1(fstart(1):fend(1),k,fstart(3):fend(3)) = 0.
            fvf2(fstart(1):fend(1),k,fstart(3):fend(3)) = 0.
            fvf3(fstart(1):fend(1),k,fstart(3):fend(3)) = 0.

            ftf(fstart(1):fend(1),k,fstart(3):fend(3)) = 0.
        endif
    enddo
    end
end
```