

Alliance

1.0

Generated by Doxygen 1.9.1

1 File Index	1
1.1 File List	1
2 File Documentation	3
2.1 array.c File Reference	3
2.1.1 Detailed Description	4
2.1.2 Function Documentation	4
2.1.2.1 get_flat_c()	4
2.1.2.2 get_flat_r()	4
2.1.2.3 get_flatIndexComplex3D()	5
2.1.2.4 getIndChi()	5
2.1.2.5 getIndChiBufEL_c()	5
2.1.2.6 getIndChiBufEL_r()	7
2.1.2.7 getIndChiBufEM_c()	7
2.1.2.8 getIndChiBufEM_r()	8
2.1.2.9 multiply_ar_c()	8
2.1.2.10 multiply_ar_r()	8
2.2 diagnostics.c File Reference	8
2.2.1 Detailed Description	9
2.2.2 Function Documentation	9
2.2.2.1 diag_compute()	9
2.2.2.2 diag_computeFreeEnergy()	10
2.2.2.3 diag_computeFreeEnergyFields()	10
2.2.2.4 diag_computeKSpectrum()	10
2.2.2.5 diag_computeMSpectrum()	11
2.2.2.6 diag_computeSpectra()	11
2.2.2.7 diag_getShells()	11
2.2.2.8 diag_initSpec()	12
2.3 distrib.c File Reference	12
2.3.1 Detailed Description	12
2.3.2 Function Documentation	12
2.3.2.1 distrib_enforceReality()	12
2.3.2.2 distrib_getG()	13
2.3.2.3 distrib_getH()	13
2.3.2.4 distrib_getXGrad()	13
2.3.2.5 distrib_getYGrad()	14
2.3.2.6 distrib_getZGrad()	14
2.3.2.7 distrib_setZeroNHalf()	14
2.4 equation.c File Reference	15
2.4.1 Detailed Description	15
2.4.2 Function Documentation	16
2.4.2.1 equation_getLinearTerm()	16

2.4.2.2 equation_getNonlinearElectromagnetic()	16
2.4.2.3 equation_getNonlinearElectrostatic()	16
2.4.2.4 equation_getNonlinearProduct()	17
2.4.2.5 equation_getNonlinearTerm()	17
2.5 fields.c File Reference	18
2.5.1 Detailed Description	19
2.5.2 Function Documentation	19
2.5.2.1 fields_getA()	19
2.5.2.2 fields_getAFromH()	20
2.5.2.3 fields_getB()	20
2.5.2.4 fields_getBFromH()	20
2.5.2.5 fields_getChi()	21
2.5.2.6 fields_getChiA()	21
2.5.2.7 fields_getChiB()	21
2.5.2.8 fields_getChiPhi()	21
2.5.2.9 fields_getFields()	21
2.5.2.10 fields_getFieldsFromH()	22
2.5.2.11 fields_getGradX()	22
2.5.2.12 fields_getGradY()	22
2.5.2.13 fields_getPhi()	23
2.5.2.14 fields_getPhiFromH()	23
2.5.2.15 fields_init()	23
2.5.2.16 fields_sendG()	23
2.6 init.c File Reference	24
2.6.1 Detailed Description	25
2.6.2 Macro Definition Documentation	25
2.6.2.1 RANK_IO	25
2.6.3 Function Documentation	25
2.6.3.1 fill_rand()	25
2.6.3.2 fill_randM0()	25
2.6.3.3 fill_randSingleKM()	26
2.6.3.4 init_conditions()	26
2.6.3.5 init_energySpec()	26
2.6.3.6 init_initEnums()	27
2.6.3.7 init_printParameters()	27
2.6.3.8 init_start()	27
2.7 parameters_io.c File Reference	27
2.7.1 Detailed Description	28
2.7.2 Function Documentation	28
2.7.2.1 init_global_size()	28
2.7.2.2 read_parameters()	28
2.7.2.3 read_parametersFromFile()	28

2.8 solver.c File Reference	29
2.8.1 Detailed Description	29
2.8.2 Function Documentation	29
2.8.2.1 solver_init()	29
2.8.2.2 solver_makeStep()	29
2.9 space_config.c File Reference	30
2.9.1 Detailed Description	31
2.9.2 Function Documentation	31
2.9.2.1 free_wavespace()	31
2.9.2.2 space_generateMSpace()	31
2.9.2.3 space_generateWaveSpace()	31
2.10 utils_fftw.c File Reference	31
2.10.1 Detailed Description	33
2.10.2 Function Documentation	33
2.10.2.1 cosinus()	33
2.10.2.2 dealiasing23()	33
2.10.2.3 fftw_c2r()	34
2.10.2.4 fftw_c2r_chi()	34
2.10.2.5 fftw_c2r_field()	34
2.10.2.6 fftw_copy_buffer_c()	34
2.10.2.7 fftw_copy_buffer_r()	35
2.10.2.8 fftw_copyChiBuf_c()	35
2.10.2.9 fftw_copyChiBuf_r()	35
2.10.2.10 fftw_copyFieldBuf_c()	35
2.10.2.11 fftw_copyFieldBuf_r()	36
2.10.2.12 fftw_kill()	36
2.10.2.13 fftw_normalise_chi_r()	36
2.10.2.14 fftw_normalise_data()	37
2.10.2.15 fftw_normalise_data_r()	37
2.10.2.16 fftw_normalise_field_r()	37
2.10.2.17 fftw_r2c()	37
2.10.2.18 fftw_r2c_chi()	37
2.10.2.19 fftw_r2c_field()	38
2.10.2.20 fftw_test_fill()	38
2.11 utils_hdf.c File Reference	38
2.11.1 Detailed Description	40
2.11.2 Function Documentation	40
2.11.2.1 hdf_create_file_c()	40
2.11.2.2 hdf_create_file_r()	40
2.11.2.3 hdf_createCheckpoint()	40
2.11.2.4 hdf_createChiFile_c()	40
2.11.2.5 hdf_createChiFile_r()	40

2.11.2.6 hdf_createFieldFile()	41
2.11.2.7 hdf_createFiles()	41
2.11.2.8 hdf_createParamFile()	41
2.11.2.9 hdf_createSaveDirs()	41
2.11.2.10 hdf_dumpCheckpoint()	41
2.11.2.11 hdf_init()	41
2.11.2.12 hdf_initCheckpoints()	41
2.11.2.13 hdf_initChi()	42
2.11.2.14 hdf_initField()	42
2.11.2.15 hdf_readData()	42
2.11.2.16 hdf_saveData()	42
2.11.2.17 hdf_saveEnergy()	42
2.11.2.18 hdf_saveField_r()	42
2.11.2.19 hdf_saveFieldA()	42
2.11.2.20 hdf_saveFieldB()	43
2.11.2.21 hdf_saveFieldPhi()	43
2.11.2.22 hdf_saveFields()	43
2.11.2.23 hdf_saveKSpec()	43
2.11.2.24 hdf_saveMSpec()	43
2.12 utils_mpi.c File Reference	43
2.12.1 Detailed Description	44
2.12.2 Function Documentation	45
2.12.2.1 mpi_createTopology()	45
2.12.2.2 mpi_exchangeMBoundaries()	45
2.12.2.3 mpi_exchangeMBoundaries_r()	45
2.12.2.4 mpi_findHermiteNeighbours()	45
2.12.2.5 mpi_generateTopology()	45
2.12.2.6 mpi_getLocalArrayOffsets()	45
2.12.2.7 mpi_getLocalArraySize()	46
2.12.2.8 mpi_init()	46
2.12.2.9 mpi_initMExchange()	46
2.12.2.10 mpi_kill()	46
2.12.2.11 mpi_sendVector()	46
2.12.2.12 mpi_splitInCols()	46
2.12.2.13 mpi_splitInRows()	46
2.13 variables.c File Reference	47
2.13.1 Detailed Description	47
2.13.2 Function Documentation	47
2.13.2.1 var_getJ0()	47
2.13.2.2 var_getJ1()	47

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

array.c	Array manipulation module	3
diagnostics.c	Diagnostics module	8
distrib.c	Gyrokinetic distribution function module	12
equation.c	Equation module	15
fields.c	Field computation and manipulation module	18
init.c	Initialization module for alliance	24
parameters_io.c	Reads input parameters from parameter file provided by user	27
solver.c	Numerical solver	29
space_config.c	Space configuration module	30
utils_fftw.c	FFT module	31
utils_hdf.c	Hdf module	38
utils_mpi.c	Mpi module	43
variables.c	Stores physical variables	47

Chapter 2

File Documentation

2.1 array.c File Reference

array manipulation module

```
#include "array.h"
#include "utils_fftw.h"
#include "space_config.h"
```

Macros

- `#define CHI_EM 3`
- `#define CHI_EL 1`
- `#define FFT_OFFSET 2`

Functions

- `size_t get_flat_c` (size_t is, size_t il, size_t im, size_t ix, size_t iy, size_t iz)
returns flat index of the element of complex 6D array
- `size_t getIndChiBufEM_c` (size_t ix, size_t iy, size_t iz, size_t is, size_t ifield)
returns flat index of an element of electromagnetic gyrokinetic potential in FOURIER SPACE
- `size_t getIndChiBufEM_r` (size_t ix, size_t iy, size_t iz, size_t is, size_t ifield)
returns flat index of an element of electromagnetic gyrokinetic potential in POSITION SPACE
- `size_t getIndChiBufEL_c` (size_t ix, size_t iy, size_t iz, size_t is)
returns returns flat index of an element of electrostatic gyrokinetic potential in FOURIER SPACE
- `size_t getIndChiBufEL_r` (size_t ix, size_t iy, size_t iz, size_t is)
returns returns flat index of an element of electrostatic gyrokinetic potential in REAL SPACE
- `size_t get_flat_r` (size_t is, size_t il, size_t im, size_t ix, size_t iy, size_t iz)
returns flat index of the element of real 6D array
- `size_t get_flatIndexComplex3D` (size_t ix, size_t iy, size_t iz)
returns flat array of complex 3D array
- `size_t getIndChi` (size_t ix, size_t iy, size_t iz, size_t is)
- `void multiply_ar_c` (COMPLEX *ar1, COMPLEX *ar2, COMPLEX *ret)
- `void multiply_ar_r` (const double *ar1, const double *ar2, double *ret)

Variables

- struct array_size **array_local_size**
- struct array_size **array_global_size**
- struct offset_size **array_offset**
- struct offset_size **array_offset3D**

2.1.1 Detailed Description

array manipulation module

contains functions which are supposed to make array manipulation simpler

2.1.2 Function Documentation

2.1.2.1 `get_flat_c()`

```
size_t get_flat_c (
    size_t is,
    size_t il,
    size_t im,
    size_t ix,
    size_t iy,
    size_t iz )
```

returns flat index of the element of complex 6D array

Parameters

<i>is</i>	species type
<i>il</i>	Laguerre moment
<i>im</i>	Hermite moment
<i>ix</i>	kx index
<i>iy</i>	ky index
<i>iz</i>	kz index

returns flattened index of a complex array from its 6D index. Flattened index then can be passed to distribution function 6D array to get a required element at position (is,il,im,ix,iy,iz).

2.1.2.2 `get_flat_r()`

```
size_t get_flat_r (
    size_t is,
    size_t il,
    size_t im,
    size_t ix,
```

```

size_t iy,
size_t iz )

```

returns flat index of the element of real 6D array

Parameters

<i>is</i>	species type
<i>il</i>	Laguerre moment
<i>im</i>	Hermite moment
<i>ix</i>	x index
<i>iy</i>	y index
<i>iz</i>	z index

returns flattened index of a real array from its 6D index. Flattened index then can be passed to distribution function 6D array to get a required element at position (is,il,im,ix,iy,iz).

2.1.2.3 get_flatIndexComplex3D()

```

size_t get_flatIndexComplex3D (
    size_t ix,
    size_t iy,
    size_t iz )

```

returns flat array of complex 3D array

Parameters

<i>ix</i>	kx index
<i>iy</i>	ky index
<i>iz</i>	kz index

returns flattened index of a complex array from its 3D position index. Flattened index then can be passed to one of the fields ($\phi(\mathbf{k})$, $A_{||}(\mathbf{k})$, $B_{||}(\mathbf{k})$) 6D array to get a required element at position (ix,iy,iz).

2.1.2.4 getIndChi()

```

size_t getIndChi (
    size_t ix,
    size_t iy,
    size_t iz,
    size_t is )

```

[getIndChi\(size_t ix,size_t iy, size_t iz, size_t is\)](#)

2.1.2.5 getIndChiBufEL_c()

```

size_t getIndChiBufEL_c (
    size_t ix,

```

```
size_t iy,  
size_t iz,  
size_t is )
```

returns returns flat index of an element of electrostatic gyrokinetic potential in FOURIER SPACE

Parameters

<i>ix</i>	kx index
<i>iy</i>	ky index
<i>iz</i>	kz index
<i>is</i>	particle species index

returns flattened index of a gyrokinetic potential $\chi^\phi(\mathbf{k})$ from its 4D index in FOURIER SPACE. flattened index is then can be used to access required value of the gyrokinetic potential at position (ix,iy,iz,is).

2.1.2.6 getIndChiBufEL_r()

```
size_t getIndChiBufEL_r (
    size_t ix,
    size_t iy,
    size_t iz,
    size_t is )
```

returns returns flat index of an element of electrostatic gyrokinetic potential in REAL SPACE

Parameters

<i>ix</i>	x index
<i>iy</i>	y index
<i>iz</i>	z index
<i>is</i>	particle species index

returns flattened index of a gyrokinetic potential $\chi^\phi(\mathbf{r})$ from its 4D index in REAL SPACE. flattened index is then can be used to access required value of the gyrokinetic potential at position (ix,iy,iz,is).

2.1.2.7 getIndChiBufEM_c()

```
size_t getIndChiBufEM_c (
    size_t ix,
    size_t iy,
    size_t iz,
    size_t is,
    size_t ifield )
```

returns flat index of an element of electromagnetic gyrokinetic potential in FOURIER SPACE

Parameters

<i>ix</i>	kx index
<i>iy</i>	ky index
<i>iz</i>	kz index
<i>is</i>	particle species index
<i>ifield</i>	field type

returns flattened index of a gyrokinetic potential $\chi^{\phi,A,B}$ from its 4D index in FOURIER SPACE. flattened index is then can be used to access required value of the gyrokinetic potential at position (ix,iy,iz,is). Type of gyrokinetic potential is specified by ifield parameter. Use 0 is to access $\chi^{\phi}(\mathbf{k})$, 1 to access $\chi^A(\mathbf{k})$ and 2 to access $\chi^B(\mathbf{k})$.

2.1.2.8 getIndChiBufEM_r()

```
size_t getIndChiBufEM_r (
    size_t ix,
    size_t iy,
    size_t iz,
    size_t is,
    size_t ifield )
```

returns flat index of an element of electromagnetic gyrokinetic potential in POSITION SPACE

Parameters

<i>ix</i>	x index
<i>iy</i>	y index
<i>iz</i>	z index
<i>is</i>	particle species index
<i>ifield</i>	field type

returns flattened index of a gyrokinetic potential $\chi^{\phi,A,B}(\mathbf{k})$ from its 4D index in POSITION SPACE. flattened index is then can be used to access required value of the gyrokinetic potential at position (ix,iy,iz,is). Type of gyrokinetic potential is specified by ifield parameter. Use 0 is to access $\chi^{\phi}(\mathbf{r})$, 1 to access $\chi^A(\mathbf{r})$ and 2 to access $\chi^B(\mathbf{r})$.

2.1.2.9 multiply_ar_c()

```
void multiply_ar_c (
    COMPLEX * ar1,
    COMPLEX * ar2,
    COMPLEX * ret )
```

[multiply_ar_c\(COMPLEX *ar1, COMPLEX *ar2, COMPLEX *ret\)](#)

2.1.2.10 multiply_ar_r()

```
void multiply_ar_r (
    const double * ar1,
    const double * ar2,
    double * ret )
```

[multiply_ar_r\(const double *ar1, const double *ar2, double *ret\)](#)

2.2 diagnostics.c File Reference

diagnostics module

```
#include "diagnostics.h"
#include "parameters_io.h"
```

Macros

- #define **TO_ROOT** 0
- #define **BUFFER_SIZE** 1

Functions

- void [diag_computeSpectra](#) (const COMPLEX *g, const COMPLEX *h, int timestep)
general function to compute k or m spectra
- void [diag_initSpec](#) ()
initialize spectra computation
- void [diag_computeFreeEnergy](#) (COMPLEX *g, COMPLEX *h)
compute free energy
- void [diag_computeKSpectrum](#) (const COMPLEX *g, const COMPLEX *h, double *spec)
- void [diag_computeMSpectrum](#) (const COMPLEX *g, const COMPLEX *h, double *spec)
computes free energy spectra in m space
- void [diag_getShells](#) ()
computes shells from parameters
- double [diag_computeFreeEnergyFields](#) (COMPLEX *g, COMPLEX *fields)
to be done later
- void [diag_compute](#) (COMPLEX *g, COMPLEX *h, int timestep)
computes all diagnostics

Variables

- double * [diag_kSpec](#) = 0
used to store free energy k spectra
- double * [diag_mSpec](#) = 0
used to store free energy m spectra
- double * [diag_shells](#) = 0
used to store positions of k shells required to compute k spectra
- double [diag_freeEnergy](#)
free energy

2.2.1 Detailed Description

diagnostics module

different diagnostic tools are gathered in this module

2.2.2 Function Documentation

2.2.2.1 [diag_compute\(\)](#)

```
void diag_compute (
    COMPLEX * g,
    COMPLEX * h,
    int timestep )
```

computes all diagnostics

Parameters

<i>g</i>	modified distribution function
<i>h</i>	distribution function
<i>iter</i>	current time step

2.2.2.2 diag_computeFreeEnergy()

```
void diag_computeFreeEnergy (
    COMPLEX * g,
    COMPLEX * h )
```

compute free energy

Parameters

<i>g</i>	modified gyrokinetic distribution function
<i>h</i>	gyrokinetic distribution function

computes free energy as $W = 2.\Re(\sum_{k_x, k_y, k_z > 0, m, l, s} g * \bar{h})$, taking into account reality condition.

2.2.2.3 diag_computeFreeEnergyFields()

```
diag_computeFreeEnergyFields (
    COMPLEX * g,
    COMPLEX * fields )
```

to be done later

Parameters

<i>g</i>	
<i>fields</i>	computes free energy from the fields and distribution function.

2.2.2.4 diag_computeKSpectrum()

```
void diag_computeKSpectrum (
    const COMPLEX * g,
    const COMPLEX * h,
    double * spec )
```

Parameters

<i>g</i>	modified gyrokinetic distribution function
<i>h</i>	gyrokinetic distribution function
<i>spec</i>	spectra array

computes free energy k_{\perp} spectra $W(k_i^{shell}) = \frac{1}{N} \sum_{k_{i-1}^{shell} < |k_{\perp}| < k_i^{shell}} \sum_{k_z, l, m, s} g \bar{h}$ where N is a number of wave vectors between shells k_{i-1}^{shell} and k_i^{shell}

2.2.2.5 diag_computeMSpectrum()

```
void diag_computeMSpectrum (
    const COMPLEX * g,
    const COMPLEX * h,
    double * spec )
```

computes free energy spectra in m space

Parameters

<i>g</i>	modified gyrokinetic distribution function
<i>h</i>	gyrokinetic distribution function
<i>spec</i>	spectra array

computes free energy m spectra as $W(m) = \sum_{k_x, k_y, k_z, l, s} g \bar{h}$

2.2.2.6 diag_computeSpectra()

```
void diag_computeSpectra (
    const COMPLEX * g,
    const COMPLEX * h,
    int timestep )
```

general function to compute k or m spectra

Parameters

<i>g</i>	gyrokinetic distribution function
<i>h</i>	distribution function
<i>timestep</i>	current time step

function computes spectra at timestep as given in parameter file. k_{\perp} spectra is computed using [diag_computeKSpectrum](#), and m spectra is computed using [diag_computeMSpectrum](#)

2.2.2.7 diag_getShells()

```
diag_getShells ( )
```

computes shells from parameters

computes positions of k_shells in between last_shell and first_shell as provided by user in parameter file. Position of i^{th} shell is computed as $k_i^{shell} = (last_shell - first_shell)/(k_shells) \cdot i$

2.2.2.8 diag_initSpec()

```
void diag_initSpec ( )
```

initialize spectra computation

Prepares free energy spectra computation. For spectra in k: allocates diag_kSpec array used to store k spectra. Allocates diag_shells array and fills it with shell positions k^{shells} , used for binning of wave vectors when computing k_{\perp} spectra. For spectra in m: allocates diag_mSpec array used to store m spectra. Called in [init_start](#) function

2.3 distrib.c File Reference

gyrokinetic distribution function module

```
#include "distrib.h"
```

Functions

- void [distrib_getH](#) (COMPLEX *h, const COMPLEX *g)
computes h from g
- void [distrib_getG](#) (COMPLEX *g, const COMPLEX *h)
computes g from h
- void [distrib_getXGrad](#) (const COMPLEX *in, COMPLEX *out)
Computes gradient in kx direction.
- void [distrib_getYGrad](#) (const COMPLEX *in, COMPLEX *out)
Computes gradient in ky direction.
- void [distrib_getZGrad](#) (const COMPLEX *in, COMPLEX *out)
Computes gradient in kz direction.
- void [distrib_enforceReality](#) (COMPLEX *f)
enforces reality condition on distribution function array
- void [distrib_setZeroNHalf](#) (COMPLEX *f)
sets all Nk/2 modes to zero

2.3.1 Detailed Description

gyrokinetic distribution function module

everything required to perform different manipulations to distribution functions

2.3.2 Function Documentation

2.3.2.1 distrib_enforceReality()

```
void distrib_enforceReality (
    COMPLEX * f )
```

enforces reality condition on distribution function array

Parameters

<i>f</i>	complex array for which reality condition will be forced.
----------	---

Enforces reality condition $f(k) = \text{conj}(f(-k))$ in plane $k_z = 0$. For a given k_x , it first checks where modes $-k_x$ are located using the `#mpi_whereIsX` function:

```
where_neg = mpi_whereIsX[kxNeg * 2];
```

If $-k_x$ is stored on a different processor, `MPI_VECTOR` with a 4D data slice $f(k_x, k_z = 0)$ is sent to this processor, to the buffer array:

```
mpi_sendVector(&f[ind6D], buffer, where_pos, where_neg);
```

if the data stored on the same processor, no vector is being sent. Reality condition is fulfilled in a loop over all other coordinates:

```
f[ind6D_neg] = conj(buffer[ind6D_pos]);
```

2.3.2.2 distrib_getG()

```
distrib_getG (
    COMPLEX * g,
    const COMPLEX * h )
```

computes g from h

Parameters

<i>g</i>	complex array to store g
<i>h</i>	complex array with h

computes modified gyrokinetic distribution function g from gyrokinetic distribution function h . Please note that before calling this function gyrokinetic potentials must be computed

2.3.2.3 distrib_getH()

```
void distrib_getH (
    COMPLEX * h,
    const COMPLEX * g )
```

computes h from g

Parameters

<i>h</i>	complex array to store h
<i>g</i>	complex array with g

computes gyrokinetic distribution function h from modified gyrokinetic distribution function g . Please note that before calling this function gyrokinetic potentials must be computed

2.3.2.4 distrib_getXGrad()

```
void distrib_getXGrad (
```

```
const COMPLEX * in,
COMPLEX * out )
```

Computes gradient in kx direction.

Parameters

<i>in</i>	complex array. Distribution function of which gradient will be taken
<i>out</i>	complex array, where gradient is stored

Computes gradient in kx direction as following:

$$\text{grad}(f) = i * kx * f$$

2.3.2.5 distrib_getYGrad()

```
void distrib_getYGrad (
    const COMPLEX * in,
    COMPLEX * out )
```

Computes gradient in ky direction.

Parameters

<i>in</i>	complex array. Distribution function of which gradient will be taken
<i>out</i>	complex array, where gradient is stored

Computes gradient in ky direction as following:

$$\text{grad}(f) = i * ky * f$$

2.3.2.6 distrib_getZGrad()

```
void distrib_getZGrad (
    const COMPLEX * in,
    COMPLEX * out )
```

Computes gradient in kz direction.

Parameters

<i>in</i>	complex array. Distribution function of which gradient will be taken
<i>out</i>	complex array, where gradient is stored

Computes gradient in kz direction as following:

$$\text{grad}(f) = i * kz * f$$

2.3.2.7 distrib_setZeroNHalf()

```
void distrib_setZeroNHalf (
    COMPLEX * f )
```

sets all $N_k/2$ modes to zero

Parameters

<i>f</i>	complex array
----------	---------------

sets $N_{kx}/2$, $N_{ky}/2$ and $N_z/2$ modes of distribution function to zero. Due to reality condition, for k_z the last mode should be set to zero.

2.4 equation.c File Reference

equation module

```
#include "equation.h"
#include "array.h"
#include "fields.h"
#include "distrib.h"
```

Macros

- `#define CHI_EM 3`
- `#define CHI_EL 1`
- `#define CHI_PHI 0`
- `#define CHI_A 1`
- `#define CHI_B 2`

Functions

- void [equation_getLinearTerm](#) (const COMPLEX *in, const COMPLEX *plus_boundary, const COMPLEX *minus_boundary, COMPLEX *out)
computes linear term
- void [equation_getNonlinearElectromagnetic](#) (double *in, double *chiAr, double *out, double sign)
returns nonlinear electromagnetic term
- void [equation_getNonlinearElectrostatic](#) (double *in, double *chiAr, double *out, double sign)
returns nonlinear electrostatic term
- void [equation_getNonlinearProduct](#) (double *in, double *chiAr, double *out, double sign)
chooses between computing electrostatic or electromagnetic term
- void [equation_getNonlinearTerm](#) (const COMPLEX *h, COMPLEX *out)
computes nonlinear term
- void [equation_getRHS](#) (const COMPLEX *in_g, COMPLEX *in_h, COMPLEX *out)

2.4.1 Detailed Description

equation module

module required to compute RHS of the equation.

2.4.2 Function Documentation

2.4.2.1 `equation_getLinearTerm()`

```
void equation_getLinearTerm (
    const COMPLEX * in,
    const COMPLEX * plus_boundary,
    const COMPLEX * minus_boundary,
    COMPLEX * out )
```

computes linear term

Parameters

<i>in</i>	complex array
<i>out</i>	complex array
<i>plus_boundary</i>	complex array
<i>minus_boundary</i>	complex array

computes linear term *out* from distribution function *in* .

2.4.2.2 `equation_getNonlinearElectromagnetic()`

```
void equation_getNonlinearElectromagnetic (
    double * in,
    double * chiAr,
    double * out,
    double sign )
```

returns nonlinear electromagnetic term

Parameters

<i>in</i>	input double array
<i>chiAr</i>	input double array
<i>out</i>	output double array
<i>sign</i>	should be 1 or -1

performs multiplication between input 6D complex array *in* and gyrokinetic potential array *chiAr*, in such way that the structure of the product is the same as nonlinear term of drift-kinetic equations. Used by [equation_getNonlinearProduct](#). *sign* is used to determine the sign of the resulting product. See [equation_getNonlinearTerm](#) for explanation.

2.4.2.3 `equation_getNonlinearElectrostatic()`

```
void equation_getNonlinearElectrostatic (
    double * in,
```

```
double * chiAr,
double * out,
double sign )
```

returns nonlinear electrostatic term

Parameters

<i>in</i>	input double array
<i>chiAr</i>	input double array
<i>out</i>	output double array
<i>sign</i>	should be 1 or -1

see [equation_getNonlinearElectromagnetic](#) for explanation

2.4.2.4 equation_getNonlinearProduct()

```
equation_getNonlinearProduct (
    double * in,
    double * chiAr,
    double * out,
    double sign )
```

chooses between computing electrostatic or electromagnetic term

Parameters

<i>in</i>	input double array
<i>chiAr</i>	input double array
<i>out</i>	output double array
<i>sign</i>	should be 1 or -1

depending on flag `systemType` provided by user in parameter file, chooses between [equation_getNonlinearElectrostatic](#) and [equation_getNonlinearElectromagnetic](#)

2.4.2.5 equation_getNonlinearTerm()

```
void equation_getNonlinearTerm (
    const COMPLEX * h,
    COMPLEX * out )
```

computes nonlinear term

Parameters

<i>h</i>	input complex array
<i>out</i>	output complex array

function returns nonlinear term. First it takes y gradient of distribution function *h*, and x gradient of gyrokinetic

potentials chi, and transforms them to real space:

```
distrib_getYGrad(h, fftw_hBuf);
fields_getGradX(fftw_chiBuf);
fftw_c2r();
fftw_c2r_chi();
```

after that, it computes $\frac{\partial h}{\partial y} \frac{\partial \chi}{\partial x}$ part of the poisson brackets:

```
equation_getNonlinearProduct((double *)fftw_hBuf, (double *)fftw_chiBuf,
buffer, 1.);
```

with the result stored in `buffer` after that, it computes x gradient of h and y gradient of gyrokinetic potential chi, and transforms results to real space:

```
distrib_getXGrad(h, fftw_hBuf);
fields_getGradY(fftw_chiBuf);
fftw_c2r();
fftw_c2r_chi();
```

and computes second part of the poisson brackets $-\frac{\partial h}{\partial x} \frac{\partial \chi}{\partial y}$ and adds the result to `buffer`. `buffer` is then transformed back to Fourier space, and dealiasing is performed.

2.5 fields.c File Reference

field computation and manipulation module

```
#include "fields.h"
```

Macros

- `#define CHI_PHI 0`
- `#define CHI_A 1`
- `#define CHI_B 2`

Functions

- void `fields_init()`
intializes fields
- void `fields_getA`(const COMPLEX *g)
compute A field
- void `fields_getB`(const COMPLEX *g0, const COMPLEX *g1)
computes B potential
- void `fields_getPhi`(const COMPLEX *g0, const COMPLEX *g1)
computes phi potential
- void `fields_getFields`(COMPLEX *g00, COMPLEX *g10, COMPLEX *g01)
wrapper to get all the fields simultaneously
- void `fields_getChi()`
computes gyrokinetic potentials chi
- void `fields_getChiPhi()`
computes chiPhi gyrokinetic potential from phi potential
- void `fields_getChiB()`
computes chiB gyrokinetic potential from B potential
- void `fields_getChiA()`
computes chiA gyrokinetic potential from A potential

- void `fields_sendG` (COMPLEX *g)
sends moments of gyrokinetic distribution function which are required to compute fields
- void `fields_getFieldsFromH` (COMPLEX *h00, COMPLEX *h10, COMPLEX *h01)
wrapper to get all the fields simultaneously, computed from gyrokinetic distribution function
- void `fields_getAFromH` (const COMPLEX *h)
compute A field
- void `fields_getBFromH` (const COMPLEX *h0, const COMPLEX *h1)
computes B potential
- void `fields_getPhiFromH` (const COMPLEX *h)
computes phi potential
- void `fields_getGradX` (COMPLEX *out)
computes chi gradient in x direction
- void `fields_getGradY` (COMPLEX *out)
computes chi gradient in y direction

Variables

- struct fields_fields **fields_fields**
- struct fields_chi **fields_chi**
- double * **A_denom**
- double * **qnvTsJ**
- double * **I_B**
- double * **I_phi**
- double * **a_pot**
- double * **b_pot**
- double * **c_pot**
- double * **phiB_denom**
- int * **global_nm_index**
- COMPLEX * **g00**
- COMPLEX * **g10**
- COMPLEX * **g01**

2.5.1 Detailed Description

field computation and manipulation module

Required to compute $A_{||}(\mathbf{k})$, $B_{||}(\mathbf{k})$, $\phi(\mathbf{k})$ potentials, as well as gyrokinetic potentials $\chi_s^A(\mathbf{k})$, $\chi_s^B(\mathbf{k})$, $\chi_s^\phi(\mathbf{k})$

2.5.2 Function Documentation

2.5.2.1 fields_getA()

```
void fields_getA (
    const COMPLEX * g )
```

compute A field

Parameters

<i>g</i>	4D complex array (kx,ky,kz,s). Must be first Hermite and zeroth Laguerre moment of modified gyrokinetic distribution function g.
----------	--

computes $A_{||}(\mathbf{k})$ potential from g_{s0}^1 (g parameter)

2.5.2.2 fields_getAFromH()

```
void fields_getAFromH (
    const COMPLEX * h )
```

compute A field

Parameters

<i>h</i>	4D complex array (kx,ky,kz,s). Must be first Hermite and zeroth Laguerre moment of gyrokinetic distribution function h.
----------	---

computes $A_{||}(\mathbf{k})$ potential from h_{s0}^1 (h parameter)

2.5.2.3 fields_getB()

```
void fields_getB (
    const COMPLEX * g0,
    const COMPLEX * g1 )
```

computes B potential

Parameters

<i>g0</i>	4D complex array (kx,ky,kz,s). Zeroth Hermite and Laguerre moment of modified gyrokinetic distribution function.
<i>g1</i>	4D complex array (kx,ky,kz,s). Zeroth Hermite and first Laguerre moment of the modified distribution function.

Computes $B_{\perp}(\mathbf{k})$ from g_{s0}^0 (g0 parameter) and g_{s0}^1 (g1 parameter).

2.5.2.4 fields_getBFromH()

```
void fields_getBFromH (
    const COMPLEX * h0,
    const COMPLEX * h1 )
```

computes B potential

Parameters

<i>h0</i>	4D complex array (kx,ky,kz,s). Zeroth Hermite and Laguerre moment of gyrokinetic distribution function.
<i>h1</i>	4D complex array (kx,ky,kz,s). Zeroth Hermite and first Laguerre moment of distribution function.

Computes $B_{\perp}(\mathbf{k})$ from h_{s0}^0 (h0 parameter) and h_{s0}^1 (h1 parameter).

2.5.2.5 fields_getChi()

```
void fields_getChi ( )
```

computes gyrokinetic potentials chi

Wrapper for functions [fields_getChiPhi](#), [fields_getChiA](#), [fields_getChiB](#)

2.5.2.6 fields_getChiA()

```
void fields_getChiA ( )
```

computes chiA gyrokinetic potential from A potential

computes $chi_s^A(\mathbf{k})$

2.5.2.7 fields_getChiB()

```
void fields_getChiB ( )
```

computes chiB gyrokinetic potential from B potential

computes $\chi_s^B(\mathbf{k})$

2.5.2.8 fields_getChiPhi()

```
fields_getChiPhi ( )
```

computes chiPhi gyrokinetic potential from phi potential

computes $chi_s^{\phi}(\mathbf{k})$

2.5.2.9 fields_getFields()

```
void fields_getFields (
    COMPLEX * g00,
    COMPLEX * g10,
    COMPLEX * g01 )
```

wrapper to get all the fields simultaneously

Parameters

<i>g00</i>	4D complex array (kx,ky,kz,s). Zeroth Hermite and Laguerre moment of modified gyrokinetic distribution function.
<i>g10</i>	4D complex array (kx,ky,kz,s). Must be first Hermite and zeroth Laguerre moment of modified gyrokinetic distribution function g.
<i>g01</i>	4D complex array (kx,ky,kz,s). Zeroth Hermite and first Laguerre moment of the modified distribution function.

Wrapper for functions [fields_getPhi](#), [fields_getB](#), [fields_getA](#).

2.5.2.10 fields_getFieldsFromH()

```
void fields_getFieldsFromH (
    COMPLEX * h00,
    COMPLEX * h10,
    COMPLEX * h01 )
```

wrapper to get all the fields simultaneously, computed from gyrokinetic distribution function

Parameters

<i>h00</i>	4D complex array (kx,ky,kz,s). Zeroth Hermite and Laguerre moment of gyrokinetic distribution function.
<i>h10</i>	4D complex array (kx,ky,kz,s). Must be first Hermite and zeroth Laguerre moment of gyrokinetic distribution function h.
<i>h01</i>	4D complex array (kx,ky,kz,s). Zeroth Hermite and first Laguerre moment of the distribution function.

Wrapper for functions [fields_getPhiFromH](#), [#fields_get_BFromH](#), [fields_getAFromH](#).

2.5.2.11 fields_getGradX()

```
void fields_getGradX (
    COMPLEX * out )
```

computes chi gradient in x direction

Parameters

<i>out</i>	output complex array of size (kx,ky,kz,s,Nfields).
------------	--

computes gradient in x direction for chi potentials. Nfields<\tt> can be 1 or 3, and chosen automatically at start of the simulation depending on the simulation type (electrostatic or electromagnetic)

2.5.2.12 fields_getGradY()

```
void fields_getGradY (
    COMPLEX * out )
```

computes chi gradient in y direction

Parameters

<i>out</i>	output complex array of size (kx,ky,kz,s,Nfields).
------------	--

computes gradient in y direction for chi potentials. Nfields<\tt> can be 1 or 3, and chosen automatically at start of the simulation depending on the simulation type

(electrostatic or electromagnetic)

2.5.2.13 fields_getPhi()

```
void fields_getPhi (
    const COMPLEX * g0,
    const COMPLEX * g1 )
```

computes phi potential

Parameters

<i>g0</i>	4D complex array (kx,ky,kz,s). Zeroth Hermite and Laguerre moment of modified gyrokinetic distribution function.
<i>g1</i>	4D complex array (kx,ky,kz,s). Zeroth Hermite and first Laguerre moment of the modified distribution function.

Computes $\phi(\mathbf{k})$ from g_{s0}^0 (*g0* parameter) and g_{s0}^1 (*g1* parameter).

2.5.2.14 fields_getPhiFromH()

```
void fields_getPhiFromH (
    const COMPLEX * h )
```

computes phi potential

Parameters

<i>h</i>	4D complex array (kx,ky,kz,s). Zeroth Hermite and Laguerre moment of gyrokinetic distribution function.
----------	---

Computes $\phi(\mathbf{k})$ from h_{s0}^0 (*h* parameter)

2.5.2.15 fields_init()

```
void fields_init ( )
```

initializes fields

pre-computes some constants required to compute fields. Called in [init_start](#) function

2.5.2.16 fields_sendG()

```
fields_sendG (
    COMPLEX * g )
```

sends moments of gyrokinetic distribution function which are required to compute fields

Parameters

g	complex array. Modified or non-modified gyrokinetic distribution function
-----	---

sends $g_{s0}^1(\mathbf{k})f$, $g_{s1}^0(\mathbf{k})f$, $g_{s0}^0(\mathbf{k})f$

to all processes to compute potentials locally.

2.6 init.c File Reference

initialization module for alliance.

```
#include "init.h"
#include "distrib.h"
```

Macros

- `#define RANK_IO 0`

Functions

- void `init_start` (char *filename)
initialization of ALLIANCE
- void `init_printParameters` ()
parameter output
- void `init_initEnums` ()
enumerator initialization
- void `fill_rand` (COMPLEX *ar1)
fills the initial conditions randomly
- void `fill_randM0` (COMPLEX *ar1)
fill zeroth Hermite moment with random values
- void `fill_randSingleKM` (COMPLEX *ar1)
fill single chosen wavevector and Hermite moment
- void `init_conditions` (COMPLEX *data)
distribution function initialization
- double `init_energySpec` (double k, double m, double amp, double disp)
returns energy spectrum

Variables

- enum adiabatic `kinetic`
- enum electromagnetic `systemType`
- enum initial `initialConditions`

2.6.1 Detailed Description

initialization module for alliance.

all the initialization routines are here.

2.6.2 Macro Definition Documentation

2.6.2.1 RANK_IO

```
#define RANK_IO 0
```

defines rank of the processor used to output information to console

2.6.3 Function Documentation

2.6.3.1 fill_rand()

```
void fill_rand (
    COMPLEX * data )
```

fills the initial conditions randomly

Parameters

<i>data</i>	complex 6d array to fill initializes distribution with spectrum defined in init_energySpec This function is supposed to be used in-module only and should not be used elsewhere outside init.c file.
-------------	--

2.6.3.2 fill_randM0()

```
void fill_randM0 (
    COMPLEX * data )
```

fill zeroth Hermite moment with random values

Parameters

<i>data</i>	complex 6D array to fill
-------------	--------------------------

fills 0-th Hermite moment of a distribution function `ar1` with random values. This function is supposed to be used in-module only and should not be used elsewhere outside `init.c` file.

2.6.3.3 `fill_randSingleKM()`

```
void fill_randSingleKM (
    COMPLEX * ar1 )
```

fill single chosen wavevector and Hermite moment

Parameters

<i>data</i>	complex 6D array
-------------	------------------

initializes single wavevector and Hermite moment of a distribution function with random variable. This function is only for in-module use and should not be used elsewhere outside `init.c` file.

2.6.3.4 `init_conditions()`

```
void init_conditions (
    COMPLEX * data )
```

distribution function initialization

Parameters

<i>data</i>	complex 6D array
-------------	------------------

initializes distribution function with chosen method (see `fill_rand`, `fill_randM0`, `fill_randSingleKM`)

2.6.3.5 `init_energySpec()`

```
double init_energySpec (
    double k,
    double m,
    double amp,
    double disp )
```

returns energy spectrum

Parameters

<i>k</i>	a wavenumber at which spectrum is computed
<i>m</i>	Hermite moment at which amplitude is computed
<i>amp</i>	amplitude of the spectrum
<i>disp</i>	dispersion of the spectrum

computes spectrum of form $A \cdot k^2 \exp(-2k^2/\sigma^2)$, where $\sigma = \text{disp}$, and $A = \text{amp}$. This function is supposed to be

used in-module only and should not be used elsewhere outside [init.c](#) file.

2.6.3.6 init_initEnums()

```
void init_initEnums ( )
```

enumerator initialization

initializes enumerators, which are then used to define if system is adiabatic or kinetic, electromagnetic or electrostatic, and type of initial conditions

2.6.3.7 init_printParameters()

```
void init_printParameters ( )
```

parameter output

prints parameters of the simulation

2.6.3.8 init_start()

```
void init_start (
    char * filename )
```

initialization of ALLIANCE

Parameters

<i>filename</i>	specifies parameter filename
-----------------	------------------------------

initializes all the modules required for ALLIANCE to work.

2.7 parameters_io.c File Reference

reads input parameters from parameter file provided by user

```
#include "parameters_io.h"
#include "utils_fftw.h"
```

Macros

- `#define VERBOSE 0`

Functions

- void `init_global_size` ()
initializes global size of the 6D array
- void `read_parameters` (char *filename)
reads parameters from user parameter file.
- void `read_parametersFromFile` (char *filename)

Variables

- struct system_param **parameters**

2.7.1 Detailed Description

reads input parameters from parameter file provided by user

2.7.2 Function Documentation

2.7.2.1 `init_global_size()`

```
void init_global_size ( )
```

initializes global size of the 6D array

initializes `array_local_size` structure with global simulation size.

2.7.2.2 `read_parameters()`

```
read_parameters (
    char * filename )
```

reads parameters from user parameter file.

Reads parameters from user parameter file. All the parameters are stored in the `parameters` structure

2.7.2.3 `read_parametersFromFile()`

```
void read_parametersFromFile (
    char * filename )
```

`read_parametersFromFile(char *filename):`

2.8 solver.c File Reference

numerical solver

```
#include "solver.h"
```

Macros

- `#define SOLVERTYPE RK4`
- `#define IORANK 0`

Functions

- void `solver_init` ()
initializes solver
- void `solver_makeStep` (COMPLEX **g, COMPLEX *h)
iterate solver forward

Variables

- enum solverType `solverType`
- struct solver `solver`
- struct rk4 `rk4`

2.8.1 Detailed Description

numerical solver

2.8.2 Function Documentation

2.8.2.1 `solver_init()`

```
void solver_init ( )
```

initializes solver

initializes solver with the `solverType`.

2.8.2.2 `solver_makeStep()`

```
void solver_makeStep (  
    COMPLEX ** g,  
    COMPLEX * h )
```

iterate solver forward

Parameters

<i>g</i>	address of the 6D complex array. Modified gyrokinetic distribution function
<i>h</i>	6D complex array. Gyrokinetic distribution function

solves one simulation time step

2.9 space_config.c File Reference

space configuration module

```
#include "space_config.h"
#include <complex.h>
```

Macros

- #define **MINUS_I** -1.j

Functions

- void [space_init](#) ()
initializes wave space. Called in [init_start\(\)](#) function.
- void [space_generateWaveSpace](#) ()
generates wave space.
- void [space_generateMSpace](#) ()
generates Hermite space.
- void [free_wavespace](#) ()
deallocates all the arrays.

Variables

- double **space_Lx** = 100
- double **space_Ly** = 100
- double **space_Lz** = 100
- double * **space_kx**
- double * **space_ky**
- double * **space_kz**
- double * **space_kPerp**
- double * **space_kPerp2**
- double * **space_kSq**
- double * **space_sqrtM**
- size_t * **space_globalMIndex**
- COMPLEX * **space_iKx**
- COMPLEX * **space_iKy**
- COMPLEX * **space_iKz**

2.9.1 Detailed Description

space configuration module

creates k and m spaces

2.9.2 Function Documentation

2.9.2.1 free_wavespace()

```
free_wavespace ( )
```

deallocates all the arrays.

to be added...

2.9.2.2 space_generateMSpace()

```
space_generateMSpace ( )
```

generates Hermite space.

to be added

2.9.2.3 space_generateWaveSpace()

```
void space_generateWaveSpace ( )
```

generates wave space.

generates wave number arrays space_kx, space_ky, space_kz of lengths nkx,nky,nkz for a numerical box of size [lx, ly, lz] in kx,ky,kz directions as following:

$[0, \pi / l_x, 2 \pi / l_x, \dots, (n / 2 + 1) \pi / l_x, - (n / 2) \pi / l_x, \dots, - \pi / l_x]$ generates arrays space_iKx, space_iKy, space_iKz, of lengths nkx,nky,nkz. These arrays are later used to compute gradients by [fields_getGradX](#), [fields_getGradY](#), [distrib_getXGrad](#), [distrib_getYGrad](#), [distrib_getZGrad](#).

2.10 utils_fftw.c File Reference

FFT module.

```
#include "utils_fftw.h"
```

Macros

- `#define FFTW_RANK 3`
- `#define CHI_EL 1`
- `#define CHI_EM 3`
- `#define VERBOSE 0`

Functions

- void `fftw_init` (MPI_Comm communicator)
initializes fftw transform.
- void `fftw_r2c` ()
real to complex fft transform.
- void `fftw_c2r` ()
complex to real fft transform.
- void `fftw_r2c_chi` ()
real to complex transform of chi potentials
- void `fftw_c2r_chi` ()
complex to real transform of chi potentials
- void `fftw_r2c_field` ()
real to complex transform of field potentials
- void `fftw_c2r_field` ()
complex to real transform of field potentials
- void `fftw_kill` ()
kills fftw
- void `fftw_copy_buffer_r` (double *to, double *from)
copy 6D real array
- void `fftw_copy_buffer_c` (COMPLEX *to, COMPLEX *from)
copy 6D complex array
- void `fftw_copyChiBuf_r` (double *ar1, double *ar2)
copy 5D real array
- void `fftw_copyChiBuf_c` (COMPLEX *ar1, COMPLEX *ar2)
copy 5D complex array.
- void `fftw_copyFieldBuf_r` (double *to, double *from)
copy 3D real data array.
- void `fftw_copyFieldBuf_c` (COMPLEX *to, COMPLEX *from)
copy 3D complex data array.
- double `cosinus` (double f, int ix)
- void `fftw_test_fill` (double *ar, double f)
- void `fftw_normalise_data` (COMPLEX *data)
- void `fftw_normalise_data_r` (double *data)
normalise data.
- void `fftw_normalise_chi_r` (double *data)
normalise chi data
- void `fftw_normalise_field_r` (double *data)
normalise real 3D data
- void `dealiasing23` (COMPLEX *data_c)
2/3 rule dealiasing

Variables

- ptrdiff_t **size_c** [3]
- ptrdiff_t **size_r** [3]
- ptrdiff_t **howmany**
- ptrdiff_t **howmany_chi**
- ptrdiff_t **local_size**
- ptrdiff_t **local_n0**
- ptrdiff_t **local_0_start**
- ptrdiff_t **local_size_chi**
- ptrdiff_t **local_n0_chi**
- ptrdiff_t **local_0_start_chi**
- ptrdiff_t **local_size_field**
- ptrdiff_t **local_n0_field**
- ptrdiff_t **local_0_start_field**
- fftw_plan **plan_c2r**
- fftw_plan **plan_r2c**
- fftw_plan **plan_c2r_chi**
- fftw_plan **plan_r2c_chi**
- fftw_plan **plan_c2r_field**
- fftw_plan **plan_r2c_field**
- COMPLEX * **fftw_hBuf**
- COMPLEX * **fftw_chiBuf**
- COMPLEX * **fftw_field**
- double **fftw_norm**
- void(* **fftw_dealiasing**)(COMPLEX *) = NULL
- int * **global_nkx_index**

2.10.1 Detailed Description

FFT module.

contains FFT related routines

2.10.2 Function Documentation

2.10.2.1 **cosinus()**

```
double cosinus (
    double f,
    int ix )
```

[cosinus\(double f,int ix\)](#)

2.10.2.2 **dealiasing23()**

```
void dealiasing23 (
    COMPLEX * data_c )
```

2/3 rule dealiasing

Parameters

<i>data</i> ↔ _c	complex 6D data array
---------------------	-----------------------

2.10.2.3 fftw_c2r()

```
fftw_c2r ( )
```

complex to real fft transform.

Performs complex to real in-place fft transform on array `fftw_hBuf`. Used to transform 6D arrays (x,y,z,m,l,s).

2.10.2.4 fftw_c2r_chi()

```
void fftw_c2r_chi ( )
```

complex to real transform of chi potentials

Performs complex to real in-place fft transform on array `fftw_chiBuf`. Used to transform 5D arrays (kx,ky,kz,s,field).

2.10.2.5 fftw_c2r_field()

```
void fftw_c2r_field ( )
```

complex to real transform of field potentials

Performs complex to real in-place fft transform on array `fftw_field`. Used to transform 3D arrays (kx,ky,kz).

2.10.2.6 fftw_copy_buffer_c()

```
fftw_copy_buffer_c (
    COMPLEX * to,
    COMPLEX * from )
```

copy 6D complex array

Parameters

<i>to</i>	where to copy array
<i>from</i>	array which will be copied

copies complex data `from` array to array `to`

2.10.2.7 fftw_copy_buffer_r()

```
fftw_copy_buffer_r (
    double * to,
    double * from )
```

copy 6D real array

Parameters

<i>to</i>	where to copy array
<i>from</i>	array which will be copied

copies real data *from* array to array *to*

2.10.2.8 fftw_copyChiBuf_c()

```
fftw_copyChiBuf_c (
    COMPLEX * ar1,
    COMPLEX * ar2 )
```

copy 5D complex array.

Parameters

<i>ar1</i>	destination
<i>ar2</i>	source

copies complex χ array from *ar1* to *ar2*.

2.10.2.9 fftw_copyChiBuf_r()

```
fftw_copyChiBuf_r (
    double * ar1,
    double * ar2 )
```

copy 5D real array

Parameters

<i>ar1</i>	destination
<i>ar2</i>	source

copies real χ array from *ar1* to *ar2*.

2.10.2.10 fftw_copyFieldBuf_c()

```
fftw_copyFieldBuf_c (
```

```

COMPLEX * to,
COMPLEX * from )

```

copy 3D complex data array.

Parameters

<i>to</i>	
<i>from</i>	copies 3D complex data array <i>from</i> to

2.10.2.11 `fftw_copyFieldBuf_r()`

```

fftw_copyFieldBuf_r (
    double * to,
    double * from )

```

copy 3D real data array.

Parameters

<i>to</i>	
<i>from</i>	copies 3D data array <i>from</i> to

2.10.2.12 `fftw_kill()`

```

void fftw_kill ( )

```

kills fftw

to be added

2.10.2.13 `fftw_normalise_chi_r()`

```

void fftw_normalise_chi_r (
    double * data )

```

normalise chi data

Parameters

<i>data</i>	5D real data array
-------------	--------------------

normalises *data* by `#fftw_norm`.

2.10.2.14 fftw_normalise_data()

```
void fftw_normalise_data (
    COMPLEX * data )
```

fftw_normalise_data(double *data)

2.10.2.15 fftw_normalise_data_r()

```
void fftw_normalise_data_r (
    double * data )
```

normalise data.

Parameters

<i>data</i>	6D data array
-------------	---------------

normalises *data* by #fftw_norm.

2.10.2.16 fftw_normalise_field_r()

```
void fftw_normalise_field_r (
    double * data )
```

normalise real 3D data

Parameters

<i>data</i>	3D real array
-------------	---------------

normalises *data* by #fftw_norm.

2.10.2.17 fftw_r2c()

```
fftw_r2c ( )
```

real to complex fft transform.

Performs real to complex in-place fft transform of on array *fftw_hBuf*. Used to transform 6D arrays (kx,ky,kz,m,l,s).

2.10.2.18 fftw_r2c_chi()

```
void fftw_r2c_chi ( )
```

real to complex transform of chi potentials

Performs real to complex in-place fft transform on array *fftw_chiBuf*. Used to transform 5D arrays (x,y,z,s,field).

2.10.2.19 `fftw_r2c_field()`

```
void fftw_r2c_field ( )
```

real to complex transform of field potentials

Performs real to complex in-place fft transform on array `fftw_field`. Used to transform 3D arrays (x,y,z).

2.10.2.20 `fftw_test_fill()`

```
void fftw_test_fill (
    double * ar,
    double f )
```

[fftw_test_fill\(double *ar,double f\)](#)

2.11 `utils_hdf.c` File Reference

hdf module

```
#include "utils_hdf.h"
#include <unistd.h>
#include <sys/stat.h>
```

Macros

- `#define BASE_DIR "."`
- `#define WORK_DIR "."`
- `#define CHCK_DIR "checkpoint"`
- `#define CHCK_NAME "chk"`
- `#define PATH_SEPARATOR "/"`
- `#define VERBOSE 1`
- `#define FILENAME_ID_LEN 128`
- `#define CHECKPOINT_ROOT 0`
- `#define PATH_LEN 128`

Functions

- void `complex_t_init()`
- void `hdf_init()`
- void `hdf_createSaveDirs()`
- void `hdf_create_file_c` (char *filename, COMPLEX *data)
- void `hdf_create_file_r` (char *filename, double *data)
- void `hdf_initChi()`
- void `hdf_createChiFile_r` (char *filename, double *data)
- void `hdf_createChiFile_c` (char *filename, COMPLEX *data)
- void `hdf_initField()`
- void `hdf_saveFieldA` (char *filename)
- void `hdf_saveField_r` (double *f, char *filename)

- void [hdf_saveFieldB](#) (char *filename)
- void [hdf_saveFieldPhi](#) (char *filename)
- void [hdf_saveEnergy](#) (int timestep)
- void [hdf_saveData](#) (COMPLEX *h, int timestep)
- void [hdf_createParamFile](#) ()
- void [hdf_createFiles](#) ()
- void [hdf_saveKSpec](#) (int timestep)
- void [hdf_saveMSpec](#) (int timestep)
- void [hdf_initCheckpoints](#) ()
- void [hdf_createCheckpoint](#) (COMPLEX *h, int timestep)
- void [hdf_dumpCheckpoint](#) (COMPLEX *h, int timestep, char *filename)
- void [hdf_saveDistrib](#) (COMPLEX *h, int timestep)
- void [hdf_createFieldFile](#) ()
- void [hdf_saveFields](#) (int timestep)
- void [hdf_readData](#) (char *filename, COMPLEX *h)

Variables

- int **hdf_rank** = 6
- int **hdf_rankFields** = 3
- int **hdf_rankChi** = 5
- int **hdf_freeEnergyCalls** = 0
- char ** **hdf_checkpointNames**
- char **hdf_newCheckpointName** [FILENAME_ID_LEN]
- char **SIMULATION_PATH** [PATH_LEN]
- char **CHECKPOINT_PATH** [PATH_LEN]
- char **PARAMETER_FILENAME** [FILENAME_ID_LEN]
- char **DISTRIBUTION_FILENAME** [FILENAME_ID_LEN]
- char **FIELD_FILENAME** [FILENAME_ID_LEN]
- size_t **hdf_checkpointCount** = 0
- hid_t **complex_id**
- hsize_t **dataspace_dims_r** [6]
- hsize_t **dataspace_dims_c** [6]
- hsize_t **dataspace_dimsFields** [3]
- hsize_t **dataspace_dimsFields_r** [3]
- hsize_t **dataspace_dimsChi** [5]
- hsize_t **dataspace_dimsChi_r** [5]
- hsize_t **chunk_dims_r** [6]
- hsize_t **chunk_dims_c** [6]
- hsize_t **chunk_dimsFields** [3]
- hsize_t **chunk_dimsFields_r** [3]
- hsize_t **chunk_dimsChi** [5]
- hsize_t **chunk_dimsChi_r** [5]
- hsize_t **offset** [6]
- hsize_t **offsetFields** [3]
- hsize_t **offsetFields_r** [3]
- hsize_t **offsetChi** [5]
- hsize_t **offsetChi_r** [5]
- hsize_t **count** [6] = {1,1,1,1,1,1}
- hsize_t **stride** [6] = {1,1,1,1,1,1}
- hsize_t **countFields** [3] = {1,1,1}
- hsize_t **strideFields** [3] = {1,1,1}
- hsize_t **countChi** [5] = {1,1,1,1,1}
- hsize_t **strideChi** [5] = {1,1,1,1,1}
- herr_t **status**
- MPI_Info **info** = MPI_INFO_NULL
- complex_t **tmp**

2.11.1 Detailed Description

hdf module

contains HDF related routines to save and read hdf files

2.11.2 Function Documentation

2.11.2.1 hdf_create_file_c()

```
void hdf_create_file_c (
    char * filename,
    COMPLEX * data )
```

hdf_create_file_c

2.11.2.2 hdf_create_file_r()

```
void hdf_create_file_r (
    char * filename,
    double * data )
```

hdf_create_file_r

2.11.2.3 hdf_createCheckpoint()

```
void hdf_createCheckpoint (
    COMPLEX * h,
    int timestep )
```

hdf_createCheckpoint

2.11.2.4 hdf_createChiFile_c()

```
void hdf_createChiFile_c (
    char * filename,
    COMPLEX * data )
```

hdf_createChiFile_c

2.11.2.5 hdf_createChiFile_r()

```
void hdf_createChiFile_r (
    char * filename,
    double * data )
```

hdf_createChiFile_r

2.11.2.6 hdf_createFieldFile()

```
void hdf_createFieldFile ( )
```

FIELD FILE `hdf_createFieldFile`

2.11.2.7 hdf_createFiles()

```
void hdf_createFiles ( )
```

`hdf_createFiles`

2.11.2.8 hdf_createParamFile()

```
void hdf_createParamFile ( )
```

PARAMETER FILE `hdf_createParamFile`

2.11.2.9 hdf_createSaveDirs()

```
void hdf_createSaveDirs ( )
```

`hdf_createSaveDirs`

2.11.2.10 hdf_dumpCheckpoint()

```
void hdf_dumpCheckpoint (
    COMPLEX * h,
    int timestep,
    char * filename )
```

`hdf_dumpCheckpoint`

2.11.2.11 hdf_init()

```
void hdf_init ( )
```

INITIALIZE HDF5 `hdf_init`

2.11.2.12 hdf_initCheckpoints()

```
void hdf_initCheckpoints ( )
```

CHECKPOINTS `hdf_initCheckpoints`

2.11.2.13 hdf_initChi()

```
void hdf_initChi ( )
```

`hdf_initChi`

2.11.2.14 hdf_initField()

```
void hdf_initField ( )
```

`hdf_initField`

2.11.2.15 hdf_readData()

```
void hdf_readData (
    char * filename,
    COMPLEX * h )
```

READ FILE `hdf_readData`

2.11.2.16 hdf_saveData()

```
void hdf_saveData (
    COMPLEX * h,
    int timestep )
```

`hdf_saveData`

2.11.2.17 hdf_saveEnergy()

```
void hdf_saveEnergy (
    int timestep )
```

`hdf_saveEnergy`

2.11.2.18 hdf_saveField_r()

```
void hdf_saveField_r (
    double * f,
    char * filename )
```

`hdf_saveField_r`

2.11.2.19 hdf_saveFieldA()

```
void hdf_saveFieldA (
    char * filename )
```

`hdf_saveFieldA`

2.11.2.20 hdf_saveFieldB()

```
void hdf_saveFieldB (
    char * filename )
```

`hdf_saveFieldB`

2.11.2.21 hdf_saveFieldPhi()

```
void hdf_saveFieldPhi (
    char * filename )
```

`hdf_saveFieldPhi`

2.11.2.22 hdf_saveFields()

```
void hdf_saveFields (
    int timestep )
```

`hdf_saveFields`

2.11.2.23 hdf_saveKSpec()

```
void hdf_saveKSpec (
    int timestep )
```

`hdf_saveKSpec`

2.11.2.24 hdf_saveMSpec()

```
void hdf_saveMSpec (
    int timestep )
```

`hdf_saveMSpec`

2.12 utils_mpi.c File Reference

mpi module

```
#include "utils_mpi.h"
```

Macros

- `#define VERBOSE 0`
- `#define SUBARRAY_COUNT 1`
- `#define SUBARRAY_M_SIZE 1`
- `#define SUBARRAY_DIMS 6`

Enumerations

- enum **DIRECTIONS** { **MINUS** , **PLUS** }

Functions

- void [mpi_init](#) ()
- void [mpi_generateTopology](#) ()
- void [mpi_kill](#) ()
- void [mpi_createTopology](#) ()
- void [mpi_getLocalArraySize](#) ()
- void [mpi_getLocalArrayOffsets](#) ()
- void [mpi_findHermiteNeighbours](#) ()
- void [mpi_splitInRows](#) ()
- void [mpi_splitInCols](#) ()
- void [mpi_initMExchange](#) ()
- void [mpi_exchangeMBoundaries](#) (COMPLEX *input_array, COMPLEX *plus_boundary, COMPLEX *minus_boundary)
- void [mpi_exchangeMBoundaries_r](#) (double *input_array, double *plus_boundary, double *minus_boundary)
- void [mpi_sendVector](#) (COMPLEX *from_array, COMPLEX *to_buffer, int from_proc, int to_proc)

Variables

- int **mpi_my_rank**
- int **mpi_size**
- int **mpi_my_row_rank**
- int **mpi_my_col_rank**
- int **mpi_my_coords** [2]
- int **mpi_dims** [] = {0, 0}
- int **m_neighbour_ranks** [2]
- int **mpi_sub_buf_size**
- int **mpi_sub_buf_size_r**
- int * **mpi_whereIsX**
- size_t **mpi_vectorSliceLength**
- MPI_Comm **mpi_cube_comm**
- MPI_Comm **mpi_row_comm**
- MPI_Comm **mpi_col_comm**
- MPI_Datatype **mpi_subarray_type_plus**
- MPI_Datatype **mpi_subarray_type_minus**
- MPI_Datatype **mpi_subarray_type_plus_r**
- MPI_Datatype **mpi_subarray_type_minus_r**
- MPI_Datatype **mpi_vector_kxSlice**

2.12.1 Detailed Description

mpi module

module to generate mpi topology and other mpi related routines

2.12.2 Function Documentation

2.12.2.1 mpi_createTopology()

```
void mpi_createTopology ( )
```

mpi_createTopology

2.12.2.2 mpi_exchangeMBoundaries()

```
void mpi_exchangeMBoundaries (
    COMPLEX * input_array,
    COMPLEX * plus_boundary,
    COMPLEX * minus_boundary )
```

mpi_exchangeMBoundaries

2.12.2.3 mpi_exchangeMBoundaries_r()

```
void mpi_exchangeMBoundaries_r (
    double * input_array,
    double * plus_boundary,
    double * minus_boundary )
```

mpi_exchangeMBoundaries_r

2.12.2.4 mpi_findHermiteNeighbours()

```
void mpi_findHermiteNeighbours ( )
```

mpi_findHermiteNeighbours

2.12.2.5 mpi_generateTopology()

```
void mpi_generateTopology ( )
```

mpi_generateTopology

2.12.2.6 mpi_getLocalArrayOffsets()

```
void mpi_getLocalArrayOffsets ( )
```

mpi_getLocalArrayOffsets

2.12.2.7 `mpi_getLocalArraySize()`

```
void mpi_getLocalArraySize ( )
```

`mpi_getLocalArraySize`

2.12.2.8 `mpi_init()`

```
void mpi_init ( )
```

`mpi_init`

2.12.2.9 `mpi_initMExchange()`

```
void mpi_initMExchange ( )
```

`mpi_initMExchange`

2.12.2.10 `mpi_kill()`

```
void mpi_kill ( )
```

`mpi_kill`

2.12.2.11 `mpi_sendVector()`

```
void mpi_sendVector (
    COMPLEX * from_array,
    COMPLEX * to_buffer,
    int from_proc,
    int to_proc )
```

`mpi_sendVector(COMPLEX *from_array, COMPLEX *to_buffer, int to_proc)`

2.12.2.12 `mpi_splitInCols()`

```
void mpi_splitInCols ( )
```

`mpi_splitInCols`

2.12.2.13 `mpi_splitInRows()`

```
void mpi_splitInRows ( )
```

`mpi_splitInRows`

2.13 variables.c File Reference

stores physical variables

```
#include "variables.h"
```

Functions

- void `var_init` ()
initializes variables
- void `var_getJ0` ()
generates J0.
- void `var_getJ1` ()
generates J1.
- size_t `var_getJIndex` (size_t ikx, size_t iky, size_t is)
returns index to get data from #var_J0 and #var_J1
- void `var_varlnit` ()
initializes physical variables

Variables

- struct phys_params `var_var`
- double * `var_J0`
- double * `var_J1`

2.13.1 Detailed Description

stores physical variables

2.13.2 Function Documentation

2.13.2.1 var_getJ0()

```
void var_getJ0 ( )
```

generates J0.

generates zeroth Laguerre moment #var_J0 of Bessel function used for gyroaveraging.

2.13.2.2 var_getJ1()

```
void var_getJ1 ( )
```

generates J1.

generates first Laguerre moment #var_J1 of Bessel function used for gyroaveraging.

Index

- array.c, [3](#)
 - get_flat_c, [4](#)
 - get_flat_r, [4](#)
 - get_flatIndexComplex3D, [5](#)
 - getIndChi, [5](#)
 - getIndChiBufEL_c, [5](#)
 - getIndChiBufEL_r, [7](#)
 - getIndChiBufEM_c, [7](#)
 - getIndChiBufEM_r, [8](#)
 - multiply_ar_c, [8](#)
 - multiply_ar_r, [8](#)
- cosinus
 - utils_fftw.c, [33](#)
- dealiasing23
 - utils_fftw.c, [33](#)
- diag_compute
 - diagnostics.c, [9](#)
- diag_computeFreeEnergy
 - diagnostics.c, [10](#)
- diag_computeFreeEnergyFields
 - diagnostics.c, [10](#)
- diag_computeKSpectrum
 - diagnostics.c, [10](#)
- diag_computeMSpectrum
 - diagnostics.c, [11](#)
- diag_computeSpectra
 - diagnostics.c, [11](#)
- diag_getShells
 - diagnostics.c, [11](#)
- diag_initSpec
 - diagnostics.c, [11](#)
- diagnostics.c, [8](#)
 - diag_compute, [9](#)
 - diag_computeFreeEnergy, [10](#)
 - diag_computeFreeEnergyFields, [10](#)
 - diag_computeKSpectrum, [10](#)
 - diag_computeMSpectrum, [11](#)
 - diag_computeSpectra, [11](#)
 - diag_getShells, [11](#)
 - diag_initSpec, [11](#)
- distrib.c, [12](#)
 - distrib_enforceReality, [12](#)
 - distrib_getG, [13](#)
 - distrib_getH, [13](#)
 - distrib_getXGrad, [13](#)
 - distrib_getYGrad, [14](#)
 - distrib_getZGrad, [14](#)
 - distrib_setZeroNHalf, [14](#)
- distrib_enforceReality
 - distrib.c, [12](#)
- distrib_getG
 - distrib.c, [13](#)
- distrib_getH
 - distrib.c, [13](#)
- distrib_getXGrad
 - distrib.c, [13](#)
- distrib_getYGrad
 - distrib.c, [14](#)
- distrib_getZGrad
 - distrib.c, [14](#)
- distrib_setZeroNHalf
 - distrib.c, [14](#)
- equation.c, [15](#)
 - equation_getLinearTerm, [16](#)
 - equation_getNonlinearElectromagnetic, [16](#)
 - equation_getNonlinearElectrostatic, [16](#)
 - equation_getNonlinearProduct, [17](#)
 - equation_getNonlinearTerm, [17](#)
- equation_getLinearTerm
 - equation.c, [16](#)
- equation_getNonlinearElectromagnetic
 - equation.c, [16](#)
- equation_getNonlinearElectrostatic
 - equation.c, [16](#)
- equation_getNonlinearProduct
 - equation.c, [17](#)
- equation_getNonlinearTerm
 - equation.c, [17](#)
- fftw_c2r
 - utils_fftw.c, [34](#)
- fftw_c2r_chi
 - utils_fftw.c, [34](#)
- fftw_c2r_field
 - utils_fftw.c, [34](#)
- fftw_copy_buffer_c
 - utils_fftw.c, [34](#)
- fftw_copy_buffer_r
 - utils_fftw.c, [34](#)
- fftw_copyChiBuf_c
 - utils_fftw.c, [35](#)
- fftw_copyChiBuf_r
 - utils_fftw.c, [35](#)
- fftw_copyFieldBuf_c
 - utils_fftw.c, [35](#)
- fftw_copyFieldBuf_r
 - utils_fftw.c, [36](#)

- fftw_kill
 - utils_fftw.c, 36
- fftw_normalise_chi_r
 - utils_fftw.c, 36
- fftw_normalise_data
 - utils_fftw.c, 36
- fftw_normalise_data_r
 - utils_fftw.c, 37
- fftw_normalise_field_r
 - utils_fftw.c, 37
- fftw_r2c
 - utils_fftw.c, 37
- fftw_r2c_chi
 - utils_fftw.c, 37
- fftw_r2c_field
 - utils_fftw.c, 37
- fftw_test_fill
 - utils_fftw.c, 38
- fields.c, 18
 - fields_getA, 19
 - fields_getAFromH, 20
 - fields_getB, 20
 - fields_getBFromH, 20
 - fields_getChi, 21
 - fields_getChiA, 21
 - fields_getChiB, 21
 - fields_getChiPhi, 21
 - fields_getFields, 21
 - fields_getFieldsFromH, 22
 - fields_getGradX, 22
 - fields_getGradY, 22
 - fields_getPhi, 23
 - fields_getPhiFromH, 23
 - fields_init, 23
 - fields_sendG, 23
- fields_getA
 - fields.c, 19
- fields_getAFromH
 - fields.c, 20
- fields_getB
 - fields.c, 20
- fields_getBFromH
 - fields.c, 20
- fields_getChi
 - fields.c, 21
- fields_getChiA
 - fields.c, 21
- fields_getChiB
 - fields.c, 21
- fields_getChiPhi
 - fields.c, 21
- fields_getFields
 - fields.c, 21
- fields_getFieldsFromH
 - fields.c, 22
- fields_getGradX
 - fields.c, 22
- fields_getGradY
 - fields.c, 22
- fields_getPhi
 - fields.c, 23
- fields_getPhiFromH
 - fields.c, 23
- fields_init
 - fields.c, 23
- fields_sendG
 - fields.c, 23
- fill_rand
 - init.c, 25
- fill_randM0
 - init.c, 25
- fill_randSingleKM
 - init.c, 26
- free_wavespace
 - space_config.c, 31
- get_flat_c
 - array.c, 4
- get_flat_r
 - array.c, 4
- get_flatIndexComplex3D
 - array.c, 5
- getIndChi
 - array.c, 5
- getIndChiBufEL_c
 - array.c, 5
- getIndChiBufEL_r
 - array.c, 7
- getIndChiBufEM_c
 - array.c, 7
- getIndChiBufEM_r
 - array.c, 8
- hdf_create_file_c
 - utils_hdf.c, 40
- hdf_create_file_r
 - utils_hdf.c, 40
- hdf_createCheckpoint
 - utils_hdf.c, 40
- hdf_createChiFile_c
 - utils_hdf.c, 40
- hdf_createChiFile_r
 - utils_hdf.c, 40
- hdf_createFieldFile
 - utils_hdf.c, 40
- hdf_createFiles
 - utils_hdf.c, 41
- hdf_createParamFile
 - utils_hdf.c, 41
- hdf_createSaveDirs
 - utils_hdf.c, 41
- hdf_dumpCheckpoint
 - utils_hdf.c, 41
- hdf_init
 - utils_hdf.c, 41
- hdf_initCheckpoints
 - utils_hdf.c, 41

- hdf_initChi
 - utils_hdf.c, [41](#)
- hdf_initField
 - utils_hdf.c, [42](#)
- hdf_readData
 - utils_hdf.c, [42](#)
- hdf_saveData
 - utils_hdf.c, [42](#)
- hdf_saveEnergy
 - utils_hdf.c, [42](#)
- hdf_saveField_r
 - utils_hdf.c, [42](#)
- hdf_saveFieldA
 - utils_hdf.c, [42](#)
- hdf_saveFieldB
 - utils_hdf.c, [42](#)
- hdf_saveFieldPhi
 - utils_hdf.c, [43](#)
- hdf_saveFields
 - utils_hdf.c, [43](#)
- hdf_saveKSpec
 - utils_hdf.c, [43](#)
- hdf_saveMSpec
 - utils_hdf.c, [43](#)
- init.c, [24](#)
 - fill_rand, [25](#)
 - fill_randM0, [25](#)
 - fill_randSingleKM, [26](#)
 - init_conditions, [26](#)
 - init_energySpec, [26](#)
 - init_initEnums, [27](#)
 - init_printParameters, [27](#)
 - init_start, [27](#)
 - RANK_IO, [25](#)
- init_conditions
 - init.c, [26](#)
- init_energySpec
 - init.c, [26](#)
- init_global_size
 - parameters_io.c, [28](#)
- init_initEnums
 - init.c, [27](#)
- init_printParameters
 - init.c, [27](#)
- init_start
 - init.c, [27](#)
- mpi_createTopology
 - utils_mpi.c, [45](#)
- mpi_exchangeMBoundaries
 - utils_mpi.c, [45](#)
- mpi_exchangeMBoundaries_r
 - utils_mpi.c, [45](#)
- mpi_findHermiteNeighbours
 - utils_mpi.c, [45](#)
- mpi_generateTopology
 - utils_mpi.c, [45](#)
- mpi_getLocalArrayOffsets
 - utils_mpi.c, [45](#)
- mpi_getLocalArraySize
 - utils_mpi.c, [45](#)
- mpi_init
 - utils_mpi.c, [46](#)
- mpi_initMExchange
 - utils_mpi.c, [46](#)
- mpi_kill
 - utils_mpi.c, [46](#)
- mpi_sendVector
 - utils_mpi.c, [46](#)
- mpi_splitInCols
 - utils_mpi.c, [46](#)
- mpi_splitInRows
 - utils_mpi.c, [46](#)
- multiply_ar_c
 - array.c, [8](#)
- multiply_ar_r
 - array.c, [8](#)
- parameters_io.c, [27](#)
 - init_global_size, [28](#)
 - read_parameters, [28](#)
 - read_parametersFromFile, [28](#)
- RANK_IO
 - init.c, [25](#)
- read_parameters
 - parameters_io.c, [28](#)
- read_parametersFromFile
 - parameters_io.c, [28](#)
- solver.c, [29](#)
 - solver_init, [29](#)
 - solver_makeStep, [29](#)
- solver_init
 - solver.c, [29](#)
- solver_makeStep
 - solver.c, [29](#)
- space_config.c, [30](#)
 - free_wavespace, [31](#)
 - space_generateMSpace, [31](#)
 - space_generateWaveSpace, [31](#)
- space_generateMSpace
 - space_config.c, [31](#)
- space_generateWaveSpace
 - space_config.c, [31](#)
- utils_fftw.c, [31](#)
 - cosinus, [33](#)
 - dealiasing23, [33](#)
 - fftw_c2r, [34](#)
 - fftw_c2r_chi, [34](#)
 - fftw_c2r_field, [34](#)
 - fftw_copy_buffer_c, [34](#)
 - fftw_copy_buffer_r, [34](#)
 - fftw_copyChiBuf_c, [35](#)
 - fftw_copyChiBuf_r, [35](#)
 - fftw_copyFieldBuf_c, [35](#)

- fftw_copyFieldBuf_r, [36](#)
- fftw_kill, [36](#)
- fftw_normalise_chi_r, [36](#)
- fftw_normalise_data, [36](#)
- fftw_normalise_data_r, [37](#)
- fftw_normalise_field_r, [37](#)
- fftw_r2c, [37](#)
- fftw_r2c_chi, [37](#)
- fftw_r2c_field, [37](#)
- fftw_test_fill, [38](#)
- utils_hdf.c, [38](#)
 - hdf_create_file_c, [40](#)
 - hdf_create_file_r, [40](#)
 - hdf_createCheckpoint, [40](#)
 - hdf_createChiFile_c, [40](#)
 - hdf_createChiFile_r, [40](#)
 - hdf_createFieldFile, [40](#)
 - hdf_createFiles, [41](#)
 - hdf_createParamFile, [41](#)
 - hdf_createSaveDirs, [41](#)
 - hdf_dumpCheckpoint, [41](#)
 - hdf_init, [41](#)
 - hdf_initCheckpoints, [41](#)
 - hdf_initChi, [41](#)
 - hdf_initField, [42](#)
 - hdf_readData, [42](#)
 - hdf_saveData, [42](#)
 - hdf_saveEnergy, [42](#)
 - hdf_saveField_r, [42](#)
 - hdf_saveFieldA, [42](#)
 - hdf_saveFieldB, [42](#)
 - hdf_saveFieldPhi, [43](#)
 - hdf_saveFields, [43](#)
 - hdf_saveKSpec, [43](#)
 - hdf_saveMSpec, [43](#)
- utils_mpi.c, [43](#)
 - mpi_createTopology, [45](#)
 - mpi_exchangeMBoundaries, [45](#)
 - mpi_exchangeMBoundaries_r, [45](#)
 - mpi_findHermiteNeighbours, [45](#)
 - mpi_generateTopology, [45](#)
 - mpi_getLocalArrayOffsets, [45](#)
 - mpi_getLocalArraySize, [45](#)
 - mpi_init, [46](#)
 - mpi_initMExchange, [46](#)
 - mpi_kill, [46](#)
 - mpi_sendVector, [46](#)
 - mpi_splitInCols, [46](#)
 - mpi_splitInRows, [46](#)
- var_getJ0
 - variables.c, [47](#)
- var_getJ1
 - variables.c, [47](#)
- variables.c, [47](#)
 - var_getJ0, [47](#)
 - var_getJ1, [47](#)