GLS user's reference

平成 19 年 8 月 21 日

目 次

第 1 章　はじめに **3**
1.1 概要 . . . 3
1.2 構成 . . . 3
1.3 経緯 . . . 3
1.4 免責事項とその他 . . . 3

第 2 章　libgls の関数（コントロールウィンドウ関連） **5**
2.1 数値の操作 . . . 5
2.1.1 int の操作（gls_value_add_int4 関数） . . . 5
2.1.2 float の操作（gls_value_add_real4 関数） . . . 5
2.1.3 double の操作（gls_value_add_real8 関数） . . . 5
2.2 チェックボックスとボタンの作成 . . . 5
2.2.1 チェックボックスの作成（gls_check_add 関数） . . . 5
2.2.2 ボタンの作成（gls_button_add 関数） . . . 6
2.2.3 チェックボックスやボタンの状態の取得（gls_check_get_status 関数） . . . 6
2.3 スクロールバー関連のウィジェット . . . 6
2.3.1 スクロールバーを作成（gls_scroll_add 関数） . . . 6
2.3.2 gls_scroll_update 関数 . . . 6
2.4 区切りの作成（gls_separator_add 関数） . . . 6

第 3 章　libstat2 の計算クラス一覧とグローバル変数 **7**
3.1 scalar1D . . . 7
3.1.1 メンバ変数 . . . 7
3.1.2 メンバ関数 . . . 7
3.1.3 宣言とデフォルトコンストラクタ . . . 8
3.2 scalar2D . . . 9
3.2.1 メンバ変数 . . . 9
3.2.2 メンバ関数 . . . 9
3.2.3 宣言とデフォルトコンストラクタ . . . 10
3.3 vector2D . . . 10
3.3.1 メンバ変数 . . . 10
3.3.2 メンバ関数 . . . 10
3.3.3 宣言とデフォルトコンストラクタ . . . 11
3.4 director2D . . . 12
3.4.1 メンバ変数 . . . 12
3.4.2 メンバ関数 . . . 12
3.4.3 宣言とデフォルトコンストラクタ . . . 13
3.5 scalar3D . . . 13
3.5.1 メンバ変数 . . . 13
3.5.2 メンバ関数 . . . 13
3.5.3 宣言とデフォルトコンストラクタ . . . 14

3.6 vector3D 14
3.6.1 メンバ変数 14
3.6.2 メンバ関数 14
3.6.3 宣言とデフォルトコンストラクタ 15
3.7 グローバル変数 16

第1章　はじめに

1.1　概要

GLS は数値計算を実行しながらその様子を X-Window system 上で視覚的にチェックすることのできるライブラリである。可視化部分は OpenGL で、コントロールウィンドウなどは GTK+2.0 で作成されている。このライブラリ群では 1 次元から 3 次元までの数値計算とその可視化をサポートしている。

1.2　構成

GLS は大きく分けて libgls と libstat2 の二つから構成されている。

libgls 可視化ウィンドウ作成やコントロールウィンドウを作成し、可視化ウィンドウへのマウスクリックやコントロールウィンドウの数値入力を行うウィジェットを提供

libstat2 libstat2 可視化ウィンドウにどのような図形を描くかという具体的なアウトプットを担ったり、データの保存、画像の保存を行ったり、変数の初期化などを行うクラスを提供

OpenGL を明示的に利用しているのは libstat2 の方だけで、libgls はコントロールウィンドウを提供するのが主な仕事と言ってよい。故に、OpenGL による可視化をせずに、コントロールウィンドウだけを開くというシミュレーションも可能である。

1.3　経緯

- 2003.4：小本氏が OpenGL で 3D 熱対流の可視化コードを書く。
- 2003.8：南がスカラー場しか可視化できない XRS に不満を持って、可視化ライブラリを自作しようとする。
- 2003.9：スカラー場・ベクトル場・液晶場の可視化が完成。相互の連携もできるようにする。小本氏の 3D 可視化と統合して手を加える。
- 2004.5：gtk+2 を使って書き直し。
- 2004.11：C++を使って書き直し。
- 2005.2：ライブラリ化して配布可能な形にする。
- 2007.5：OpenGL Simulator を略して GLS と名付ける。
- 2007.8：1 次元可視化を作成。

1.4　免責事項とその他

- このライブラリは BSD ライセンスに従います。
- このライブラリを使用は自己責任でお願いします。

- ライブラリに関するご質問にはお答えできない場合もあります。
- 「こういう機能が欲しい」などのご要望や、バグ修正のご要望には一切お応えできません。ご自身でソースコードを解読して改造してください。
- 3 次元等値面のリアルタイムシミュレーションはシステムサイズが大きいとき、可視化が律速になる場合があります。
- より高度な 3 次元可視化を行う場合は、OpenDX や AVS をおすすめします。

第2章　libglsの関数（コントロールウィンドウ関連）

2.1　数値の操作

2.1.1　int の操作（gls_value_add_int4 関数）

整数の値を変更するウィンドウを作る関数。

```
GtkWidget* gls_value_add_int4(const gchar* name, gint32* value);
```

const gchar* name には入力枠の横に表示する文字列を与える。gint32* value には値を変更したい変数のポインタを渡す。

2.1.2　float の操作（gls_value_add_real4 関数）

浮動小数点実数の値を変更するウィンドウを作る関数。

```
GtkWidget* gls_value_add_real4(const gchar* name, float* value);
```

const gchar* name には入力枠の横に表示する文字列を与える。float* value には値を変更したい変数のポインタを渡す。

2.1.3　double の操作（gls_value_add_real8 関数）

倍精度実数の値を変更するウィンドウを作る関数。

```
GtkWidget* gls_value_add_real8(const gchar* name, double* value);
```

const gchar* name には入力枠の横に表示する文字列を与える。double* value には値を変更したい変数のポインタを渡す。

2.2　チェックボックスとボタンの作成

2.2.1　チェックボックスの作成（gls_check_add 関数）

チェックボックスを作る関数。

```
GtkWidget* gls_check_add(const gchar* name, GLS_EVENT_PUSH callback, gpointer p);
```

第一引数 const gchar* name はコントロールウィンドウに表示する文字列。第二引数 GLS_EVENT_PUSH callback はチェックを入れたとき呼び出す関数 (コールバック関数) の名前をいれる。第三引数はコールバック関数に渡す値を入れる。

コールバック関数は以下のような形をしている。

```
void callback(gpointer p);
```

gls_check_add 関数の最後の引数が gpointer p に渡される。

2.2.2 ボタンの作成（gls_button_add 関数）

ボタンを作る関数。

```
GtkWidget* gls_button_add(const gchar* name, GLS_EVENT_PUSH callback, gpointer p);
```

第一引数 const gchar* name はコントロールウィンドウに表示する文字列。第二引数 GLS_EVENT_PUSH callback はチェックを入れたとき呼び出す関数 (コールバック関数) の名前をいれる。第三引数はコールバック関数に渡す値を入れる。

コールバック関数は以下のような形をしている。

```
void callback(gpointer p);
```

gls_button_add 関数の最後の引数が gpointer p に渡される。

2.2.3 チェックボックスやボタンの状態の取得（gls_check_get_status 関数）

```
gboolean gls_check_get_status(GtkWidget* w);
```

gls_check_add や gls_button_add の返り値を渡せばボタンやチェックボックスが ON になっているときに TRUE、OFF になっているとき FALSE を返す。

2.3 スクロールバー関連のウィジェット

2.3.1 スクロールバーを作成（gls_scroll_add 関数）

水平方向スクロールを作成する関数。

```
GtkWidget* gls_scroll_add(int min, int max, int *val);
```

第一引数にスクロールバーの最小値を、第二引数に最大値を、第三引数に現在の値のポインタを与えればよい。

2.3.2 gls_scroll_update 関数

スクロールの値を指定されたステップだけ増やす関数。

```
void gls_scroll_update(GtkWidget *w, int *index_scroll, int min, int max, int step);
```

第一引数にスクロールのウィジェットを、第二引数にスクロールの値へのポインタを、第三引数にスクロールの最小値、第四引数に最大値をいれる。最後の引数は、一度のアップデートでどれだけスクロールの値を動かすかを決める。

2.4 区切りの作成（gls_separator_add 関数）

セパレーターをコントロールウィンドウに追加する関数。ただ呼び出すだけ。

```
void gls_separator_add();
```

第3章　libstat2の計算クラス一覧とグローバル変数

3.1　scalar1D

3.1.1　メンバ変数

- `double *s`：可視化用変数。宣言と同時にデフォルトコンストラクタで動的にメモリが確保される。
- `double up, low`：可視化をするときの上下の閾値。クラスの宣言時に指定する。
- `int NNX`：システムサイズ（NNX）を表す。NNX はクラスの宣言時に指定する。
- `void (*draw_add)(void)`：お絵描き関数へのポインタ。

3.1.2　メンバ関数

view 関数

```
void view (const gchar * title, int xsize, int ysize);
```

可視化ウィンドウを作成する関数。`const gchar *title` はウィンドウの上端に現れる文字列。`int xsize` と `int ysize` はウィンドウの大きさ（ピクセル）。

tga 関数

```
void tga (const char *dir, const char *name, int sizex, int sizey, int index);
```

tga 形式の画像を出力する関数。`const char *dir` には出力するディレクトリを入れる。このディレクトリが無いときには画像が出力されないので注意。`const char *name` にはファイルに付ける名前を入れる。なお、画像ファイルには拡張子 ".tga" が自動で付加される。`int sizex` と `int sizey` は画像の縦横の大きさ（ピクセル）。`int index` に適当な正の整数を入れると、ファイル名に 4 桁の番号が付く。ここに NOINDEX と書くと番号が付かない。最後の引数は省略することができて、その時は番号のないファイル名が用いられる（つまり NOINDEX と同じ）。

eps 関数

```
void eps (const char *dir, const char *name, int sizex, int sizey, int index);
```

eps 形式の画像を出力する関数。`const char *dir` には出力するディレクトリを入れる。このディレクトリが無いときには画像が出力されないので注意。`const char *name` にはファイルに付ける名前を入れる。なお、画像ファイルには拡張子 ".eps" が自動で付加される。`int sizex` と `int sizey` は画像の縦横の大きさ（ピクセル）。`int index` に適当な正の整数を入れると、ファイル名に 4 桁の番号が付く。ここに NOINDEX と書くと番号が付かない。最後の引数は省略することができて、その時は番号のないファイル名が用いられる（つまり NOINDEX と同じ）。

read 関数

```
void read (const char *dir, const char *name, int index);
```

可視化用のポインタ double *s に確保された領域のデータを丸ごとバイナリ形式で保存する関数。const char *dir には出力するディレクトリを入れる。このディレクトリが無いときにはデータが出力されないので注意。const char *name にはファイルに付ける名前を入れる。int index に適当な正の整数を入れると、ファイル名に 4 桁の番号が付く。ここに NOINDEX と書くと番号が付かない。最後の引数は省略することができて、その時は番号のないファイル名が用いられる（つまり NOINDEX と同じ）。

write 関数

```
void write (const char *dir, const char *name, int index);
```

read 関数で出力したデータを読み込む関数。const char *dir には出力するディレクトリを入れる。このディレクトリが無いときにはデータが出力されないので注意。const char *name にはファイルに付ける名前を入れる。int index に適当な正の整数を入れると、ファイル名に 4 桁の番号が付く。ここに NOINDEX と書くと番号が付かない。最後の引数は省略することができて、その時は番号のないファイル名が用いられる（つまり NOINDEX と同じ）。

draw_point 関数

```
void draw_point(double x[2], double size, double col[3])
```

横軸 x[0]、縦軸 x[1] の位置に size ピクセルの点を打つ関数。色は col[3] で指定する。col[0] は赤を、col[1] は青を、col[2] は緑を表す。

draw_line 関数

```
void draw_line(double x1[2], double x2[2], double col[3])
```

(x1[0], x1[1]) の位置から (x2[0], x2[1]) へ直線を引く関数。色は col[3] で指定する。col[0] は赤を、col[1] は青を、col[2] は緑を表す。

3.1.3 宣言とデフォルトコンストラクタ

変数 psi を宣言するときには以下のようにすればよい。

```
scalar1D psi(NX, 1, -1);
```

第一引数はシステムサイズを表す。第二引数と第三引数ではメンバ変数 double up, low に値を代入し、可視化をする際のグラフの上限と下限を設定している。ただし、後ろ二つの引数は省略できる。その際は、up=1、low=0 が自動的に設定される。

例えば、上記のように宣言したとして、可視化変数に具体的な値を入れて図示したいときは以下の例のように psi.s[i] を一つの配列とみなして計算すればよい。

```
for (i = 1; i <= NX; i++) {
     phi.s[i] = i;
}
```

可視化ウィンドウの上端と下端の値を変えたければ psi.up や psi.low を変えればよい。

3.2 scalar2D

3.2.1 メンバ変数

- double **s：可視化用変数。宣言と同時にデフォルトコンストラクタで動的にメモリが確保される。
- double SCALE：可視化をするときの濃度を表す。
- int NNX, NNY：システムサイズ（NNX*NNY）を表す。NNX と NNY はクラスの宣言時に指定する。
- void (*draw_add)(void)：お絵描き関数へのポインタ。

3.2.2 メンバ関数

view 関数

scalar1D と同じ。

tga 関数

scalar1D と同じ。

eps 関数

```
void eps (const char *dir, const char *name, int index);
```

eps 形式の画像を出力する関数。const char *dir には出力するディレクトリを入れる。このディレクトリが無いときには画像が出力されないので注意。const char *name にはファイルに付ける名前を入れる。なお、画像ファイルには拡張子 ".eps" が自動で付加される。int index に適当な正の整数を入れると、ファイル名に 4 桁の番号が付く。ここに NOINDEX と書くと番号が付かない。最後の引数は省略することができて、その時は番号のないファイル名が用いられる（つまり NOINDEX と同じ）。

read 関数

scalar1D と同じ。

write 関数

scalar1D と同じ。

draw_point 関数

scalar1D と同じ。

draw_line 関数

scalar1D と同じ。

3.2.3 宣言とデフォルトコンストラクタ

変数 psi を宣言するときには以下のようにすればよい。

```
scalar2D psi(NX, NY, 1);
```

最初二つの引数はシステムサイズを表す。最後の数値は可視化した時の濃淡の強度 SCALE の初期値をいれる。この値が大きい程濃淡が強烈に出る。また、最後の引数は省略できて、省略した場合はデフォルト値 SCALE=1 が設定される。

実際に計算を行う場合は計算領域 double **s に値をいれたりすればよい。

例えば、上記のように宣言したとして、可視化変数に具体的な値を入れて図示したいときは以下の例のようにpsi.s[i][j] を一つの配列とみなして計算すればよい。

```
for (i = 1; i <= NX; i++) {
  for (j = 1; j <= NY; j++) {
    phi.s[i][j] = 0.01 * (i + j);
  }
}
```

また、濃淡を変えたければ psi.SCALE の値を変えればよい。

3.3 vector2D

3.3.1 メンバ変数

- double ***v：可視化用変数。宣言と同時にデフォルトコンストラクタで動的にメモリが確保される。
- double SCALE：ベクトルの長さを変える変数。SCALE=1 のとき、ベクトルの長さが 1 ならば計算格子一つ分の長さになる。
- int NNX, NNY：システムサイズ（NNX*NNY）を表す。NNX と NNY はクラスの宣言時に指定する。
- void (*draw_add)(void)：お絵描き関数へのポインタ。

3.3.2 メンバ関数

view 関数

scalar1D と同じ。

tga 関数

scalar1D と同じ。

eps 関数

```
void eps (const char *dir, const char *name, int thind_down, int index);
```

eps 形式の画像を出力する関数。const char *dir には出力するディレクトリを入れる。このディレクトリが無いときには画像が出力されないので注意。const char *name にはファイルに付ける名前を入れる。なお、画像ファイルには拡張子 ".eps" が自動で付加される。int thin_down はベクトルの間引き率を表す。これが 2 の時は、2 格子に一つだけが描画される。int index に適当な正の整数を入れると、ファイル名に 4 桁の番号が付く。ここに NOINDEX と書くと番号が付かない。最後の引数は省略することができて、その時は番号のないファイル名が用いられる（つまり NOINDEX と同じ）。

read 関数

scalar1D と同じ。

write 関数

scalar1D と同じ。

draw_point 関数

scalar1D と同じ。

draw_line 関数

scalar1D と同じ。

3.3.3 宣言とデフォルトコンストラクタ

2 次元のベクトル場 vec を計算したい場合には以下のように宣言すればよい。

```
vector2D vec(NX, NY, 1);
```

最初二つの引数はシステムサイズを表す。最後の数値は可視化した時の矢印の長さの拡大率 SCALE の初期値をいれる。また、最後の引数は省略できて、省略した場合はデフォルト値 SCALE=1 が設定される。

実際に計算を行う場合は計算領域 double ***v に値をいれたりすればよい。つまり上記のように宣言すれば、後は vec.v[0][i][j] と vec.v[1][i][j] を一つの配列とみなして計算すればよい。ただし vec.v[0] は x 成分、vec.v[1] は y 成分を表す。実際に使うときは以下のようにする。

```
for (i = 1; i <= NX; i++) {
  for (j = 1; j <= NY; j++) {
    vec.v[0][i][j] -= DT * chemx[i][j];
    vec.v[1][i][j] -= DT * chemy[i][j];
  }
}
```

3.4 director2D

3.4.1 メンバ変数

- double ***v：ディレクターの向きを与える可視化用変数。宣言と同時にデフォルトコンストラクタで動的にメモリが確保される。
- double **c：ディレクターに色をつける変数。
- double SCALE：ディレクターの長さを変える変数。SCALE=1 のとき、ディレクターの長さが 1 ならば計算格子一つ分の長さになる。
- double CSCALE：ディレクターに付ける色の濃さを設定する変数。
- int NNX, NNY：システムサイズ（NNX*NNY）を表す。NNX と NNY はクラスの宣言時に指定する。
- void (*draw_add)(void)：お絵描き関数へのポインタ。

3.4.2 メンバ関数

view 関数

scalar1D と同じ。

tga 関数

scalar1D と同じ。

eps 関数

vector2D と同じ。

read 関数

scalar1D と同じ。

write 関数

scalar1D と同じ。

draw_point 関数

scalar1D と同じ。

draw_line 関数

scalar1D と同じ。

3.4.3 宣言とデフォルトコンストラクタ

2 次元のディレクター場 n を計算したい場合には以下のように宣言すればよい。

```
director2D n(NX, NY, 1, 1);
```

最初二つの引数はシステムサイズを表す。三つ目の数値は可視化した時のディレクターの長さの拡大率 SCALE の初期値をいれる。最後の数値は色をつけるときの濃度を表す。基本的な使い方は vector2D とそんなにかわらない。また、後ろ二つの引数は省略できて、省略した場合はデフォルト値 SCALE=1 および CSCALE=1 が設定される。

3.5 scalar3D

3.5.1 メンバ変数

- double ***s：可視化用変数。宣言と同時にデフォルトコンストラクタで動的にメモリが確保される。
- double SCALE：表面表示モードのときは色づけの濃度を表す。等値面表示モードでかつマニュアル設定（後述）のときは等値面の値を設定する。自動設定（後述）のときは適当。
- GtkWidget *drawmode：チェックボックスを作成する関数 gls_check_add で表面表示モードと等値面表示モードを切り替えるための変数。
- SCALAR3D_DRAW_MODE mode：等値面表示のときの描画モードを設定する変数。
- int NNX, NNY, NNZ：システムサイズ（NNX*NNY*NNZ）を表す。NNX と NNY と NNZ はクラスの宣言時に指定する。
- void (*draw_add)(void)：お絵描き関数へのポインタ。

3.5.2 メンバ関数

view 関数

scalar1D と同じ。

tga 関数

scalar1D と同じ。

eps 関数

scalar3D クラスでは用意されていない（厳密には作ったけど使い物にならない）。

read 関数

scalar1D と同じ。

write 関数

scalar1D と同じ。

3.5.3 宣言とデフォルトコンストラクタ

3 次元のスカラー場 psi を計算したい場合には以下のように宣言すればよい。

```
scalar3D psi(NX, NY, NZ, TWO_AUTO, 1);
```

最初三つの引数はシステムサイズを表す。四つ目の引数は等値面表示のときの閾値の設定方法を決める。五つ目の引数はメンバ変数 double SCALE を設定する。これは表面表示モードのときは scalar2D と同じような形で濃度を設定する変数と考えてよい。等値面表示モードのときは閾値の設定方法によって異なる意味を持つ。

scalar3D の等高面表示では大きく分けて二つの閾値設定方法が用意されている。一つは与えられたスカラー場の二乗平均を計算し、double SCALE で割ったものを閾値とする等高面を描く方法（自動設定）、もう一つは手で閾値（double SCALE）を入力する方法（手動設定）である。

また、この二つの閾値の設定方法に加えて、さらに等高面の枚数を一枚か二枚かを設定できる。一枚の場合は上で設定した閾値の面を赤く（もしくは青く）描く。二枚の場合は閾値を t とすると、値 t の面を赤く、値 $-t$ の面を青く描く。これはメンバ変数の SCALAR3D_DRAW_MODE mode で指定することができる。この変数として代入できるものを以下に整理しておく。

閾値設定方法	SCALAR3D_DRAW_MODE mode の値
自動設定/赤一枚	RED_AUTO
自動設定/青一枚	BLUE_AUTO
自動設定/二枚	TWO_AUTO
手動設定/赤一枚	RED_MANUAL
手動設定/青一枚	BLUE_MANUAL
手動設定/二枚	TWO_MANUAL

また、後ろ二つの引数は省略できて、省略した場合はデフォルト値 mode=TWO_AUTO および SCALE=1 が設定される。

3.6 vector3D

3.6.1 メンバ変数

- double ****v：可視化用変数。宣言と同時にデフォルトコンストラクタで動的にメモリが確保される。
- double SCALE：意味の無い変数。
- int NNX, NNY, NNZ：システムサイズ（NNX*NNY*NNZ）を表す。NNX と NNY と NNZ はクラスの宣言時に指定する。

3.6.2 メンバ関数

3 次元ベクトルの汎用的な可視化方法はないので、このクラスには可視化および画像出力のメンバ関数は定義されていない。ただ、データ出力などの文法を他の scalar3D などに合わせて統一的に書くために便宜上定義してある。

read 関数

scalar1D と同じ。

write 関数

scalar1D と同じ。

draw_point 関数

```
void draw_point(double x[3], double size, double col[3])
```

(x[0], x[1], x[2]) の位置に size ピクセルの点を打つ関数。色は col[3] で指定する。col[0] は赤を、col[1] は青を、col[2] は緑を表す。

draw_line 関数

```
void draw_line(double x1[3], double x2[3], double col[3])
```

(x1[0], x1[1], x1[2]) の位置から (x2[0], x2[1], x2[2]) へ直線を引く関数。色は col[3] で指定する。col[0] は赤を、col[1] は青を、col[2] は緑を表す。

3.6.3 宣言とデフォルトコンストラクタ

3 次元のベクトル場 vec を計算したい場合には以下のように宣言すればよい。

```
vector3D vec(NX, NY, NZ, 1);
```

最初三つの引数はシステムサイズを表す。可視化の際の矢印の長さの倍率を表す変数 double SCALE は、3 次元でのベクトル場可視化が困難であることから定義しているが実際には使わない。

実際に計算を行う場合は計算領域 double ****v に値をいれたりすればよい。つまり上記のように宣言すれば、後は vec.v[0][i][j][k] と vec.v[1][i][j][k] と vec.v[2][i][j][k] を一つの配列とみなして計算すればよい。vec.v[0] は x 成分、vec.v[1] は y 成分、vec.v[2] は z 成分を表す。以下に例を示す。

```
for (i = 1; i <= NX; i++) {
  for (j = 1; j <= NY; j++) {
    for (k = 1; k <= NZ; k++) {
      vec.v[0][i][j][k] -= DT * chemx[i][j][k];
      vec.v[1][i][j][k] -= DT * chemy[i][j][k];
      vec.v[2][i][j][k] -= DT * chemz[i][j][k];
  }
}
```

3.7 グローバル変数

- GtkWidget *check_axis：scalar3D で可視化をするときに、座標軸の方向を描画するかどうかを決める変数。図示するときは xyz がそれぞれ赤青緑で表される。gls_check_add などのチェックボックスで操作する。
- GtkWidget *check_vector:vector2D で可視化する領域を scalar2D 上に表示するかどうかを決める変数。gls_check_add などのチェックボックスで操作する。
- GtkWidget *check_vector：vector2D で可視化するときの表示範囲を設定する変数。デフォルトでは 20 格子に設定されている。