電子の相互作用シミュレーション

01ca0125 鈴木 藍
2002 年 9 月 27 日

目 次

概 要

しめのレポートである。行った作業や、システムの概要、 関数と変数の表などをまとめた。

レポートの目的

- 基礎的な数学の理解確認
- 基本的なプログラミングの練習
- ソフトウェア・アーキテクチャを考える

1 はじめに

まず、レポートの提出期限を守れずに、ここまで遅い提出になってしまった事を謹んでここで反省したいと思います。

2 全体の仕様

課題として提示されたものを最低限実装した。

3 オプション一覧

今回は、オプションの処理に getopt_long を使ったので、-option ではなく –option とした。

- $--if$ オプション, $-i < filename >$ 初期状態入力ファイル
- $--of$ オプション, $-o < filename >$ 最終状態ファイル
- $--bn$ オプション, $-b < number >$ 乱数で電子を生成
- $--fn$ オプション, $-f < number >$ 指定のフレームを出力する

なお、オプションが指定されなかった時のデフォルトとして 2 つの電子が乱数を使ったパラメータを状態として持っている。

3.1 ファイルのフォーマット

出力されるファイルのフォーマットは以下のとおりである。

```
(nx 20) (ny 20) (t 100) (dt 0.100000) (bn 1)
(e
((x 18.129819) (y 48.093291) (vx 0.000000) (vy 0.000000) (q -1.000000) (m 5))
((x 23.576020) (y 29.195874) (vx -0.029880) (vy 0.142618) (q 1.000000) (m 6))
)
```

一番はじまりの行は、場の状態を表す。以下の (e で始まる行は、1 行に付き、一つの電子の電子の状態を格納している。

4 開発工程

今回は、時間をかける事が出来なくて大きな成果を得られなかったが、今までとは違う方法をとることで、よい方向に向かった事を書いておく。

4.1 バージョン管理

常に、動くものを保存しておく。今回は ディレクトリに バージョン番号を付け、管理した。もし、新しく改良を行って失敗してしまった場合にも、すぐに動くものにもどす事が出来る上に、差分を見て 新しく問題を発見する事も出来た。また、0.95 〜 1.0 になるまでを目標に作る事で作業にめりはりが付いたと思う。この間に $6 + \alpha$ 分の プログラムを作成した。

4.2 データの管理

これまで、定数の扱いに悩んできたが enum を使って 一貫した定数の管理が出来るようになった。また、配列の先頭から順に 美しく意味付けをすることが出来て良かったと思う。

5 データ構造

大きなデータ構造は使用しなかった。電子の構造体と場の構造体は、一番始めのバージョンから作られていたため、これを継続する事にした。

5.1 電子を表す構造体

```
/*----------- an Electron ----------------------*/
    struct electron {
    double   q;
    int      m;
    double   x, y;
    double   vx, vy;
    struct electron *next;
};
/*----------------------------------------------*/
```

5.2　スカラー場を表す構造体

要素は、double 型の 2 次元配列のみである。

```
/*------------ scale field ----------------------*/
    struct sfield {
    double  **potential;
};
/*-----------------------------------------------*/
```

5.3　ベクトル場を表す構造体

一ヶ所に付き、二次元の性質を持つ構造体をもう一つ定義し、ベクトル場の一点の要素とした。

```
/*-------------- 2D being ----------------------*/
    struct twoDbeing {
    double  x, y;
};
/*-----------------------------------------------*/
/*------------- vector field --------------------*/
    struct vfield {
    struct  twoDbeing **pointOf;
};
/*-----------------------------------------------*/
```

6　おもな変数一覧

主な変数はグローバル変数として定義している。

型	変数名	説明
int	nx, ny	画面の分割数
double	dx, dy	分割された一つのセルの大きさ
int	t	実行する時間
double	dt	ある時間から時間までの微小な差分
int	bn	電子の数
int	fn	出力するフレームの数
anElectorn	electorn	電子
ScaleField	scalefiled	スカラー場
VectorField	vectorfiled	ベクトル場

6.1 関数一覧

型	関数名	引数	説明
ScaleField	newScaleField	なし	スカラー場のメモリを確保する。
ScaleField	makeScaleField	ScaleField *, anElectron *	電子からスカラー場を形成する。
ScaleField	clearScaleField	ScaleField *	スカラー場を 0.0 で初期化する。
VectorField	newVectorField	なし	ベクトル場のメモリを確保する。
VectorField	makeVectorField	VectorField *, ScaleField *	ベクトル場を形成する。
VectorField	clearVectorField	Vectorield *	ベクトル場を 0.0 で初期化する。
void	initWorld	なし	gnuplot 用の準備をする。
void	outputVectorDataFormat	VectorField *	ベクトル場用の出力
void	outputScaleDataFormat	ScaleField *	スカラー場用の出力
void	inspectIt	anElectron *	電子の状態を出力する
anElectorn	initElectrons	なし	電子を生成する
void	theWorld	なし	main 関数の次に中心となる関数
void	procOptions	int, char **	オプションを処理する
char	skipto	char *, char	トークンのポインタを返す
void	outputFile	anElectron *	ファイルに最終状態を出力する
void	loadFormFile	anElectron *	ファイルから状態を入力する
void	get_field_elem	char *	場の要素を取り出す
void	get_field_val	なし	要素から 値を取り出す
void	get_elect_elem	FILE *	電子の要素を取り出す
void	get_elect_val	anElectrons *	要素から 値を取り出す

7 実行例

紙面でアニメーションがしづらいので、今回は割愛させていただきます。

7.1 動いている事の実証

テストを ベクトル場のアニメーション、スカラー場のアニメーション、電子のみのアニメーション全てをそのつど行う事で、不自然な動きがないかを確認してきた。中でもベクトル場は、計算がずれるとすぐにおかしな画面になるので これをいつも確認してきたことは良かったと思う。

まとめ・結論

設計という事がまた、良くできていなかったが いままで読んで来た本などが生かせる場があった。いつも、何かの形でできあがっているプログラムをこまめに保存しておくことは良かったと思う。

感想

後悔しきりでありますが、全てが言い訳に聞こえますので止めます。

参考文献

[1] プログラミング言語 C
カーニハン , リッチー 著
2000 年 4 月 15 日 (第 2 版)

[2] Smalltalk イディオム
青木 淳 著
1997 年 2 月 25 日 (第 1 版)

[3] プログラミング作法
カーニハン , パイク
2001 年 6 月 21 日 (第 1 版)