L A P O R A N

KEGIATAN MAGANG MAHASISWA

KODE SIMULASI MAGNETOHYDRODYNAMIC DENGAN VARIABLE MESH

DI

LEMBAGA PENERBANGAN DAN ANTARIKSA NASIONAL (LAPAN)

Jl. Dr Djunjunan 133 Bandung

Disusun Oleh:

Gesit Tali Singgih

M0213037

PROGRAM STUDI FISIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS SEBELAS MARET

AGUSTUS 2017

HALAMAN PENGESAHAN

1. Judul KMM	: Kode Simulasi Magnetohydrodynamic Dengan Variable Mesh
2. Peserta KMM	
a. Nama Lengkap	: Gesit Tali Singgih
b. NIM	: M0213037
c. Jenis Kelamin	: Laki-Laki
d. Semester	: 8
e. IPK Sekarang	: 3.24
f. Fakultas/Prodi	: MIPA/Fisika
g. Universitas	: Universitas Sebelas Maret
4. Tempat KMM	
a. Nama Instansi	: Lembaga Penerbangan dan Antariksa Nasional (LAPAN)
b. Alamat Instansi	: Jl. Dr. Djunjunan 133, Bandung
5. Waktu KMM	: 10 Juli-9 Agustus 2017

Bandung, 9 Agustus 2017

Peserta KMM

Gesit Tali Singgih

NIM: M0213037

Disetujui

Dosen Pembimbing KMM	Pembimbing Lapangan
Mohtar Yunianto S.Si., M.Si	**La Ode M. Musafar, M.Sc**
NIP. 19800630 200501 1 001	NIP. 19710930 199703 1 004

Mengetahui

Kepala Program Studi Fisika FMIPA UNS	Kepala Pusat Sains Antariksa LAPAN
Dr. Fahru Nurosyid S.Si, M.Si	**Dra. Clara Yono Yatini M.Sc**
NIP. 19721013 200003 1 002	NIP. 19640309 199007 2 001

KATA PENGANTAR

Terimakasih banyak kepada seluruh pihak yang telah mendukung agar pelaksanaan KMM dapat berlangsung dengan baik. Dalam kesempatan kali ini penulis menyampaikan penghargaan dan terimakasih kepada:

1. Bapak La Ode M. Musafar, M.Sc selaku pembimbing magang di LAPAN Bandung
2. Ibu Dra. Clara Yono Yatini M.Sc. selaku Kepala LAPAN Bandung atas izinnya untuk penulis melaksanakan kuliah magang di LAPAN Bandung.
3. Bapak Dr. Fahru Nurosyid, S.Si.,M.Si selaku Kepala Program Studi Fisika FMIPA UNS.
4. Bapak Mohtar Tunianto S.Si., M.Si selaku dosen pembimbing magang dari Program studi Fisika FMIPA UNS.
5. Ibu Dr.Eng.Kusumandari S.Si, M.Si selaku coordinator magang di Fisika UNS
6. Bapak Drs. Bambang Suhandi selaku Kepala Bidang Diseminasi PUSSAINSA LAPAN.
7. Teman Teman Fisika 2013 yang selalu memberi dukungan

Penulis telah berusaha untuk melakukan yang terbaik dalam penulisan laporan ini, namun penulis menyadari masih terdapat banyak kekurangan yang harus diperbaiki. Oleh karenanya kritik dan saran sangat penulis harapkan untuk perbaikan.

Semoga laporan ini dapat memberikan manfaat bagi pihak yang ingin belajar mengenai magnetohydrodynamic maupun dinamika fluida secara umum.

DAFTAR ISI

HALAMAN PENGESAHAN....ii
KATA PENGANTAR....iii
DAFTAR ISI....iv
DAFTAR GAMBAR....vii
DAFTAR TABEL....viii
BAB I PENDAHULUAN....1
1.1 Latar Belakang....1
1.2 Rumusan Masalah....2
1.3 Tujuan KMM....2
1.4 Manfaat KMM....3
1.4.1 Bagi Mahasiswa....3
1.4.2 Bagi Universitas Sebelas Maret....3
1.4.3 Bagi Lembaga Penerbangan dan Antariksa Nasional (LAPAN)....3
BAB II LANDASAN TEORI....4
2.1 Tinjauan Umum Instansi....4
2.1.1 Sejarah LAPAN....4
2.1.2 Struktur Organisasi....5
2.2 Tinjauan Pustaka....5
2.2.1 Solar Flare....5
2.2.2 Karakteristik Solar Flare....6
2.2.3 Proses Terjadinya Flare....8
2.2.4 SDO (Solar Dynamics Observatory)....9
2.2.5 Sunspot....11
BAB III METODE PELAKSANAAN....14
3.1 Tempat dan Waktu Kegiatan....14
3.1.1 Tempat Kegiatan....14
3.1.2 Waktu Kegiatan....14
3.2 Sumber Data....14

BAB IV HASIL DAN PEMBAHASAN.....18
BAB V PENUTUP.....25
5.1 Kesimpulan.....25
DAFTAR PUSTAKA.....26

DAFTAR GAMBAR

Gambar 2.1 Struktur Organisasi.....5
Gambar 2.2 Proses fisis terjadinya flare dan fenomena yang terkait dengan flare..8
Gambar 2.3 Solar Dynamics Observatory (SDO).....10

BAB I
PENDAHULUAN

1.1 Latar Belakang

Sebagian besar permasalah fisika adalah system yang kompleks dimana banyak variable yang saling berinteraksi untuk mempengaruhi evolusi satu sama lain. Terlebih lagi apabila variable yang dimaksud adalah suatu medan yang memiliki nilai unik di setiap titik ruang. Apabila evolusi medan dipengaruhi oleh beberapa medan yang lain, dan apabila evolusi mereka membentuk suatu *feedback loop* maka penyelesaian secara analitik menjadi mustahil kecuali untuk beberapa kasus khusus. masalahan seperti ini memerlukan metode numeric. Dengan metode numeric, masalahan yang besar dengan banyak variable dapat dipecahkan menjadi unsur unsur yang lebih jauh lebih kecil yang dapat diselesaikan dengan mudah.

Masalah dinamika fluida merupakan masalah kompleks yang memiliki banyak aplikasi baik dalam fisika dan engineering, seperti dalam perhitungan aliran udara di sekitar pesawat, mobil, kincir angin, *propeller* kapal, aliran air dalam pipa, simulasi iklim, hingga penelitian mengenai proses pembentukan bintang, dan fisi nuklir. Persamaan yang mendeskripsikan fluida adalah set persamaan Navier-Stokes.

Magnenohydrodynamic adalah sub disiplin dinamika fluida yang mempelajari fluida yang konduktif secara elektrik. Contoh fluida magneto adalah plasma, metal cair, dan elektrolit. Inti dari Magnenohydrodynamic adalah medan magnet akan menghasilkan arus dalam fluida konduktif yang mengalir, selanjutnya arus akan mempengaruhi medan magnet itu sendiri dan menghasilkan suatu *feedback loop*. Magnenohydrodynamic dideskripsikan menggunakan persamaan Navier-Stokes dan persamaan Maxwell. Kedua set persamaan ini harus diselesaikan secara bersama-sama untuk menyelesaikan permasalahan magnenohydrodynamik. Dapat dibayangkan bahwa suatu system yang harus mematuhi persamaan Maxwell dan bergerak seperti fluida adalah system yang

sangat kompleks dimana setiap besaran fisis seperti tekanan, kecepatan, massa jenis, medan magnet, arus, dan medan listrik akan memiliki nilai yang bervariasi dalam ruang dan akan berubah terhadap waktu mengikuti nilai variable yang lain. Namun walaupun sulit untuk dipecahkan, banyak aplikasi scientific yang sangat penting dapat terbantukan dengan simulasi magnenohydrodynamik, seperti masalah plasma *inertial confinement* dalam reactor fusi, evolusi bintang dan galaksi, dan masalah geomagnet.

Setiap simulasi numeric harus mempertimbangkan akurasi dan kapasitas infrastruktur komputasi. Semakin kecil elemen dalam simulasi makan semakin baik akurasi, namun hal ini akan menambah beban komputasi karena semakin banyak juga jumlah elemen yang harus dihitung. Sebagai ilustrasi apabila kita ingin mensimulasikan magnetosfer Bumi dengan grid 3 dimensi berbentuk box dengan lebar 1 juta km dan resolusi 1 km maka kita membutuhkan elemen sebanyak 1jutaX1jutaX1juta, setiap variable dalam persamaan Navier-Stokes dan Maxwell harus direpresentasikan dengan 10^{18} angka. Dalam simulasi numerik angka sebanyak itu membutuhkan 1 milyar Gigabyte RAM apabila satu elemen memakan 1 byte, dan ini hanya untuk satu variable, hal ini tentu di luar kapasitas bahkan supercomputer tercepat di dunia (Tianhee-2 dengan kapasitas RAM 1 juta Gigabyte).

Agar simulasi dapat mendapatkan resolusi yang bagus sekaligus dalam kapasitas komputasi computer yang ada maka dibutuhkan teknik tambahan. Salah satu teknik yang ada adalah *variable grid.* Prinsip teknik ini adalah dengan mengunakan resolusi yang baik untuk daerah daerah khusus yang paling dinamik dan menggunakan resolusi normal untuk daerah yang kurang dinamik. Misalkan problem magnetosfer Bumi, resolusi baik digunakan untuk area ekor magnetosfer dan di dekat Bumi, lalu resolusi normal untuk area lain. Teknik ini memungkinkan komputasi yang kompleks untuk dilakukan bukan hanya menggunakan supercomputer, namun Personal Computer biasa.

Disini kami membangun kode numeric untuk mensimulasikan magnenohydrodynamic menggunakan variable grid menggunakan Numerical Python(Numpy). Sebagian besar pekerjaan disini dibangun berdasarkan tesis Ryan Farber berjudul Magnenohydrodynamic Modeling in Python [ref]. Diharapkan pekerjaan kami dapat digunakan sebagai pengenalan mahasiswa lain terhadap magnenohydrodynamic dan fluid dynamic secara umum.

1.2 Rumusan Masalah

1. Bagaimana melakukan simulasi numeric Magnenohydrodynamic ?
2. Bagaimana cara membangun kode simulasi dengan variable mesh ?

1.3 Tujuan KMM

1. Mendapat gambaran mengenai situasi dan kondisi dunia kerja khususnya di Lembaga Penerbangan dan Antariksa Nasional Bandung.
2. Mengaplikasikan teori yang telah didapatkan di perkuliahan dan menerapkannya dalam dunia kerja.
3. Mengetahui dinamika fluida konduktif dalam medan magnet
4. Mengetahui perilaku plasma di sekitar medan magnet bumi

1.4 Manfaat KMM

1.4.1 Bagi Mahasiswa

1. Mahasiswa dapat menerapkan ilmu yang didapatkan selama perkuliahan pada lembaga tempat Kegiatan Kuliah Magang (KMM) dilaksanakan.
2. Menyiapkan diri dalam menghadapi dunia kerja pada masa yang akan datang.
3. Meningkatkan pemahaman ilmu pengetahuan yang terkait dengan Sains Antariksa

1.4.2 Bagi Universitas Sebelas Maret

1. Sebagai sarana pengembangan IPTEK, khususnya di dunia Fisika sebagai bahan pertimbangan dalam penyusunan program bagi Universitas Sebelas Maret

2. Sebagai bahan masukan dan evaluasi program pendidikan di UNS untuk menghasilkan tenaga-tenaga terampil sesuai kebutuhan industri dalam bidang masing- masing.
3. Memperoleh informasi tentang perkembangan dunia Sains Antariksa beserta elektronika dan instrumentasinya di Lembaga Penerbangan dan Antariksa Nasional (LAPAN).

1.4.3 Bagi Lembaga Penerbangan dan Antariksa Nasional (LAPAN)

1. Untuk menjalin kerjasama dengan dunia Pendidikan Perguruan Tinggi, khususnya Universitas Sebelas Maret
2. Sebagai sarana penyebarluasan ilmu pengetahuan dan teknologi di bidang Sains Antariksa bagi dunia pendidikan, khususnya pada kalangan mahasiswa.
3. Sebagai sarana informasi terkait dengan kriteria tenaga kerja yang dibutuhkan di Pusat Sains Antariksa- Lembaga Penerbangan dan Antariksa (LAPAN)

BAB II
LANDASAN TEORI

2.1 Tinjauan Umum Instansi

2.1.1 Sejarah LAPAN

Pada tanggal 31 Mei 1962, dibentuk Panitia Astronautika oleh Menteri Pertama RI, Ir. Juanda (selaku Ketua Dewan Penerbangan RI) dan R.J. Salatun (selaku Sekretaris Dewan Penerbangan RI).

Pada tanggal 22 September 1962, terbentuknya Proyek Roket Ilmiah dan Militer Awal (PRIMA) afiliasi AURI dan ITB. Berhasil membuat dan meluncurkan dua roket seri Kartika berikut telemetrinya.

Pada tanggal 27 November 1963, Lembaga Penerbangan dan Antariksa Nasional (LAPAN) dibentuk dengan Keputusan Presiden Nomor 236 Tahun 1963 tentang LAPAN.

Penyempurnaan organisasi LAPAN melalui :

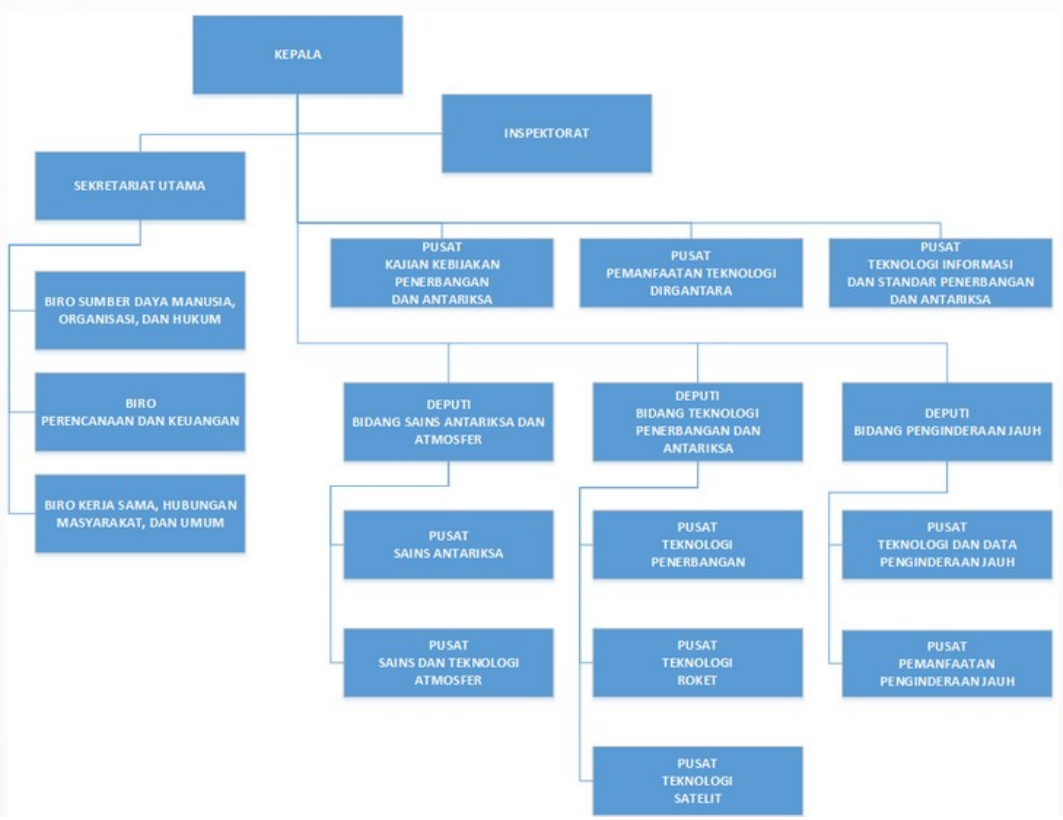
1. Keputusan Presiden (Keppres) Nomor 18 Tahun 1974,
2. Keppres Nomor 33 Tahun 1988,
3. Keppres Nomor 33 Tahun 1988 jo Keppres Nomor 24 Tahun 1994;
4. Keppres Nomor 166 Tahun 2000 sebagaimana diubah beberapa kali yang terakhir dengan Keppres Nomor 4 Tahun 2013
5. Perpres Nomor 49 Tahun 2015.

Kompetensi Utama:

1. Sains Antariksa dan Atmosfer;
2. Teknologi penerbangan, roket, dan satelit;
3. Penginderaan jauh;
4. Kajian Kebijakan Penerbangan dan Antariksa.

2.1.2 Struktur Organisasi

Berikut merupakan struktur organisasi LAPAN secara umum:



Gambar 2.1 Struktur Organisasi

LAPAN Bandung di Kepalai oleh Dra. Clara Yono Yatini, M.Sc.

2.2 Tinjauan Pustaka

2.2.1 Fluida

Fluida adaalah material yang tidak mmiliki bentuk tetap yang rigid. Artinya fluida dapat secara kontinu mengalami perubahan bentuk sesuai dengan gaya yang diterapkan kepadanya. Peristiwa perubahan bentuk secara kontinu ini dinamakan aliran.

2.2. 2 Persamaan Fluida

A. Persamaan Kontinuitas

Fluida mematuhi persamaan kontinuitas. Persamaan kontinuitas mendeskripsikan suatu kuantitas yang mengalami konservasi, intinya jumlah total

dari sesuatu di keseluruhan system adalah konstan. Dalam fluida persamaan kontinuitas menyatakan bahwa jumlah massa yang memasuki system sama dengan jumlah massa yang meninggalkan system. Bentuk differesial dari persamaan kontinuitas adalah sebagai berikut:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

ρ = densitas fluida

t = waktu

u = vector kecepatan

untuk kasus fluida yang inkompresible dimana densitas konstan, maka perubahan densitas sama dengan nol, maka persamaan kontinuitas disederhanakan menjadi berikut:

$$\nabla \cdot \mathbf{u} = 0,$$

Persamaan ini menyatakan bahwa divergen kecepatan adalah nol dimanapun. Pada bagian simulasi kami menggunakan teknik relaksasi untuk menjaga agar kecepatan divergenya tetap nol.

B. Persamaan Navier-Stokes

Persamaan Navier-Stokes mendeskripsikan gerakan fluida yang memiliki viskositas. Persamaan ini didapat dengan mengkombinasikan hukum newton 2 tentang akselerasi dengan gerakan fluida yang diasumsikan akselerasinya berasal dari tekanan dan difusi. Persamaannya adalah:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{\nabla p}{\rho} + \eta_\nu \nabla^2 \mathbf{u}$$

$(\mathbf{u} \cdot \nabla)\mathbf{u}$ disebut dengan adveksi, berdiri sendiri adveksi memiliki bentuk yang sama dengan persamaan gelombang. Persamaan gelombang memiliki property dimana bentuk gelombang yang merambat akan selalu tetap, unsur yang sama juga beroprasi dalam persamaan fluida, medan kecepatan dalam fluida akan merambat dalam fluida seperti gel$\eta_\nu \nabla^2 \mathbf{u}$, bentuknya akan tetap dan hanya

lokasinya yang berubah ~~sesuai~~ waktu. $\frac{\nabla p}{\rho}$ adalah difusi momentum, hal ini ditimbulkan oleh viskositas fluida, dapat dibayangkan apabila ada satu titik dalam fluida yang bergerak dan titik disekitarnya diam, maka dikarenakan gaya antar molekul fluida titik di sekitarnya akan ikut mengalami pergerakan. Adalah kontribusi dari tekanan, perbedaan tekanan antara satu titik dengan titik yang lain, yang di hitung dengan divergen akan menimbulkan net gaya kea rah titik yang tekanannya lebih kecil. Jadi akselerasi fluida di sebabkan oleh adveksi, difusi dan perbedaan tekanan.

2.2.3 Persamaan Magnenohydrodynamic

A. Evolusi Medan Magnet

Persamaan magnenohydrodynamic menginporasikan efek magnetic terhadap persamaan fluida. Artinya perlu ditambahkan persamaan Maxwell. Dalam simulasi ini kami membatasi terhadap persamaan dengan resistivitas nol dan mengabaikan arus hall, selain itu konstanta kecepatan cahaya, premeabilitas magnetic dan permitivitas listrik di set sama dengan satu. Maka dari itu persamaan Maxwell menjadi:

$$\nabla \cdot \mathbf{B} = 0$$

$$\nabla \cdot \mathbf{E} = \frac{q}{V}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$

$$\nabla \times \mathbf{B} = \mathbf{j}$$

Selanjutnya eliminasi densitas arus **j** dengan menulisnya dalam bentuk hukum ohm:

$$\mathbf{j} = \sigma \mathbf{E}'$$

Selanjutya medan listrik E adalah jumlah dari medan listrik static dan curl dari **B** yang merupakan hukum faraday**.**

$$\mathbf{E}' = \mathbf{E} + \mathbf{u} \times \mathbf{B}$$

Selanujutnya masukan nilai ini ke dalam hukum ohm .

$$\mathbf{j} = \sigma(\mathbf{E} + \mathbf{u} \times \mathbf{B})$$

Substitusikan persamaan diatas ke dalam persamaan Maxwell ke empat, lalu kedua sisi di curl, maka akan didapatkan:

$$\nabla \times (\nabla \times \mathbf{B}) = \sigma(\nabla \times \mathbf{E}) + \sigma(\nabla \times (\mathbf{u} \times \mathbf{B}))$$

Curl medan listrik lalu diganti dengan persamaan Maxwell ketiga, akan didapat:

$$\nabla \times (\nabla \times \mathbf{B}) = -\sigma \frac{\partial \mathbf{B}}{\partial t} + \sigma(\nabla \times (\mathbf{u} \times \mathbf{B}))$$

Menggunakan persamaan identitas dalam vector kalkulus maka persamaan diatas dapat diubah menjadi:

$$\nabla \times (\nabla \times \mathbf{B}) = \nabla(\nabla \cdot \mathbf{B}) - \nabla^2 \mathbf{B}$$

Curl **B** sama dengan nol, maka laplacian dari **B** dapat dimasukan ke persamaan sebelumnya, dengan itu kita dapat mendapatkan unsur perubahan **B** terhadap waktu dengan persamaan yang lebih sederhana:

$$-\nabla^2 \mathbf{B} = -\sigma \frac{\partial \mathbf{B}}{\partial t} + \sigma(\nabla \times (\mathbf{u} \times \mathbf{B}))$$

Perubahan **B** terhadap waktu dapat ditulis sebagai berikut:

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{u} \times \mathbf{B}) + \eta_\rho \nabla^2 \mathbf{B}$$

Dimana eta adalah satu dibagi dengan konduktivitas. Curl dari curl medan kecepatan terhadap medan magnet berarti medan magnet mengalami adveksi, sedangkan laplacian dari medan magnet mengatakan bahwa medan magnet mengalami difusi seiring berjalanya waktu.

Persamaan diatas diekstrak dari persamaan Maxwell, namun hingga tahap ini belum semua persamaan Maxwell terdeskripsikan oleh persamaan diatas,

terutama adalah divergen dari **B** yang belum terjamin sama dengan nol. Untuk menjamin **B** sama dengan nol kami menggunakan vector potenaial **A** untuk menghitung medan magnet. Vector potennsial mengikuti persamaan berikut:

$$\nabla \times \mathbf{A} = \mathbf{B}$$

Lalu subtitusikan persamaan diatas dengan persamaan sebelumnya:

$$\frac{\partial(\nabla \times \mathbf{A})}{\partial t} = \nabla \times (\mathbf{u} \times (\nabla \times \mathbf{A})) + \eta_\rho \nabla^2 (\nabla \times \mathbf{A})$$

Waktu dan ruang bersifat orthogonal, sehingga curl dapat dan turunan terhadap waktu dapat bertukar posisi:

$$\nabla \times \frac{\partial \mathbf{A}}{\partial t} = \nabla \times (\mathbf{u} \times (\nabla \times \mathbf{A})) + \eta_\rho \nabla \times (\nabla^2 \mathbf{A})$$

Mengguankan identitas vector kalkulus persamaan diatas dapat dirubah menjadi:

$$\nabla \times \frac{\partial \mathbf{A}}{\partial t} = -\nabla \times \left((\mathbf{v} \cdot \nabla)\mathbf{A} + \eta_\rho \nabla^2 \mathbf{A} \right)$$

Setelah itu curl dari kedua sisi dihilangkan, dan didapatkan:

$$\frac{\partial \mathbf{A}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{A} = \eta_\rho \nabla^2 \mathbf{A}$$

Ternyata medan magnet dapat dihitung menggunakan vector potensial, selain itu evolusi vector potensial adalah adveksi dan difusi. Dengan persamaan ini dapat dijamin system persamaan Maxwell terpenuhi.

B. Tekanan Magnetik

Di bagian ini dievaluasi peranan medan maget terhadap gerakan fluida. Penutunan dimulai dari gaya Lorentz terhadap muatan yang bergerak dalam medan electromagnet:

$$\mathbf{F} = q\mathbf{E} + q\mathbf{u} \times \mathbf{B}$$

Dalam simulasi ini kami membatasi konduktivitas sangat besar, konduktivitas besar artinya konduktor dapat memindahkan muatan dengan sangat cepat,

sehingga dalam simulasi medan listrik secara instan menjadi nol karena muatan bergerak saling menetralkan, sehingga dalam kasus kami gaya Lorentz menjadi:

$$\mathbf{F} = q\mathbf{u} \times \mathbf{B}$$

Muatan dikali dengan medan kecepatan apabila dibagi dengan massa akan mendapatkan densitas arus. Maka apabila kita memasukan hukum newton kedua didapatkan:

$$\mathbf{a} = \mathbf{j} \times \mathbf{B}$$

Masukan persamaan Maxwell dan gantikan **j** dengan medan magnet dan menggunakan identitas vector kalkulus maka didapatkan:

$$\mathbf{a} = -\nabla\left(\frac{B^2}{2}\right) + (\mathbf{B} \cdot \nabla)\mathbf{B}$$

Persamaan ini dimasukan ke dalam persamaan Navier-Stoke untuk mendapatkan kontribusi medan magnet terhadap medan kecepatan. Term yang pertama adalah tekanan magnetic, sedangkan term kedua disebut dengan curvature term magnetic.

2.2.4 Persamaan Magnetohydrodynamic

Hingga tahap ini setiap variable dalam persamaan Navier-Stokes dan Maxwell telah diinkorporasikan terhadap set persamaan berikut:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{\nabla p}{\rho} + \eta_\nu \nabla^2 \mathbf{u} + (\mathbf{B} \cdot \nabla)\mathbf{B} - \nabla\left(\frac{B^2}{2}\right)$$

$$\frac{\partial \mathbf{A}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{A} + \eta_\rho \nabla^2 \mathbf{A}$$

$$\nabla \times \mathbf{A} = \mathbf{B}$$

$$\nabla \cdot \mathbf{B} = 0$$

$$\nabla \cdot \mathbf{u} = 0$$

Set persamaan diatas cukup untuk digunakan untuk mensimulasikan magneto fluid. Dalam area simulasi keenam persamaan diatas harus selalu terpenuhi. Untuk memenuhi semuanya, digunakan metode finite difference.

2.2.5 Mendiskritkan Persamaan

Finite diference dilakukan dengan cara sebagai berikut, apabila ada variable u yang di turunkan terhadap x, maka:

$$\frac{\partial u}{\partial x} \approx \frac{u(x+\Delta x) - u(x)}{\Delta x}$$

Sedangkan turunan kedua adalah:

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}$$

Untuk melakukan simulasi numeric seluruh persaman magnenohydrodynamic dipisahkan ke komponen vektornya. Dalam proyek ini dilakukan simulasi dua dimensi untuk semua variable kecuali potensial vector dan medan magnet. Potensial vector memiliki komponen arah z dan nol untuk x dan y, sehingga medan magnet nol untuk z dan tidak nol untuk arah x dan y, sedangkan variable lain berada di arah x dan y.

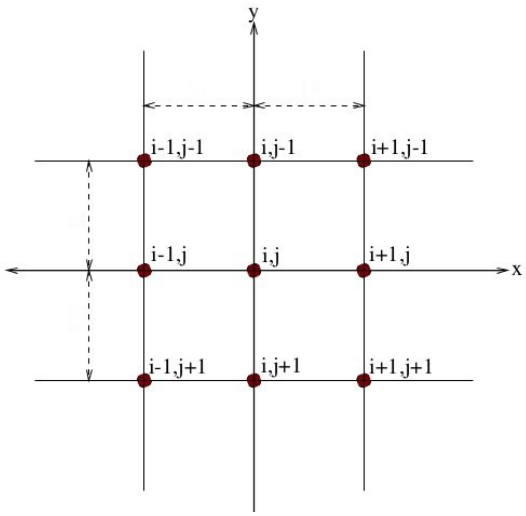
2.2.6 Metode Relaksasi

untuk menjaga agar medan kecepatan mengikuti prsamaan kontinuitas, medan tekanan harus berubah sehingga divergen kecepatan tetap nol. Hal ini dicapai apabila medan tekanan mematuhi persamaan berikut:

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = -\rho \left(\frac{\partial u}{\partial x}\frac{\partial u}{\partial x} + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial v}{\partial y} \right)$$

Persaman diatas adalah persamaan poisson untuk tekanan. Persamaan poisson diselesaikan dengan metode relaksasi. Metode ini sama saja dengan menghitung suatu titik berdasarkan rata rata titik di sekitarnya.

2.2.7 Skema Metode Numerik



Gambar 2.1 Skema simulasi numerik

Metode numeric menggunakan skema seperti diatas. Bidang simulasi dibagi menjadi cel cel kecil. Setiap cell mengikuti koordinat i dan j, untuk menghitung nilai variabel di setiap titik digunakan metode pendiskritan persamaan seperti di bagian 2.25

2.2.8 Variable Mesh

Variable mesh yakni mengaplikasikan resolusi bagus untuk daerah yang dinamik. Untuk melakukan seleksi terhadap daerah yang dinamik, dilakukan wavelet compression terhadap seluruh domain simulasi, domain yang memiliki koefisien yang nilainya memenuhi kriteria tertentu lalu dimasukan dalam subprogram untuk menaikan reslolusi simulasi. Dalam beberapa step simulasi dilakukan proses peningkatan resolusi ini. Selain itu dilakukan juga proses penurunan resolusi apabila daerah menjadi kurang dinamik dan tidak memerlukan resolusi yang bagur.

BAB III
METODE PELAKSANAAN

3.1 Tempat dan Waktu Kegiatan

3.1.1 Tempat Kegiatan

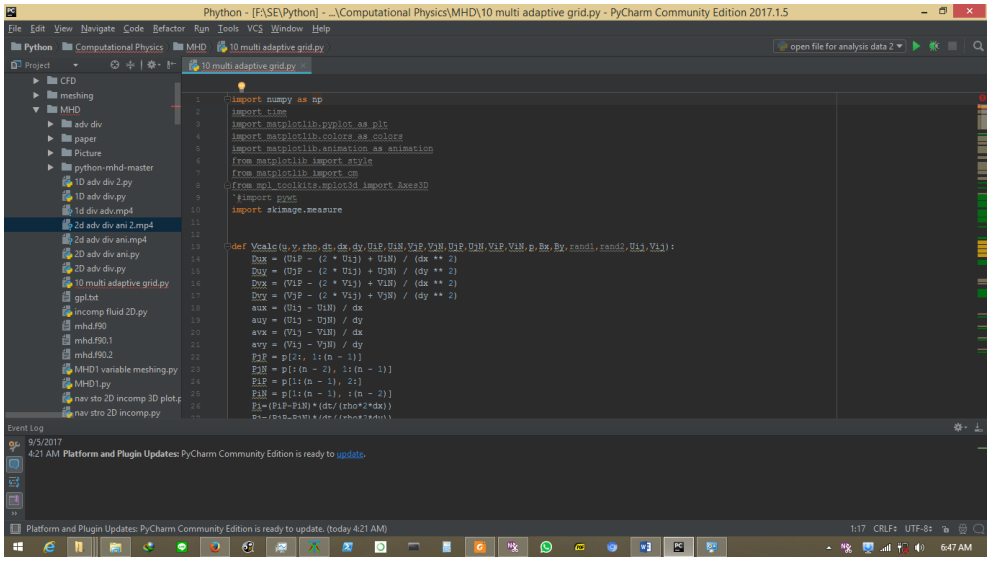
Penelitian dan simulasi dilakukan di Lembaga Penerbangan dan Antariksa Nasional (LAPAN) Deputi Bidang Sains Pengkajian dan Informasi Kedirgantaraan Pusat Sains dan Antariksa Jl. Dr Djunjunan 133 Bandung, Jawa Barat.

3.1.2 Waktu Kegiatan

Penelitian dilaksanakan dalam 1 bulan mulai dari tanggal 10 Juli 2017 sampai dengan tanggal 9 Agustus 2017.

3.2 Pembuatan Kode

Kode ditulis menggunakan bahasa Python. Untuk mempercepat komputasi digunakan modul numerical python(Numpy), yakni modul khusus untuk simulasi numeric dari python. Modul untuk melakukan plot data menggunakan matplotlib. Untuk mengatur proyek dihunakan program Pycharm, berikut adalah gambar pembuatan program. Program dilampirkan di lampian 1.



Gambar 3.1 Program yang digunakan untuk menulis simulasi

3.3 Metode Running Simulasi

Simulasi dilakukan menggunakan komputer pembimbing. Simulasi memakan waktu total selama 24 jam. Individual frame memakan waktu sekitar satu detik untuk diprosses. Sebagian besar tenaga komputasi digunakan dalam bagian perhitungan tekanan dengan menggunakan metoide relaksasi.

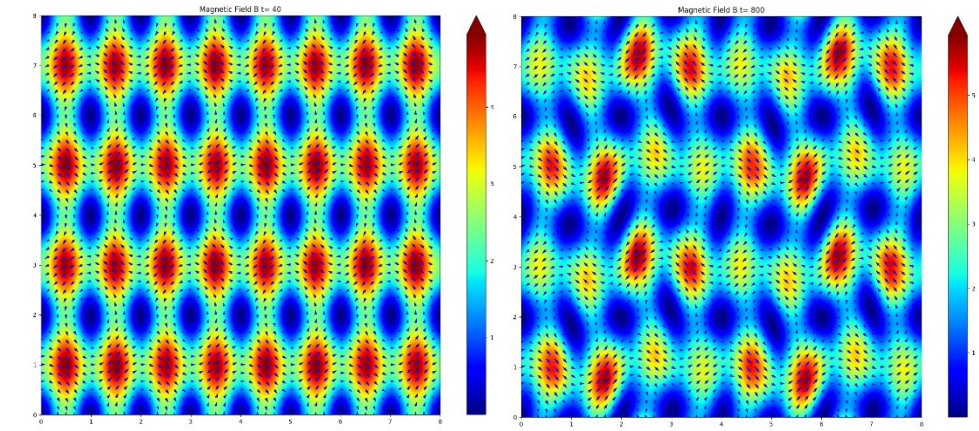
BAB IV
HASIL DAN PEMBAHASAN

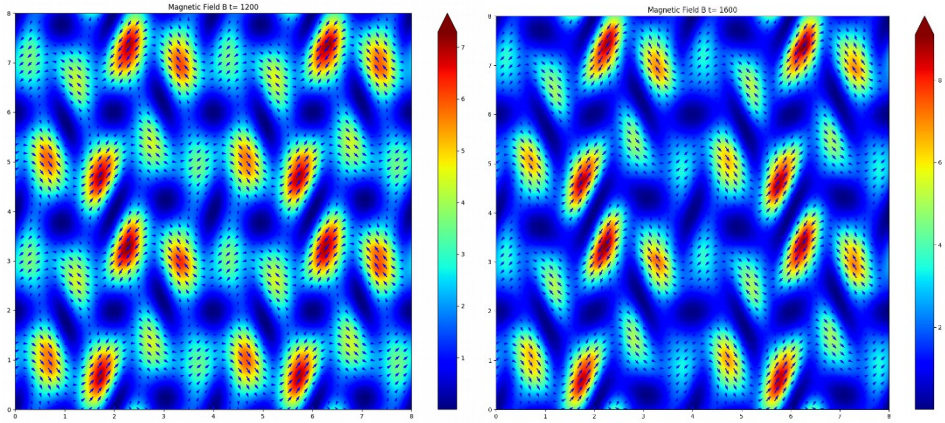
Magnetohydrodynamic atau MHD adalah studi sifat magnetic dari fluida konduktif, seperti plasma, metal cair, dan elektrolit. MHD dimulai oleh Hannes Alven, dimana beliau menerima penghargaan nobel atas deskripsinya terhadap gelombang dalam fluida MHD yang dinamakan gelombang alven.

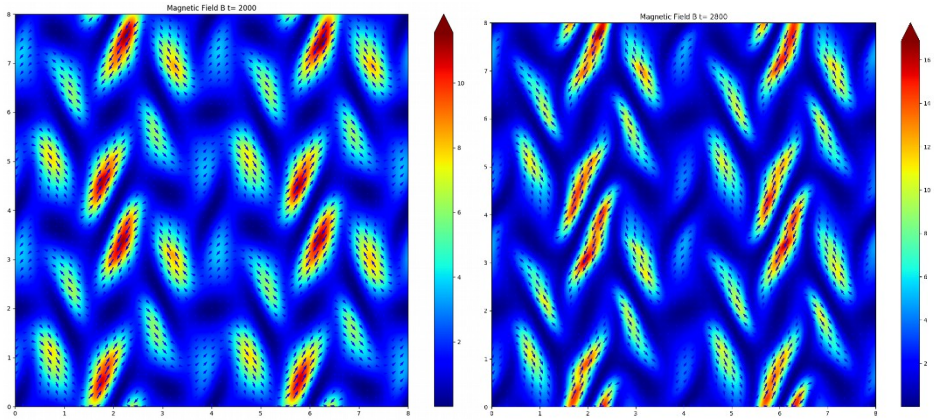
Setelah kode dibuat secara lengkap dan dapat berjalan dengan sempurna, dilakukan simulasi dan data disimpan. Di bawah ini adalah gambar evolusi medan maget, warna menunjukan kerapatan energi medan dan panah menunjukan arah medan magnet. Dapat dilihat bahwa medan magnet selalu berubah sesuai persamaan magnetohydrodynamic. Disini dapat diamati proses adveksi dan difusi medan magnet, setiap area dengan densitas medan yang tinggi akan bergerak mempertahankan bentuknya, dan di samping itu juga mengalami difusi. Tentu saja setiap bagian saling beriteraksi dengan tetangganya dan membentuk perilaku yang kompleks.Dalam gambar dapat juga dilihat proses kompresi medan magnet, dimana ada area yang semakin lama semakin memipih dan mengalami elongasi
Untuk simulasi ini digunakan parameter sebagai berikut:

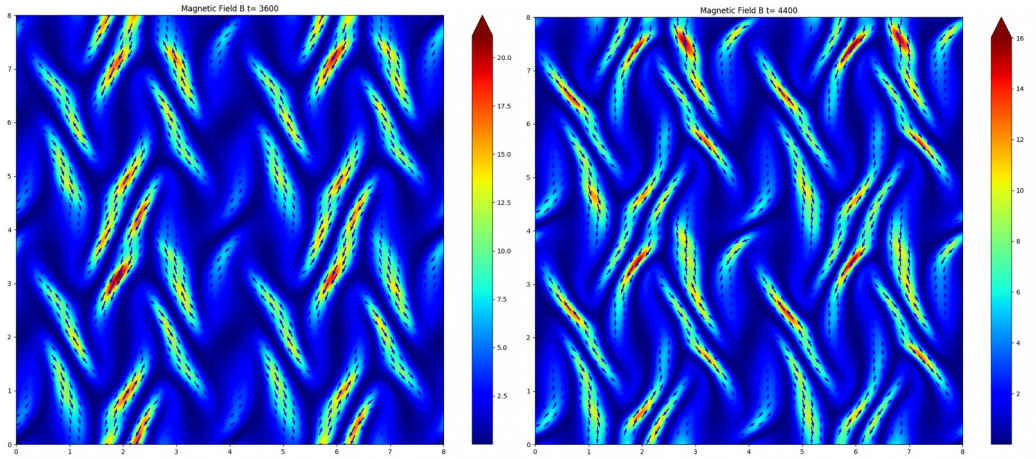
Vx	=-sin(Y)
Vy	= sin(X)
A	= cos(Y)+0.5cos(2X)
Koefisien diffusi medan kecepatan	= 0.1
Kowfisien diffuse potensial vector	= 0.1
Densitas	= 1 (uniform dan time invariant)

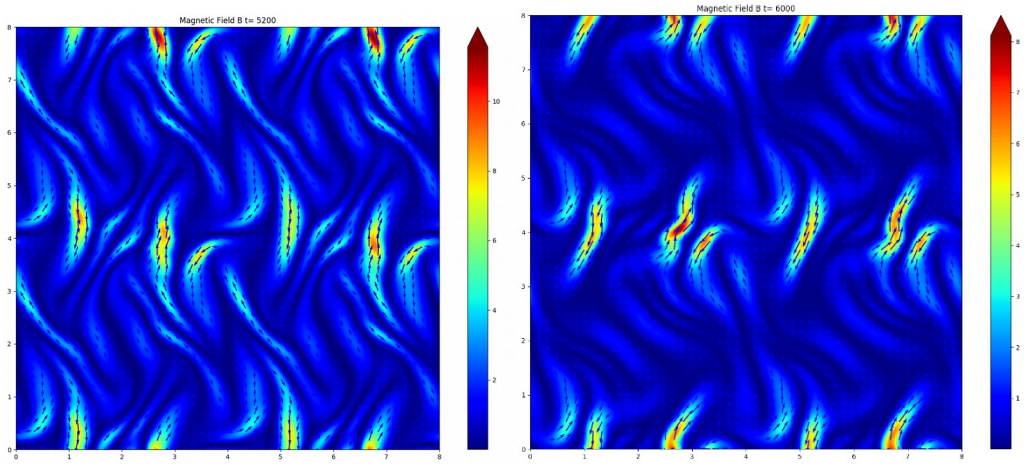
Kasus seperti ini dinamakan Orzag-Tang Vertex, biasanya digunakan untuk mengetest kose simulasi magnetohydrodynamic dalam mengatasi turbulensi dan shock dalam magneto fluida. Karena persamaan yang digunakan menggunakan asumsi incompressible fluid, maka kode hanya akurat di system yang gradient tekananya rendah, Kode simulasi diikutsrtakan dalam lampiran 1

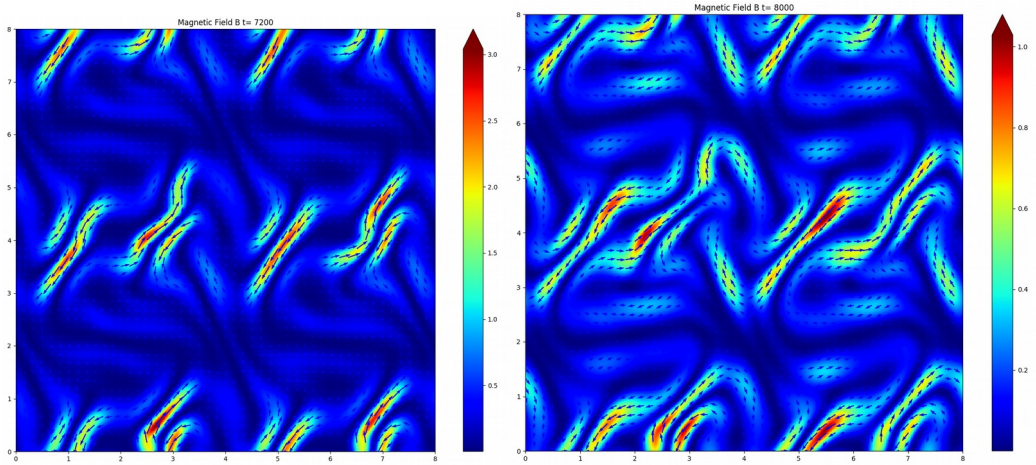

Magnetic Field B t= 40
Magnetic Field B t= 800

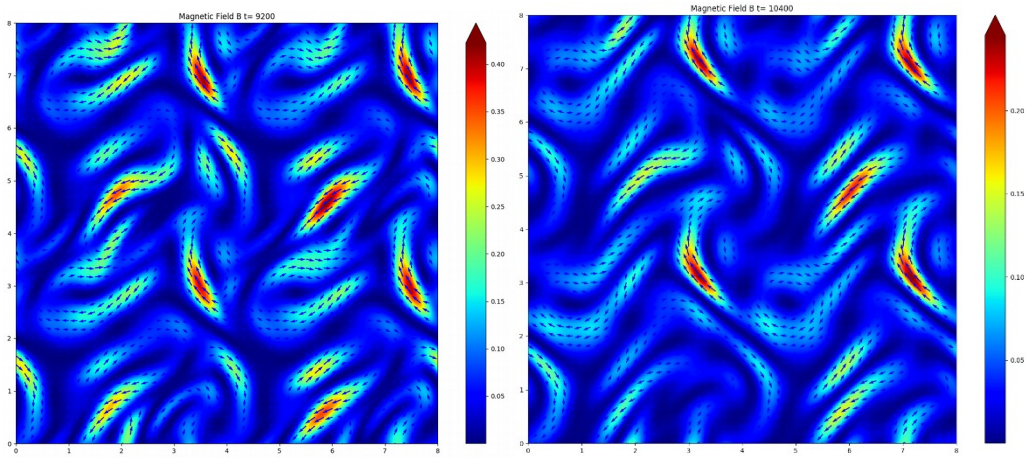

Magnetic Field B t= 1200
Magnetic Field B t= 1600

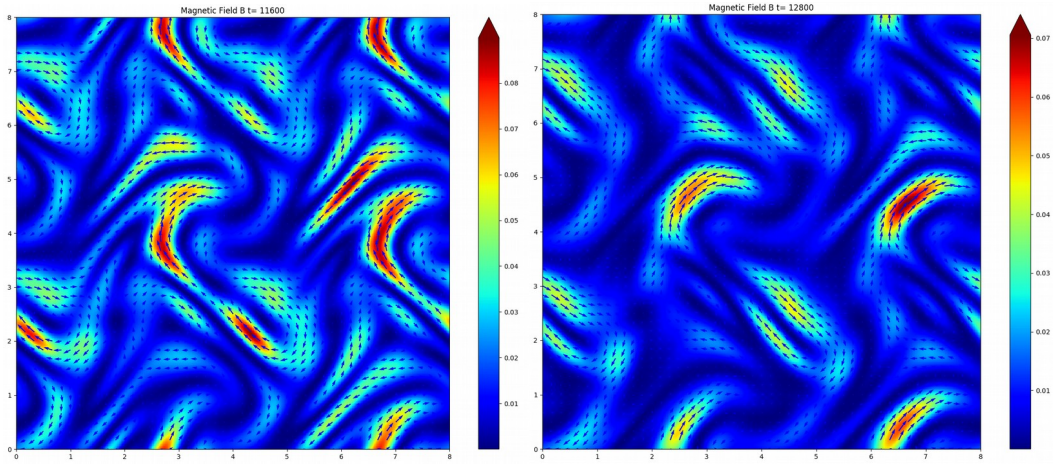

Magnetic Field B t= 2000
Magnetic Field B t= 2800

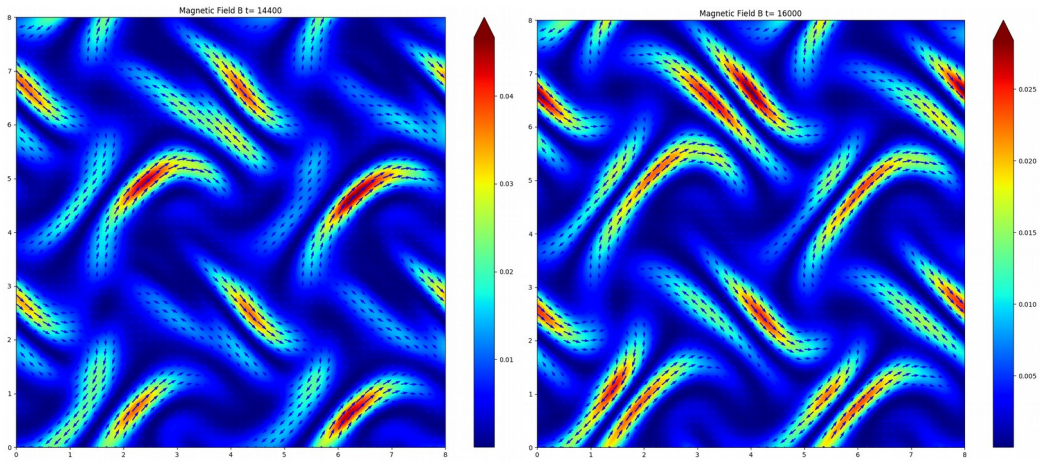

Magnetic Field B t= 3600
Magnetic Field B t= 4400

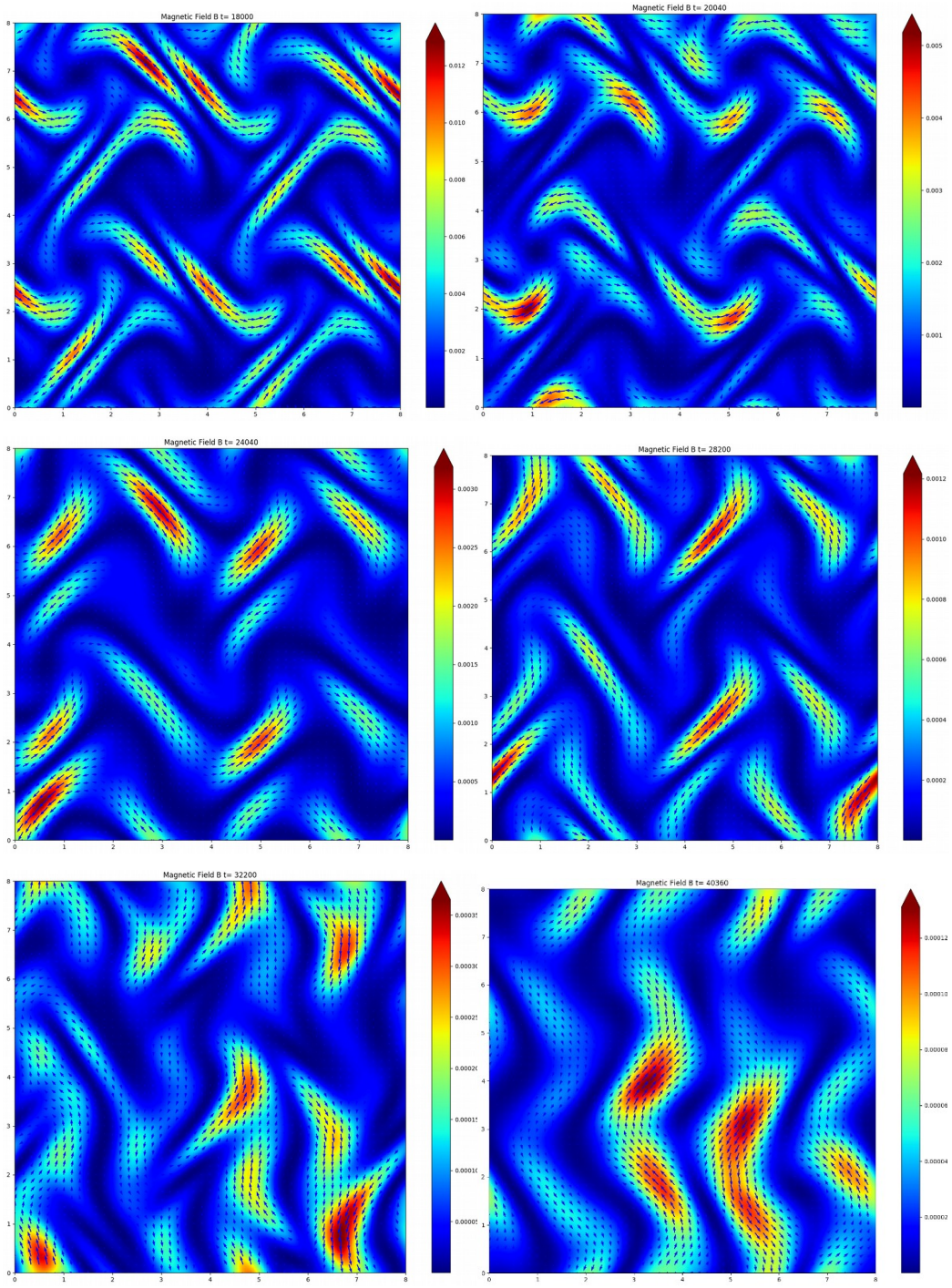

Magnetic Field B t= 5200
Magnetic Field B t= 6000


Magnetic Field B t= 7200
Magnetic Field B t= 8000


Magnetic Field B t= 9200
Magnetic Field B t= 10400


Magnetic Field B t= 11600
Magnetic Field B t= 12800


Magnetic Field B t= 14400
Magnetic Field B t= 16000



Gambar 4.1. Hasil simulasi kerapatan energi medan magnet.

Menggunakan teknik ini kita dapat menganalisa plasma tanpa perlu melakukan simulasi individual partikel, dengan itu memungkinkan computer masa kini digunakan untuk menganalisa perilaku plasma. Dalam simulasi ini kami

menjalankan program hingga 40000 time step. Dengan batasan yang digunakan kode stabil dan tidak menimbulkan underflow dalam skala waktu yang digunakan.

Meskipun cukup menarik, simulasi yang dilakukan hanya merepresentasikan kasus teoritik. Proses fisika dalam realita jauh lebih kompleks baik dari segi variable yang ada maupun domain ruang dan waktunya. Setelah meruning kode dengan kasus Orzag-Tang Vertex, selanjutnya kami berharap dapat mengaplikasikan kode ini dalam kasus yang ada di alam, misalkan simulasi atmosfer matahari, magnetosfer bumi, maupun simulasi naktivitas plasma dalam laboratorium.

Simulasi yang dibuat masih belum merepresentasikan gelombang alven maupun menggunakan besaran massa jenis yang realistis, karena dalam realita massa jenis akan berbeda beda di setiap daerah. Untuk membuat kode ini bekerja dalam system medium interplanet atau atmosfer matahari, perlu juga dimasukan variable lain seperti energy kinetic, karena system system tersebut tidak berada dalam equilibrium thermal.

Banyak fenomena yang diharapkan dapat dijelaskan dengan simulasi MHD, misalkan saja dinamika sunspot dan bintik matahari, aurora, badai magnetic bumi, dan coronal mass ejection. Dengan simulasi MHD yang semakin akurat maka ilmuwan dapat memprediksi aktivitas ruang interplanet secara lebih baik dan meneliti evolusi system dalam domain MHD, seperti evolusi galaksi dan gugus galaksi.

Selanjutnya pekerjaan ini dapat ditindaklanjuti dengan memperkenalkan perturbasi eksternal. Dalam realita perturbasi ini dapat mereprsesentasikan fenomena seperti kenaikan aktivitas matahari ataupun fenomena yang dinamakan coronal mass ejection. Dalam skala laboratorium fenomena seperti eksperimen fusi nuklir dapat memperkenalkan kondisi seperti peningktan densitas plasma ataupun pengenalan medan magnet eksternal. Hal ini dapat disimulasikan dengan memperkenalkan pertrubasi ke dalam simulasi setelah beberapa waktu

Hal lain yang perlu dipertimbangkan dalam running simulasi adalah kestabilan simulasi.simulasi numerik. Skema simulasi numerik apapun akan mengalami ketidakstabilan akibat akumulasi error. Makin lama hasil simulasi

akan semakin berbeda dengan hasil yang seharusnya. Dalam hal ini apabila nilai variabel penyebut dalam simulasi menjadi sangat kecil mendekati nol. Karena komputer memiliki limitasi akurasi dalam merepresentasikan angka, maka angka sangat kecil akan dibulatkan menjadi nol, hal ini akan membuat nilai variabel yang tidak masuk akal.

Seperti diketahui perhitungan analitik untuk masalah demikian mustahil untuk diselesaikan, karena melibatkan medan yang bentuknya sangat kompleks. Maka dari itu perbandingan hasil simulasi numerik dengan perhitungan analitik tidak dapat dilakukan. Yang dapat dilakukan adalah perbandingan hasil dengan program khusus yang digunakan profesional untuk melakukan simulasi. Dari perbandingan yang dilakukan dapat dilihat bahwa simulasi program yang dibuat dengan program professional hampir sama.

Kesulitan utama untuk memverivikasi hasil simulasi magnetohydrodynamic adalah sulitnya mengobservasi kondisi internal plasma. Ahal ini sangat terlihat untuk simulasi magnetohydrodynamic untuk objek antariksa. Untuk memperoleh data mengenai kondisi plasma objek antariksa, seperti magnetosfer bumi membutuhkan satelit observasi.walaupun demikian satelit hanya dap\at memperloleh data mengenai menai medan magnet, medan listrik dan mengenai plasma di satu titik saja dalam satu waktu. Walaupun satelit dapat bergerak untuk mengambil data di daerah lain, gerakan satelit dibatasi oleh orbit satelit, sehingga untuk memperoleh data yang bagus diperlukan banyak ssatelit dengan orbit yang berbeda beda. Kombinasi hal hal tersebut mengakibatkan data menganai aktivitas megnetosfer sulit untuk didapat, dan mempersulit konfirmasi hasil simulasi.

Untuk masa kini teknik yang digunakan untuk memantau kondisi aktivitas medan elektromagnet dan plasma di sekitar bumu adalah dengan menggunakan set magnetometer yang diletakan di lokasi lokasi strategis di permukaan bumi. Teknik ini tentunya tidak sempurna karena hanya dapat mengobservasi fenomena dengan mengukur dampaknya terhadap permukaan bumi, tidak secara langsung mengukurnya.

BAB V
PENUTUP

5.1 Kesimpulan

Dalam pekerjaan ini kami menampilkan simulasi magnetohydrodynamic menggunakan variable mesh. Pada bagian awal dijelaskan mengenai persamaan fluida dan persamaan Maxwell. Setelah itu diturunkan persamaan untuk magnetohydrodynamic.

Kemudian didiskusikan metode yang digunakan untuk mendiskritisasi persamaan magnetohydrodynamic dan skema numerik yang digunakan dalam simulasi, dijelaskan finite difference method, metode euler, metode relaksasi, dan metode variable mesh.

Kode ditulis menggunakan python dan modul nump. Beberapa simulasi dijalankan untuk mentest kemampuan kode dan efektivitas metode variable mesh. Pekerjaan tambahan diperlukan untuk mengembangkan kode ke dalam tiga dimensi dan memperluas scope masalah yang dapat diatasi. Namun pekerjaan ini diharapkan dapat digunakan menjadi pengnalan untuk orang yang ingin mempelajasi magneto fluida maupun dinamika fluida secara umum.

DAFTAR PUSTAKA

Farber, Ryan (2015). *Magenohydrodynamic Simulation in Python.* Wheaton Collage

Burgarelli et al.(2005). *A new adaptive mesh refinement strategy for numerically solving evolutionary PDE's.* Journal of Computational and Applied Mathematics 196 (2006) 115–131

Reader, Joachim.(2003). *Global Magnetohydrodynamics – A Tutorial* . Institute of Geophysics and Planetary Physics, University of California, Los Angeles, USA

Parks, George K.(2004). *Pyhsics of Space Plasmas An Introduction.* University of California, Berkeley:Westview Press

Lampiran 1

Kode Simulasi:

```
import numpy as np
import time
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import matplotlib.animation as animation
from matplotlib import style
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
#import pywt
import skimage.measure


def
Vcalc(u,v,rho,dt,dx,dy,UiP,UiN,VjP,VjN,UjP,UjN,ViP,ViN,p,Bx,By,rand1,rand2,U
ij,Vij):
    Dux = (UiP - (2 * Uij) + UiN) / (dx ** 2)
    Duy = (UjP - (2 * Uij) + UjN) / (dy ** 2)
    Dvx = (ViP - (2 * Vij) + ViN) / (dx ** 2)
    Dvy = (VjP - (2 * Vij) + VjN) / (dy ** 2)
    aux = (Uij - UiN) / dx
    auy = (Uij - UjN) / dy
    avx = (Vij - ViN) / dx
    avy = (Vij - VjN) / dy
    PjP = p[2:, 1:(n - 1)]
    PjN = p[:(n - 2), 1:(n - 1)]
    PiP = p[1:(n - 1), 2:]
    PiN = p[1:(n - 1), :(n - 2)]
    Pi=(PiP-PiN)*(dt/(rho*2*dx))
    Pj=(PjP-PjN)*(dt/(rho*2*dy))

    Babs = ((Bx**2)+(By**2))/2
    Bxij = Bx[1:(n - 1), 1:(n - 1)]
    Byij = By[1:(n - 1), 1:(n - 1)]
    BxjN = Bx[:(n - 2), 1:(n - 1)]
    BxiN = Bx[1:(n - 1), :(n - 2)]
    ByjN = By[:(n - 2), 1:(n - 1)]
    ByiN = By[1:(n - 1), :(n - 2)]
    Bxx = (Bxij - BxiN) / dx
    Bxy = (Bxij - BxjN) / dy
    Byx = (Byij - ByiN) / dx
    Byy = (Byij - ByjN) / dy
```

```
    Bij = Babs[1:(n - 1), 1:(n - 1)]
    BjN = Babs[:(n - 2), 1:(n - 1)]
    BiN = Babs[1:(n - 1), :(n - 2)]
    Bdx = (dt*(Bij - BiN)) / dx
    Bdy = (dt*(Bij - BjN)) / dy
    Bupart=(dt * Bxij * Bxx) + (dt * Byij * Bxy)-Bdx
    Bvpart=(dt * Bxij * Byx) + (dt * Byij * Byy)-Bdy

    u[1:(n - 1), 1:(n - 1)]= Uij - (dt * Uij * aux) - (dt * Vij * auy) + (eta * ((dt *
Dux) + (dt * Duy)))-Pi+Bupart
    v[1:(n - 1), 1:(n - 1)] = Vij - (dt * Uij * avx) - (dt * Vij * avy) + (eta * ((dt *
Dvx) + (dt * Dvy)))-Pj+Bvpart

    return u,v

# Density

def pbound(p):

    p=np.lib.pad(p,((1,1),(1,1)),'constant',constant_values=((1,1),(1,1)))

    return p

def bcalc(b,rho,dt,dx,dy,UiP,UiN,VjP,VjN,UjP,UjN,ViP,ViN):
    Dux = (UiP - UiN) / (2 * dx)
    Dvy = (VjP - VjN) / (2 * dy)
    Duy = (UjP - UjN) / (2 * dy)
    Dvx = (ViP - ViN) / (2 * dx)
    b[1:(n - 1), 1:(n - 1)]=rho*(((1/dt)*(Dux+Dvy))-(Dux**2)-(2*Duy*Dvx)-
(Dvy**2))
    return b
def pcalc(p,dx,dy,b,npress):
    divider=2*((dx**2)+(dy**2))
    for k in range(npress):                         #relaxing pressure

        PjP = p[2:, 1:(n - 1)]
        PjN = p[:(n - 2), 1:(n - 1)]
        PiP = p[1:(n - 1), 2:]
        PiN = p[1:(n - 1), :(n - 2)]
        p[1:(n - 1), 1:(n - 1)]=((((PiP+PiN)*(dy**2))+
((PjP+PjN)*(dx**2)))/divider)-((((dx**2)*(dy**2))/divider)*b[1:(n - 1),1:(n - 1)])

    return p
# Vector Potential

def Acalc(A,dt,dx,dy,Uij,Vij,eta2):
```

```
    Aij = A[1:(n - 1), 1:(n - 1)]
    AjP = A[2:, 1:(n - 1)]
    AjN = A[:(n - 2), 1:(n - 1)]
    AiP = A[1:(n - 1), 2:]
    AiN = A[1:(n - 1), :(n - 2)]
    aAx = (Aij - AiN) / dx
    aAy = (Aij - AjN) / dy
    DAx = (AiP - (2 * Aij) + AiN) / (dx ** 2)
    DAy = (AjP - (2 * Aij) + AjN) / (dy ** 2)

    A[1:(n - 1), 1:(n - 1)] = Aij - (dt * Uij * aAx) - (dt * Vij * aAy) + (eta2 * ((dt *
DAx) + (dt * DAy)))


    return A

#magnetic field
def Bcalc(A,dx,dy,Bx,By):
    Aij = A[1:(n - 1), 1:(n - 1)]
    AjN = A[:(n - 2), 1:(n - 1)]
    AiN = A[1:(n - 1), :(n - 2)]

    Bx[1:(n - 1), 1:(n - 1)]=(Aij-AjN)/dy
    By[1:(n - 1), 1:(n - 1)]= (Aij - AiN) / dx

    return Bx,By

#mathematical ops

def solve(u,v,A,p,Bx,By,b,dx,dy,ggg):
    Uij = u[1:(n - 1), 1:(n - 1)]
    Vij = v[1:(n - 1), 1:(n - 1)]
    UjP = u[2:, 1:(n - 1)]
    UjN = u[:(n - 2), 1:(n - 1)]
    UiP = u[1:(n - 1), 2:]
    UiN = u[1:(n - 1), :(n - 2)]
    VjP = v[2:, 1:(n - 1)]
    VjN = v[:(n - 2), 1:(n - 1)]
    ViP = v[1:(n - 1), 2:]
    ViN = v[1:(n - 1), :(n - 2)]
    A=Acalc(A, dt, dx, dy, Uij, Vij, eta2)
    Bx,By= Bcalc(A, dx, dy,Bx,By)
    npress = 50
    if ggg==0:
        npress = 1000
```

```
    b=bcalc(b,rho,dt,dx,dy,UiP,UiN,VjP,VjN,UjP,UjN,ViP,ViN)
    p=pcalc(p,dx,dy,b,npress)
    rand1=0
    rand2=0
    u,v=
Vcalc(u,v,rho,dt,dx,dy,UiP,UiN,VjP,VjN,UjP,UjN,ViP,ViN,p,Bx,By,rand1,rand2,U
ij,Vij)
    return u,v,A,p,Bx,By,b



def scaleup(value):
    new_value=value.repeat(2,axis=0).repeat(2,axis=1)
    return new_value
def scaleup1D(value):
    new_value=value.repeat(2,axis=0)
    return new_value
def scaledown1D(value):
    g1=value[0::2]
    g2=value[1::2]
    new=(g1+g2)/2
    return new

def max_pool(value):
    new_value=skimage.measure.block_reduce(value, (2,2), np.max)
    return new_value


def scaledown(value1,value2,value3,value4):
    value1=max_pool(value1)
    value2=max_pool(value2)
    value3=max_pool(value3)
    value4=max_pool(value4)
    new_value=np.zeros([10,10])
    new_value[indy1,indx1]=value1
    new_value[indy2,indx2]=value2
    new_value[indy3,indx3]=value3
    new_value[indy4,indx4]=value4
    return new_value

def refine(current,unit_location):
    new_value=current[:unit_location]
    unit=current[unit_location]
    unit=unit[1:-1,1:-1]
    a=scaleup(unit[indy1,indx1])
    b=scaleup(unit[indy2,indx2])
```

```
    c=scaleup(unit[indy3,indx3])
    d=scaleup(unit[indy4,indx4])
    a=np.lib.pad(a,((1,1),(1,1)),'constant',constant_values=((1,1),(1,1)))
    b=np.lib.pad(b,((1,1),(1,1)),'constant',constant_values=((1,1),(1,1)))
    c=np.lib.pad(c,((1,1),(1,1)),'constant',constant_values=((1,1),(1,1)))
    d=np.lib.pad(d,((1,1),(1,1)),'constant',constant_values=((1,1),(1,1)))
    new_value=np.append(new_value,[a],axis=0)
    new_value=np.append(new_value,[b],axis=0)
    new_value=np.append(new_value,[c],axis=0)
    new_value=np.append(new_value,[d],axis=0)
    new_value=np.append(new_value,current[(unit_location+1):],axis=0)
    return new_value

n=128
inda=np.arange(n/2)
indb=np.arange(n/2,n)
indx1,indy1=np.meshgrid(inda,inda)
indx2,indy2=np.meshgrid(indb,inda)
indx4,indy4=np.meshgrid(inda,indb)
indx3,indy3=np.meshgrid(indb,indb)
x = np.linspace(0.0, 8.0, n)
y = np.linspace(0.0, 8.0, n)
x,y=np.meshgrid(x,y)
t=np.array([[0,0,4,4],[4,0,8,4],[4,4,8,8,],[0,4,4,8]],dtype=np.float)

for i in range(4):
    s1=t[i,0]
    s2=t[i,1]
    d1=t[i,2]
    d2=t[i,3]
    xx = np.linspace(s1, d1, n)
    yy = np.linspace(s2, d2, n)
    xx,yy=np.meshgrid(xx,yy)
    if i==0:
        x=np.append([x],[xx],axis=0)
        y=np.append([y],[yy],axis=0)
    if i > 0:
        x=np.append(x,[xx],axis=0)
        y=np.append(y,[yy],axis=0)

def addsec(existing,new):
    new_existing=np.append(existing,[new],axis=0)
    return new_existing


def padding(vari1,vari2,vari3,vari4,vari5,vari6,vari7):
```

```
    vari1=np.lib.pad(vari1,((1,1),(1,1)),'constant',constant_values=((1,1),(1,1)))
    vari2=np.lib.pad(vari2,((1,1),(1,1)),'constant',constant_values=((1,1),(1,1)))
    vari3=np.lib.pad(vari3,((1,1),(1,1)),'constant',constant_values=((1,1),(1,1)))
    vari4=np.lib.pad(vari4,((1,1),(1,1)),'constant',constant_values=((1,1),(1,1)))
    vari5=np.lib.pad(vari5,((1,1),(1,1)),'constant',constant_values=((1,1),(1,1)))
    vari6=np.lib.pad(vari6,((1,1),(1,1)),'constant',constant_values=((1,1),(1,1)))
    vari7=np.lib.pad(vari7,((1,1),(1,1)),'constant',constant_values=((1,1),(1,1)))
    return vari1,vari2,vari3,vari4,vari5,vari6,vari7

def flowvar(var1,var2,var3,var4,var5):
    var1[0,:]=var5[-2,:];var1[-1,:]=var3[2,:]
    var1[:,0]=var2[:,-2];var1[:,-1]=var4[:,2]
    return var1



pi=np.pi
dx=8.0/(n-1)
dy=8.0/(n-1)
dx2=dx/2
dy2=dx2
dx3=dx/4
dx4=dx/8
dx5=dx/16
dt2=0.0002/2
dt3=0.0002/4
dt4=0.0002/8
dt5=0.0002/16


xm=x[0]
ym=y[0]
dt=0.0002
eta=0.1
eta2=0.1
rho=25/(36*pi)
u=-np.sin(ym*pi/2)
v=np.sin(xm*pi/2)
A=np.cos(ym*1*pi/2)+0.5*np.cos(xm*2*pi/2)
Bx=A*0
By=A*0
p=xm*0
b=ym*0

A1=scaleup(A[indy1,indx1])
Bx1=scaleup(Bx[indy1,indx1])
```

```
By1=scaleup(By[indy1,indx1])
p1=scaleup(p[indy1,indx1])
b1=scaleup(b[indy1,indx1])
u1=scaleup(u[indy1,indx1])
v1=scaleup(v[indy1,indx1])

A2=scaleup(A[indy2,indx2])
Bx2=scaleup(Bx[indy2,indx2])
By2=scaleup(By[indy2,indx2])
p2=scaleup(p[indy2,indx2])
b2=scaleup(b[indy2,indx2])
u2=scaleup(u[indy2,indx2])
v2=scaleup(v[indy2,indx2])

A3=scaleup(A[indy3,indx3])
Bx3=scaleup(Bx[indy3,indx3])
By3=scaleup(By[indy3,indx3])
p3=scaleup(p[indy3,indx3])
b3=scaleup(b[indy3,indx3])
u3=scaleup(u[indy3,indx3])
v3=scaleup(v[indy3,indx3])

A4=scaleup(A[indy4,indx4])
Bx4=scaleup(Bx[indy4,indx4])
By4=scaleup(By[indy4,indx4])
p4=scaleup(p[indy4,indx4])
b4=scaleup(b[indy4,indx4])
u4=scaleup(u[indy4,indx4])
v4=scaleup(v[indy4,indx4])

u1,v1,A1,p1,Bx1,By1,b1=padding(u1,v1,A1,p1,Bx1,By1,b1)
u2,v2,A2,p2,Bx2,By2,b2=padding(u2,v2,A2,p2,Bx2,By2,b2)
u3,v3,A3,p3,Bx3,By3,b3=padding(u3,v3,A3,p3,Bx3,By3,b3)
u4,v4,A4,p4,Bx4,By4,b4=padding(u4,v4,A4,p4,Bx4,By4,b4)

Amas=np.append([A1],[A2],axis=0)
Bxmas=np.append([Bx1],[Bx2],axis=0)
Bymas=np.append([By1],[By2],axis=0)
pmas=np.append([p1],[p2],axis=0)
bmas=np.append([b1],[b2],axis=0)
umas=np.append([u1],[u2],axis=0)
vmas=np.append([v1],[v2],axis=0)

Amas=addsec(Amas,A3)
Bxmas=addsec(Bxmas,Bx3)
Bymas=addsec(Bymas,By3)
```

```
pmas=addsec(pmas,p3)
bmas=addsec(bmas,b3)
umas=addsec(umas,u3)
vmas=addsec(vmas,v3)

Amas=addsec(Amas,A4)
Bxmas=addsec(Bxmas,Bx4)
Bymas=addsec(Bymas,By4)
pmas=addsec(pmas,p4)
bmas=addsec(bmas,b4)
umas=addsec(umas,u4)
vmas=addsec(vmas,v4)

#target=1
#Amas=refine(Amas,target)
#Bxmas=refine(Bxmas,target)
#Bymas=refine(Bymas,target)
#pmas=refine(pmas,target)
#bmas=refine(bmas,target)
#umas=refine(umas,target)
#vmas=refine(vmas,target)



def calctotal(Am,Bxm,Bym,pm,bm,um,vm,unit_count,hhh):
    for i in range(unit_count):
        a1A=Am[i]
        a1Bx=Bxm[i]
        a1By=Bym[i]
        a1p=pm[i]
        a1b=bm[i]
        a1u=um[i]
        a1v=vm[i]

a1u,a1v,a1A,a1p,a1Bx,a1By,a1b=solve(a1u,a1v,a1A,a1p,a1Bx,a1By,a1b,dx2,dy2,
hhh)
        Am[i]=a1A
        Bxm[i]=a1Bx
        Bym[i]=a1By
        pm[i]=a1p
        bm[i]=a1b
        um[i]=a1u
        vm[i]=a1v
    return Am,Bxm,Bym,pm,bm,um,vm
```

```
masL1=np.ones(4)
masL2=np.zeros([4,4])
masL3=np.zeros([4,4,4])
masL4=np.zeros([4,4,4,4])
masL5=np.zeros([4,4,4,4,4])
masindex=np.array([[1,0,0,0,0,0,0],[1,1,1,0,0,0,0],[1,2,2,0,0,0,0],
[1,3,3,0,0,0,0]],dtype=np.int16)
indexleft=np.array([[1,1,1,0,0,0,0],[1,1,0,0,0,0,0],[1,1,3,0,0,0,0],
[1,1,2,0,0,0,0]],dtype=np.int16)
indextop=np.array([[1,1,3,0,0,0,0],[1,1,2,0,0,0,0],[1,1,1,0,0,0,0],
[1,1,0,0,0,0,0]],dtype=np.int16)
indexright=np.array([[1,1,1,0,0,0,0],[1,1,0,0,0,0,0],[1,1,3,0,0,0,0],
[1,1,2,0,0,0,0]],dtype=np.int16)
indexbottom=np.array([[1,1,3,0,0,0,0],[1,1,2,0,0,0,0],[1,1,1,0,0,0,0],
[1,1,0,0,0,0,0]],dtype=np.int16)
masindex=np.append([masindex],[indexleft],axis=0)
masindex=np.append(masindex,[indextop],axis=0)
masindex=np.append(masindex,[indexright],axis=0)
masindex=np.append(masindex,[indexbottom],axis=0)
###indextop[targetlevel,deltalevel(0=lower,1=same,2=higher),target1,target2,side
(if delta 0)]

#print masindex

def transfervarindiv(var,index):
    indexcount=(index[0,-1,1])+1
    for aa in range(indexcount):
        var1=var[i]
        for bb in range(4):
            indextarget=index[bb+1,aa]
            deltaindex=indextarget[1]
            targetunit1=indextarget[2]
            targetunit2=indextarget[3]
            section=indextarget[4]
            var2=var[targetunit1]
            q=130

            if bb==0:
                if deltaindex==0:
                    if section==1:
                        cc1=var2[q/2:-1,-2]
                        cc1=scaleup1D(cc1)
                        var1[n/2:-1,-2]=cc1
                    if section==2:
                        cc1=var2[1:q/2,-2]
                        cc1=scaleup1D(cc1)
```

```
            var1[n/2:-1,-2]=cc1
        if deltaindex==1:
            cc2=var2[1:-1,-2]
            var1[1:-1,-2]=cc2
            print cc2.shape
        if deltaindex==2:
            var3=var[targetunit2]
            cc3=var2[1:-1,-2]
            cc4=var2[1:-1,-2]
            cc3=scaledown1D(cc3)
            cc4=scaledown1D(cc4)
            cc3=np.append(cc3,cc4,axis=0)
            var1[1:-1,-2]=cc3

    if bb==1:
        if deltaindex==0:
            if section==1:
                cc1=var2[q/2:-1,-2]
                cc1=scaleup1D(cc1)
                var1[n/2:-1,-2]=cc1
            if section==2:
                cc1=var2[1:q/2,-2]
                cc1=scaleup1D(cc1)
                var1[n/2:-1,-2]=cc1
        if deltaindex==1:
            cc2=var2[1:-1,-2]
            var1[1:-1,-2]=cc2
            print cc2.shape
        if deltaindex==2:
            var3=var[targetunit2]
            cc3=var2[1:-1,-2]
            cc4=var2[1:-1,-2]
            cc3=scaledown1D(cc3)
            cc4=scaledown1D(cc4)
            cc3=np.append(cc3,cc4,axis=0)
            var1[1:-1,-2]=cc3
    if bb==2:
        if deltaindex==0:
            if section==1:
                cc1=var2[q/2:-1,-2]
                cc1=scaleup1D(cc1)
                var1[n/2:-1,-2]=cc1
            if section==2:
                cc1=var2[1:q/2,-2]
                cc1=scaleup1D(cc1)
                var1[n/2:-1,-2]=cc1
```

```
        if deltaindex==1:
            cc2=var2[1:-1,-2]
            var1[1:-1,-2]=cc2
            print cc2.shape
        if deltaindex==2:
            var3=var[targetunit2]
            cc3=var2[1:-1,-2]
            cc4=var2[1:-1,-2]
            cc3=scaledown1D(cc3)
            cc4=scaledown1D(cc4)
            cc3=np.append(cc3,cc4,axis=0)
            var1[1:-1,-2]=cc3

    if bb==3:
        if deltaindex==0:
            if section==1:
                cc1=var2[q/2:-1,-2]
                cc1=scaleup1D(cc1)
                var1[n/2:-1,-2]=cc1
            if section==2:
                cc1=var2[1:q/2,-2]
                cc1=scaleup1D(cc1)
                var1[n/2:-1,-2]=cc1
        if deltaindex==1:
            cc2=var2[1:-1,-2]
            var1[1:-1,-2]=cc2
            print cc2.shape
        if deltaindex==2:
            var3=var[targetunit2]
            cc3=var2[1:-1,-2]
            cc4=var2[1:-1,-2]
            cc3=scaledown1D(cc3)
            cc4=scaledown1D(cc4)
            cc3=np.append(cc3,cc4,axis=0)
            var1[1:-1,-2]=cc3
```

transfervarindiv(Amas,masindex)

```
#def transfervar(Am,Bxm,Bym,pm,um,vm,index):
#    Am,Bxm,Bym,pm,bm,um,vm




nnn=0
n=130
      #after padding the size increase
for i in range(0):

    count=4

#    Bx1=flowvar(Bx1,Bx2,Bx1,Bx2,Bx1)
#    By1=flowvar(By1,By2,By1,By2,By1)
#    A1=flowvar(A1,A2,A1,A2,A1)
#    p1=flowvar(p1,p2,p1,p2,p2)
#    u1=flowvar(u1,u2,u1,u2,u1)

#    v1=flowvar(v1,v2,v1,v2,v1)
#
#    Bx2=flowvar(Bx2,Bx2,Bx1,Bx2,Bx1)
#    By2=flowvar(By2,By2,By1,By2,By1)
#    A2=flowvar(A2,A1,A1,A2,A1)
#    p2=flowvar(p2,p1,p1,p2,p2)
#    u2=flowvar(u2,u2,u1,u2,u1)
#    v2=flowvar(v2,v2,v1,v2,v1)
#
#    Bx3=flowvar(Bx3,Bx2,Bx1,Bx2,Bx1)
#    By3=flowvar(By3,By2,By1,By2,By1)
#    A3=flowvar(A3,A1,A1,A2,A1)
#    p3=flowvar(p3,p1,p1,p2,p2)
#    u3=flowvar(u3,u2,u1,u2,u1)
#    v3=flowvar(v3,v2,v1,v2,v1)
#
#    Bx4=flowvar(Bx4,Bx2,Bx1,Bx2,Bx1)
#    By4=flowvar(By4,By2,By1,By2,By1)
```

```
#    A4=flowvar(A4,A1,A1,A2,A1)
#    p4=flowvar(p4,p1,p1,p2,p2)
#    u4=flowvar(u4,u2,u1,u2,u1)
#    v4=flowvar(v4,v2,v1,v2,v1)


    iii=i

Amas,Bxmas,Bymas,pmas,bmas,umas,vmas=calctotal(Amas,Bxmas,Bymas,pmas
,bmas,umas,vmas,count,iii)
    print Amas.shape

    nnn=nnn+1

#    if nnn==40:
#        nnn=0
#        z=(Bx1**2)+(By1**2)
#        z2=(Bx2**2)+(By2**2)
#        z3=(Bx3**2)+(By3**2)
#        z4=(Bx4**2)+(By4**2)
#
#        plt.figure(figsize=(16, 12), dpi=100)
#        plt.plot()
#        plt.title("Magnetic Field B1 t= %s" % i)
#        II = plt.imshow(z, extent=[np.min(x[1]), np.max(x[1]), np.min(y[1]),
np.max(y[1])], cmap='jet',
#                    norm=colors.Normalize(vmin=z.min(), vmax=z.max()))
#        plt.savefig('Picture/8/%s B1.png' % i)
#        plt.clf()
#        plt.close()
#
#        plt.figure(figsize=(16, 12), dpi=100)
#        plt.plot()
#        plt.title("Magnetic Field B2 t= %s" % i)
#        II = plt.imshow(z2, extent=[np.min(x[2]), np.max(x[2]), np.min(y[2]),
np.max(y[2])], cmap='jet',
#                    norm=colors.Normalize(vmin=z2.min(), vmax=z2.max()))
#        plt.savefig('Picture/8/%s B2.png' % i)
#        plt.clf()
#        plt.close()
#
#        plt.figure(figsize=(16, 12), dpi=100)
#        plt.plot()
#        plt.title("Magnetic Field B3 t= %s" % i)
#        II = plt.imshow(z3, extent=[np.min(x[3]), np.max(x[3]), np.min(y[3]),
np.max(y[3])], cmap='jet',
```

```
#                 norm=colors.Normalize(vmin=z3.min(), vmax=z3.max()))
#       #plt.colorbar(II, extend='max')
#       plt.savefig('Picture/8/%s B3.png' % i)
#       plt.clf()
#       plt.close()
#
#
#       plt.figure(figsize=(16, 12), dpi=100)
#       plt.plot()
#       plt.title("Magnetic Field B4 t= %s" % i)
#       II = plt.imshow(z4, extent=[np.min(x[4]), np.max(x[4]), np.min(y[4]),
np.max(y[4])], cmap='jet',
#                 norm=colors.Normalize(vmin=z4.min(), vmax=z4.max()))
#       #plt.colorbar(II, extend='max')
#       plt.savefig('Picture/8/%s B4.png' % i)
#       plt.clf()
#       plt.close()
#
#       print i
```