VectorStructOperator.hの使い方
~マニュアル~

九州大学数理学研究院　秋山 正和

はじめに

2D or 3D空間で物理シミュレーションを行う場合，数々のベクトル量を扱う必要がある.

例えば3D空間にて，何らかのモデルをシミュレーションをする必要があったとしよう.

質点の位置を位置ベクトルを用いて,

$$\boldsymbol{r} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}$$

などと表し，その質点にかかる力(紙面の都合上，上記のような縦ベクトルを下記のように書く)

$$\boldsymbol{f} = (fx, fy, fz)^T$$

を求め，次式のような運動方程式をたてて解くだろう.

$$m\ddot{\boldsymbol{r}} = \boldsymbol{f}$$

これをあるプログラマが以下のように書いたとする．．．

```
~~
int main(void)
{
~いろんな宣言など~
   double u_x,u_y,u_z;
   double u_new_x,u_new_y,u_new_z;
   double f_x,f_y,f_z;
   double v_x,v_y,v_z;
   double v_new_x,v_new_y,v_new_z;

   ~初期化ルーチンなど~
   /*時間ループ*/
   for(;;)
   /*方程式を解く*/
   v_new_x = v_x + dt * f_x;
   v_new_y = v_y + dt * f_y;
   v_new_z = v_z + dt * f_z;
   u_new_x = u_x + dt * v_new_x;
   u_new_y = u_x + dt * v_new_y;
   u_new_z = u_z + dt * v_new_z;
   /*Fを計算する*/
   f_x = ...
   f_y = ...
   f_z = ...
~計算結果の表示など~
return 0;
}
```

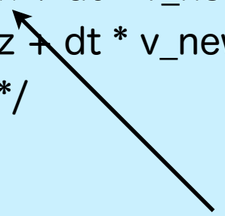
このようなプログラムは頭が悪い人の書き方である．

なぜならば，解くべき方程式はベクトルで与えられているのに，それをわざわざスカラー変数にし，連立のスカラー変数の問題として解こうとしているからである．

また，プログラムの中には「**_x,**_y,**_z」があるが，このような書き方はミスを誘発する．なぜなら，仮に「_y」を「_x」と書いてもコンパイラはエラーを返さないため，デバックが困難となるからである．

実際，このプログラムはバグがある．それを「一瞬」で見つけられるだろうか？

```
~~
int main(void)
{
~いろんな宣言など~
   double u_x,u_y,u_z;
   double u_new_x,u_new_y,u_new_z;
   double f_x,f_y,f_z;
   double v_x,v_y,v_z;
   double v_new_x,v_new_y,v_new_z;

   ~初期化ルーチンなど~
   /*時間ループ*/
   for(;;)
   /*方程式を解く*/
   v_new_x = v_x + dt * f_x;
   v_new_y = v_y + dt * f_y;
   v_new_z = v_z + dt * f_z;
   u_new_x = u_x + dt * v_new_x;
   u_new_y = u_x + dt * v_new_y;
   u_new_z = u_z + dt * v_new_z;
   /*Fを計算する*/
   f_x = ...
   f_y = ...
   f_z = ...
~計算結果の表示など~
return 0;
}
```



バグ！

このようなプログラムは頭が悪い人の書き方である．

なぜならば，解くべき方程式はベクトルで与えられているのに，それをわざわざスカラー変数にし，連立のスカラー変数の問題として解こうとしているからである．

また，プログラムの中には「**_x,**_y,**_z」があるが，このような書き方はミスを誘発する．なぜなら，仮に「_y」を「_x」と書いてもコンパイラはエラーを返さないため，デバックが困難となるからである．

実際，このプログラムはバグがある．それを「一瞬」で見つけられるだろうか？

このせいで，どんなに問題が正しくても，どんなにアルゴリズムが正しくても，どんなに速い計算機でも３日以上時間を無駄に費やすことさえあるのだ．

そこで

VectorStructOperator.h

を紹介する.

これは

１．ベクトルを定義し

２．そのベクトルに対して，演算を定め

３．非常に使いやすくした

ものである.

先の悪いコード(左)は本プログラムを
使えば右のようにすっきりかける.

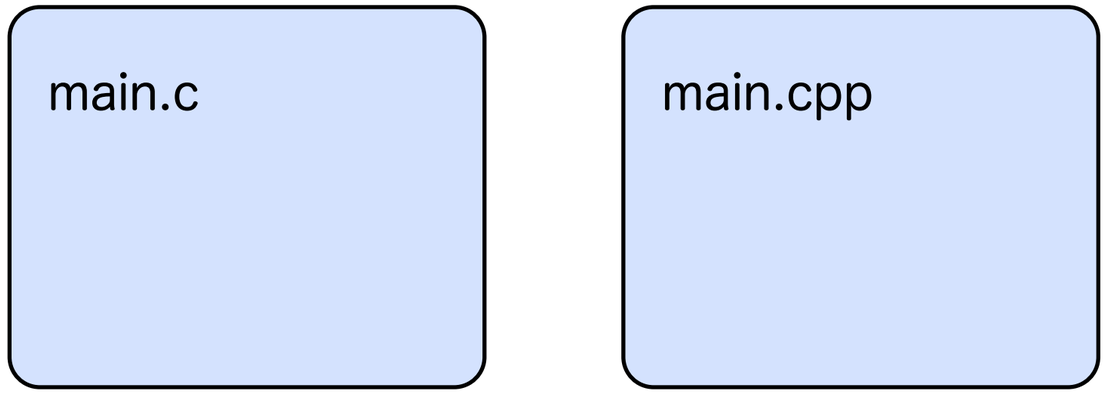
当然，「_x」などという記述はなくなるので，バグは入り込む余地がない．つまり，すっきりかけるだけなく，あなたが意図した正しい動作をするのである.

```
~~
int main(void)
{
~いろんな宣言など~
   double u_x,u_y,u_z;
   double u_new_x,u_new_y,u_new_z;
   double f_x,f_y,f_z;
   double v_x,v_y,v_z;
   double v_new_x,v_new_y,v_new_z;

   ~初期化ルーチンなど~
   /*時間ループ*/
   for(;;)
   /*方程式を解く*/
   v_new_x = v_x + dt * f_x;
   v_new_y = v_y + dt * f_y;
   v_new_z = v_z + dt * f_z;
   u_new_x = u_x + dt * v_new_x;
   u_new_y = u_y + dt * v_new_y;
   u_new_z = u_z + dt * v_new_z;
   /*Fを計算する*/
   f_x = ...
   f_y = ...
   f_z = ...
~計算結果の表示など~
return 0;
}
```

```
#include <VectorStructOperator.h>
~~
int main(void)
{
~いろんな宣言など~
   Vector3D u,u_new,f,v,v_new;

   ~初期化ルーチンなど~
   /*時間ループ*/
   for(;;)
   /*方程式を解く*/
   v_new = v + dt * f;
   u_new = u + dt * v_ new;
   /*Fを計算する*/
   f = ...
~計算結果の表示など~
return 0;
}
```

実際の使い方は以下のようにすればよい．(次元は2Dを3Dに読み替えればどちらでも有効．)

まずcプログラムをc++プログラムにする．



今まで　　これから

Q. CプログラムからC++にするって．．．
A. 恐れなくてよい．あなたのCプログラムは内容を何も変更しなくても，拡張子をcppに変更すればよい．
実際，作者はC++のことは全くわからないが，問題なく動作しているし，まちがった考え方でもない．

次にmain.cppに #include “VectorStructOperator.h”を追加する.

```
#include <stdio.h>
#include <AAA.h>
#include <BBB.h>
#include <math.h>


int main(void){
```

今まで　　main.c

```
#include <stdio.h>
#include <AAA.h>
#include <BBB.h>
#include <math.h>
#include “VectorStructOparator.h”

int main(void){
```

これから　　main.cpp

Q. #include “VectorStructOparator.h”ではなくて #include <VectorStructOparator.h>ではないのか

A. main.cppと同じ階層のフォルダにあるなら，#include “VectorStructOparator.h”とする.
もし~/includeなどのパスの通ったフォルダにおくなら#include <VectorStructOparator.h>のほうが良い．作者は後者をお勧めする.

コンパイラをccからc++にする.

```
$ cc main.c
$./a.out
```

今まで　　main.c

```
$ c++ main.cpp
$./a.out
```

これから　　main.cpp

Q. makeをつかっているのだけれども

A. CCをやめてCXX=c++,LD=c++などとしてみよう．動くはずである.

Q. 絶対にcファイルがあってはだめ？
A. mainのみcppであればよいだけで，サブプログラムはcファイルでもかまわない．ただし，注意点もある（事項）.

Q. よくわからん
A. サンプルをコンパイルできれば，ゴールは近い．がんばれ

Q. 見慣れないエラーがでた.

A. 関数のプロトタイプ宣言を以下の文字列(extern　　“C” {})でくくってみよう．なおる.

```
#include <stdio.h>
#include <AAA.h>
#include <BBB.h>
#include <math.h>
#include “VectorStructOparator.h”

int main(void){
```

今まで

main.c

```
#include <stdio.h>
extern　　“C”
{
#include <AAA.h>
#include <BBB.h>
 }
#include <math.h>
#include “VectorStructOparator.h”
int main(void){
```

これから

main.cpp

＊これはC++からCの関数を呼ぶおまじないで，気になるまでは気にしなくてよい．詳しく知りたければグーグル先生を頼ること.

インストール

パスの通ったフォルダにsrc/以下の本プログラムをおく．
もしくは，使用の度，main.cppと同じフォルダにおく．
以上

サンプル．(2D)

main.cpp

main.cpp › main(void)

```
1  #include <stdio.h>
2  #include "../src/VectorStructOperator.h"
3
4  int main(void)
5  {
6      Vector2D      u,v,w,zero;
7      double        a = 2.0;
8
9      u.x = 1;
10     u.y = 0;
11
12     v.x = 0;
13     v.y = 1;
14
15     zero = Zero2D;
16
17     printf("u = %f %f\n",u.x,u.y);
18     printf("v = %f %f\n",v.x,v.y);
19     printf("zero = %f %f\n",zero.x,zero.y);
20
21     w = u + v;
22     printf("u + v = %f %f\n",w.x,w.y);
23
24     w = zero;
25     w += u;
26     printf("w = zero, w += u , w = %f %f\n",w.x,w.y);
27
28     w = u - v;
29     printf("w = u - v, w = %f %f\n",w.x,w.y);
30
31     w = zero;
32     w -= u;
33     printf("w = zero, w -= u,w = %f %f\n",w.x,w.y);
```

インストールが完了したなら，#include <VectorStructOperator.h>でも良い.

ベクトル,u,v,w,zeroの定義.

ベクトル u, vに値を代入．「.」をつけることで，各要素にアクセスできる.

Zero2Dには０ベクトルがそもそも規定されている．ここでは，あえて代入した.

ベクトル同士の和を行うことができる.

+=を使うことができる.

ベクトル同士の差を行うことができる.

-=を使うことができる.

```
35      w = a * u;
36      printf("a * u = %f %f\n",w.x,w.y);
37
38      w = u * a;
39      printf("u * a = %f %f\n",w.x,w.y);
40
41      w = -u;
42      printf("-u = %f %f\n",w.x,w.y);
43
44      w = u / a;
45      printf("u / a = %f %f\n",w.x,w.y);
46
47      u /= a;
48      printf("u /= a, u = %f %f\n",u.x,u.y);
49
50      printf("u * v = %f \n",u * v);
51
52      printf("Dot2D(u,v) = %f \n",Dot2D(u,v));
53
54      printf("u ^ v = %f\n",u ^ v);
55
56      printf("Cross2D(u,v) = %f\n",Cross2D(u,v));
57
58      printf("~u = %f \n",~u);
59
60      printf("Norm2D(u) = %f \n",Norm2D(u));
61
62      return 0;
63  }
64
```

ベクトル×スカラー（前値系）

ベクトル×スカラー（後置系）

単項演算子「-」を使うことができる.

ベクトル/スカラー（後置系のみ！）

/=を使うことができる.

内積を計算する.

内積を計算する. 記号でも可能

クロス積を計算する.

クロス積を計算する. 記号でも可能

ベクトルの大きさを計算する.

ベクトルの大きさを計算する. 記号でも可能

サンプル．(3D)

基本仕様は同じである．クロス積は3次元の場合は外積となる．

最後に

本プログラムを使えば，あなたが作った紙の上での計算のように，自然にプログラムをすることが可能になる．またコードの量を大幅に減らし，バグを未然に防ぐこともできるだろう．

ベクトル同士の和，差，積，．．．など考え得る，演算はすべて実装している．さらに，ノルム，内積，外積なども簡単に呼び出し，使用できるため，直感的なプログラム制作が可能になる．

AdvanceフォルダにはGLSC3.8.6以降,mybasic8.0以降がインストールされた環境において，実行可能なサンプルプログラムが入っている．仮に環境が整っていなくても，コードを読むことはあなたの理解の助けとなるだろう．
ソースは「小林　秋山正和　GLSC CREST」などとググれば出てくるだろう．

本プログラムはバグがないように十分に注意して作られたものであるが，仮にバグや要望がある場合はメールしてください．(masakazu.akiyamあっとgmail.com)
本プログラムは再配布可能です．すきにやっちゃってください．ただし，使用の場合，作者にメールをいただければ実態把握ができます．よろしく

2011.10.11 著者　秋山正和

2013.1.8　3次元ベクトルの大きさを返す関数（~,Norm3D）に重大なバグがありました.
報告してくれた小林様，砂田様ありがとうございました.