シミュレーションと可視化の講習会

明治大学　秋山正和

主催：明治大学先端数理科学インスティテュート

はじめに

概要：

数理モデルから，現象解明のために様々な情報を得るためには，数理的な解析は欠かせません．数理解析では手で解析的に解くことができるケースは稀で，大抵の場合は数値的に計算して大まかな特徴を捉えることになります．捉えた特徴から，思わぬ証明のアイデアを得ることもあります．さらに，これと同じくらい重要なことは，このような数値をわかりやすい絵に変換すること（可視化）です．通常，可視化は汎用のツールを使って行うことが多いのですが，「ここにこんな棒グラフをつけたい」「ここにパラメタを印字したい」「ここの３Dアニメーションを別の角度からも見たい」など「こうしたいという欲望」と「できないという現実」の歯痒さを経験します．そこで，本講習会では，数値計算と可視化の方法を手と頭を動かし，実践的にレクチャーしながら数値計算と可視化の方法を学びます．講習はC言語(C++言語も一部使用)を中心として解説します．PCは各自で持ってきてください．秋山が作成したいくつかの教材（資料およびソースファイル）を当日までにメールもしくはHPから配布します．教材は常微分方程式の数値計算法，反応拡散方程式の数値計算法，パラメータサーチプログラム，可視化ソフトGLSC3Dを用います．これらの教材はそのまま，もしくは一部改変することで自身の勉強・研究等に活かすことができます．

本日

２５日

０９：３０－１０：００ 受付

１０：００－１０：０５ 趣旨説明

１０：０５－１０：３０ Unix系入門

内容：：：Unixのコマンドライン操作，便利な機能，簡単なシェルスクリプトも教えます.

１０：３０ －１２：００ C言語におけるプログラムの動作原理，計算ライブラリ作成，高速化の方法

内容：：：stdio.hって何？a.outでどうゆうこと？Makeって何？libxxxx.aとかライブラリって何？ちょっとの工夫で高速化できる方法論などを教えます.

１２：００－１３：００ お昼休憩

１３：００－１４：４５ 秋山謹製 常微分方程式計算ライブラリの紹介

内容：：：常微分方程式の計算方法を教えます．時間があれば，それを更に高速化した埋込み型ルンゲクッタ法を教えます.

１４：４５－１５：００ 休憩

１５：００－１６：４５ 反応拡散方程式の数値計算法入門(予定)

内容：：：明治大において開催された，反応拡散夏の学校の教材を一部利用して，1Dと２Dの反応拡散方程式の数値計算法入門を行います.

１６：４５－１７：００ 休憩

１７：００－１８：００ 秋山謹製 パラメタサーチプログラムの使い方(予定)

内容：：：効率的にパラメタ探索を行うためのツールを紹介します.

明日

２６日

１０：００－１１：００ 秋山謹製 可視化ソフトGLSC3Dの紹介およびインストール

内容：：：GLSC3Dがあれば，どんな可視化もできます．簡易な例を用いて解説します．

１１：００－１２：００ GLSC3Dを用いた，数値計算例の紹介

１２：００－１３：００ 質問コーナー＆解散

明日


MEIJI UNIVERSIT


防災センター
Fire Command Center
防災センター
現在地
You Are Here
御用の方はインターホンを押してください


明治大学
MEIJI UNIVERSITY

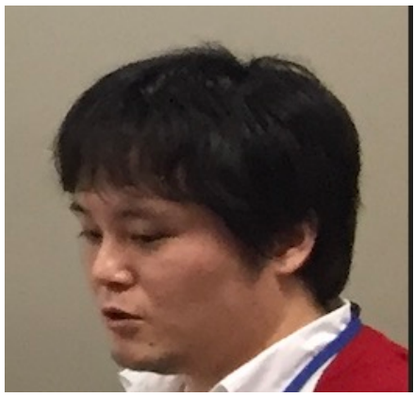
補助者紹介

Pr. 須志田さん

サレジオ工業高等専門学校

Mr. 舘入さん

北海道大学



Dr. 関坂さん

明治大学

Dr. 田邉さん

明治大学

参加者紹介

所属情報(例 明治大学MIMS)	ご身分	参
明治大学	教員	両
関西学院大学理工学部	ポストドクター	両
サレジオ工業高等専門学校	教員	両
明治大学　総合数理学部	教員	両
岐阜大学工学部	教員	両
三井情報株式会社	その他	両
総合数理学部現象数理学科	学部生	両
公益財団法人計算科学振興財団	その他	2
山梨大学・工学専攻	博士課程	2
明治大学総合数理学部	教員	両
明治大学総合数理学部	学部生	2
豊橋技術科学大学工学部	教員	両
明治大学大学院先端数理科学研究科現象数理学専攻小川研究室	修士課程	両

本講習会のルール

全員の同期を取りながら説明をするので，
待っててほしいときは必ず手を上げてください．

Step1 ーーーーーーーー
Step2 ーーーーーーーー
Step3 ーーーーーーーー
Step4 ーーーーーーーー
Step5 ーーーーーーーー

※　遠慮して手をあげないと，講師は余計に困ってしまいます．．．

ネットワークへの接続

Mac/Win/Linux共通

eduroamのアカウントとパスワードを入力し，インターネットへ接続できることを確認してください．

ID

Passwd

printed at 2019/12/20 10:18:39

Visitor account information

User name	Password	from	until
KCL01107@meiji.v.eduroam.jp (UC-K,UC-C,UC-L,Zero,One,One,Zero,Seven)	!thPdKn7dcTJ (Excl,LC-T,LC-H,LC-P,LC-D,UC-K,LC-N,Seven,LC-D,LC-C,UC-T,UC-J)	2019-12-20	2019-12-26

0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ?!@:[]

ビジター用アカウントについて(About this visitor's account)

このアカウントは訪問先機関においてeduroam無線LAN接続サービスを利用するために発行されたビジター用アカウントです。 「eduroam」のSSIDを選択し、上記の「User name」「Password」を入力してネットワークに接続してください。（すでに、eduroam用アカウントをお持ちの場合は、そのアカウントを利用することもできます。） 機器ごとの詳細な接続設定方法については、「利用者向け情報」(http://www.eduroam.jp/for_users/) およびインストール手順(https://federated-id.eduroam.jp/guide/install.php)を参照ください。 ---------- 注意 ※訪問先機関の利用規程等を遵守してください。 ※訪問先機関以外のeduroam環境では利用できないことがあります。	This account is issued for a visitor to use the Wireless LAN Connection Service called "eduroam". Please login with above "User Name" and "Password" to access the wireless network advertised with the SSID "eduroam". Please refer to the user's guide (http://www.eduroam.jp/for_users/) or installation instruction (https://federated-id.eduroam.jp/guide/install.php) for configuring your device (in Japanese). (If you have an eduroam account issued by another institution, you may use it as-is.) ---------- Notes: - Please follow the rules in the visiting organization. - This account may not be effective in other organizations.

User name: KCL01107@meiji.v.eduroam.jp

氏名、所属、電話番号、メールアドレスを記入して発行者に提出してください。

Please fill in your Name, Affiliation, Phone number, and E-mail address and submit it to the issuer of this visitor account.

Name:

Address:

E-mail:

Phone number:

講習会の資料

Googleで「秋山正和」を検索

左のカラムから「集中講義」を選択

今日の講習会のリンクをクリック

http://www.isc.meiji.ac.jp/~akiyama_masakazu/Lecture/SimuAndVisu/Slide

http://www.isc.meiji.ac.jp/~akiyama_masakazu/Lecture/SimuAndVisu/Common.zip

もしダメなら・・・・

USBメモリ

USBメモリを刺して,

Commonをデスクトップに置いてください.

Unix系入門

Commonフォルダにターミナルを移動させる.

1_Test_HelloWorld.cをコンパイルする.

a.outを実行する.

$: cc 1_Test_HelloWorld.c
$: ./a.out

※./の意味を教える.

ls, ./,
ls -la, ../
pwd, mv,
cd, cp,
cd + D&D mkdir

※上記を練習する

$: cc 2_Test_Math.c
$: ./a.out　　※-lmの必要性を教える.

$: cc 2_Test_Math.c -lm -o abxy
$: ./a.out
$: ./abxy

※実行ファイルに関して教える

実行権限とは

$: ls -la

```
-rwxr-xr-x   1 masakazu  staff   8552  7 29 13:18 a.out*
```

$:　./a.out

で反応があるはず

$:　chmod 600 a.out

```
-rw-------   1 masakazu  staff   8552  7 29 13:18 a.out
```

となる．すると

$:　./a.out

permission denied: ./a.out

となる．何故か？

r : readable
w: writable
x: executable

rは４
wは２
xは１
の値を持つと約束する.

d	r	w	x	r	w	x	r	w	x
directory	読み	書き	実行	読み	書き	実行	読み	書き	実行
	自分	自分	自分	グループ	グループ	グループ	他人	他人	他人

例１
600
は
6=4*1+2*1+1*0
0=4*0+2*0+1*0
0=4*0+2*0+1*0

つまり自分は読めるし，書けるけど，実行ができない
グループは読めない，書けない，実行ができない
他人は読めない，書けない，実行ができない

$:　./a.out
permission denied: ./a.out

例2

777

は

7=4*1+2*1+1*1

7=4*1+2*1+1*1

7=4*1+2*1+1*1

なので，だれでも

何でも出来ちゃう.

$: cc 3_Test_destroy.c

$: ./a.out

```
Destroy your PC ....
```

$: mv a.out ls

$: ls

$: ./ls

※ウイルスの話,

適切な実行権限を意識する

例３

000

誰も何もできない.

シェル

$: echo $SHELL

多分bashのはず

csh, zsh, tcshと打ってみよう．シェルが切り替わる

tcsh < csh < bash < zshとだんだん新しくなってきた.

zshは現状では一番新しいシェルで，多機能かつ高速

$: bash

$: cd pic

$: ls

シェル

$: ls *.png　　拡張子がpngのファイルを列挙する.

$: ls 000??0.png　?は任意の１文字に一致

$: ls *[0248].png　最後が0,2,4,8のどれかに一致

$: ls *[!6].png　　最後が6でないに一致

$: for x in *0.png　最後が0に一致するものを列挙
> do
> echo $x
> done

シェル

```
bash-3.2$ mkdir out
bash-3.2$ for x in *0.png
> do
> echo "copy... $x..."
> cp $x out/
> done
copy... 000010.png...
copy... 000020.png...
copy... 000030.png...
copy... 000040.png...
copy... 000050.png...
copy... 000060.png...
copy... 000070.png...
copy... 000080.png...
copy... 000090.png...
copy... 000100.png...
copy... 000110.png...
copy... 000120.png...
copy... 000130.png...
copy... 000140.png...
copy... 000150.png...
copy... 000160.png...
copy... 000170.png...
copy... 000180.png...
copy... 000190.png...
copy... 000200.png...
copy... 000210.png...
copy... 000220.png...
copy... 000230.png...
copy... 000240.png...
copy... 000250.png...
bash-3.2$
```

最後が0に一致するものを列挙して，それをoutフォルダにコピーする

他にも

- **拡張子を一括で変換**
- **画像ファイルから，特定のエリアを切り抜いて一括保存**
- **動画ファイルを連番ファイルにして，ループするGIFアニメを作成**

トイレ休憩5min

binって？？

binaryのこと　　即実行可能なファイル=実行ファイル

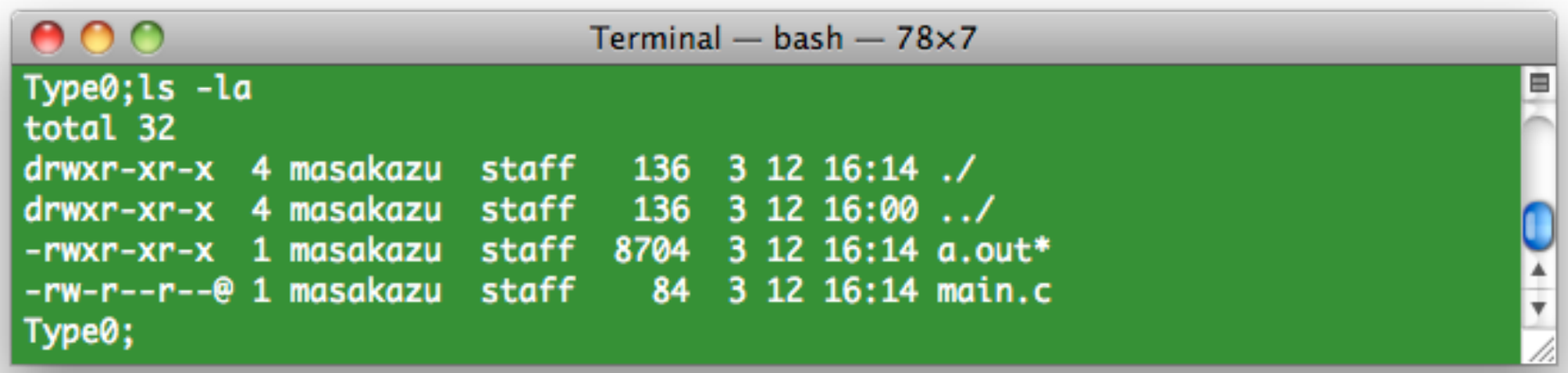
/bin　/usr/local/bin　/usr/bin　の下に大概おかれる。

演習

Type0に行き。main.cを読みコンパイルせよ。

($ cc main.c)

(*できたファイルはa.outが実行ファイルである。)

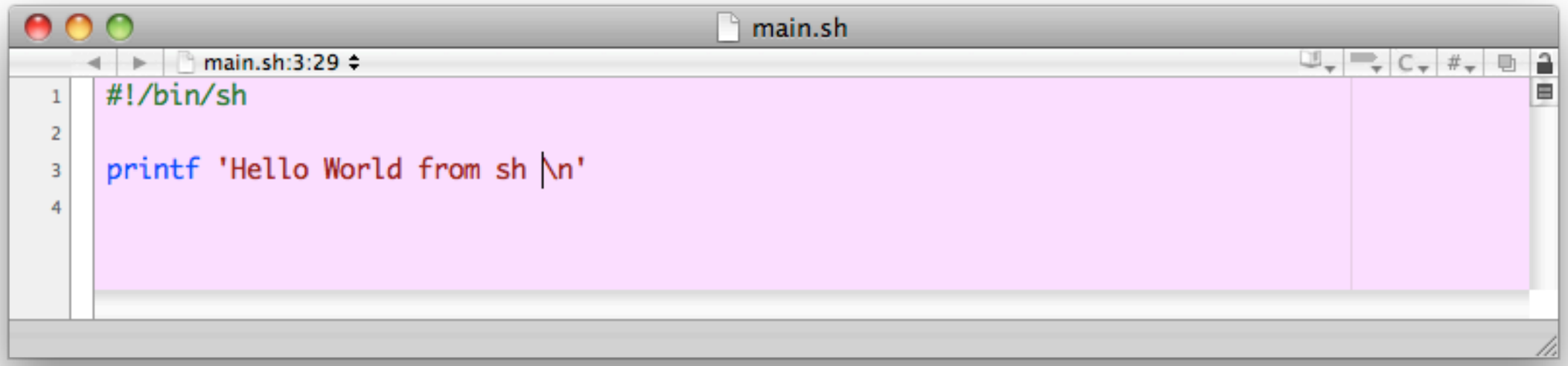


binって？？

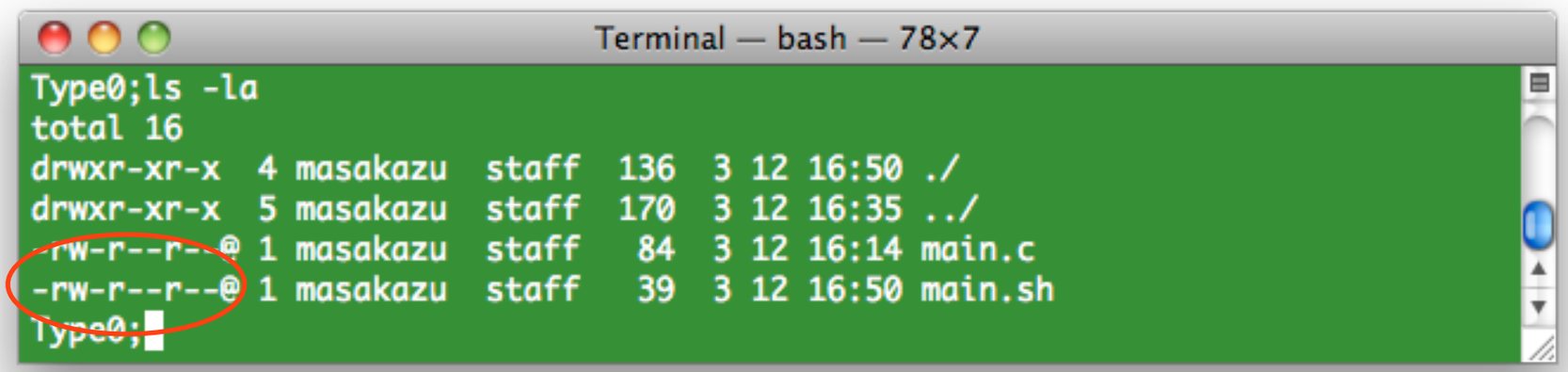
演習

$ touch main.sh

$ open main.sh (またはダブルクリック)



$ ls -la

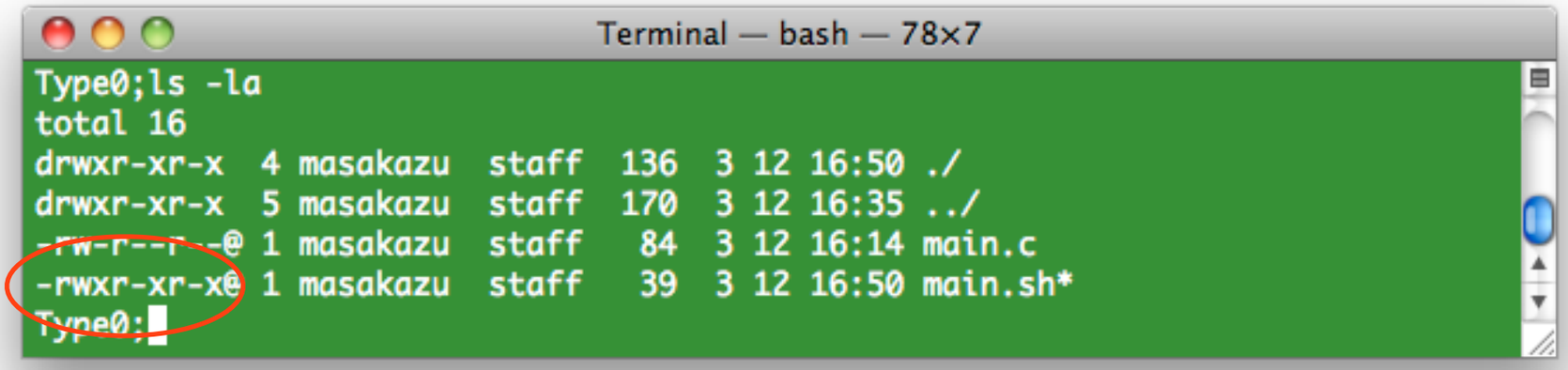


binって？？

演習

$ chmod +x main.sh

$ ls -la



$ ./main.sh

実行ファイルとは実行可能なファイルのことである。実行可能とはx(execute)フラッグがたったファイルのことである。実行ファイルには、a.outの例のようにその環境でしか実行できないもの(コンパイラ型)と、main.shのようにソースファイルが実行ファイルとなるもの(インタプリタ型)の２種類がある。インタプリタ型については後述。

演習

cc main.c

main.c

stdio.h

cc -c main.c

ソース(インクルード(後述)文も込みの)をコンパイルしてオブジェクトファイル(後述)を作る。

main.o

Cのlib

cc main.o -o a.out

oファイルにCのライブラリ(後述)を関連させて(これをリンクと言う)実行ファイル(a.out)を作ると同時に実行権限+xを与える

a.out

includeって??

ヘッダー(ホニャララ.h)をまとめて入れてあるフォルダ。

ヘッダーそのものだったりもする。

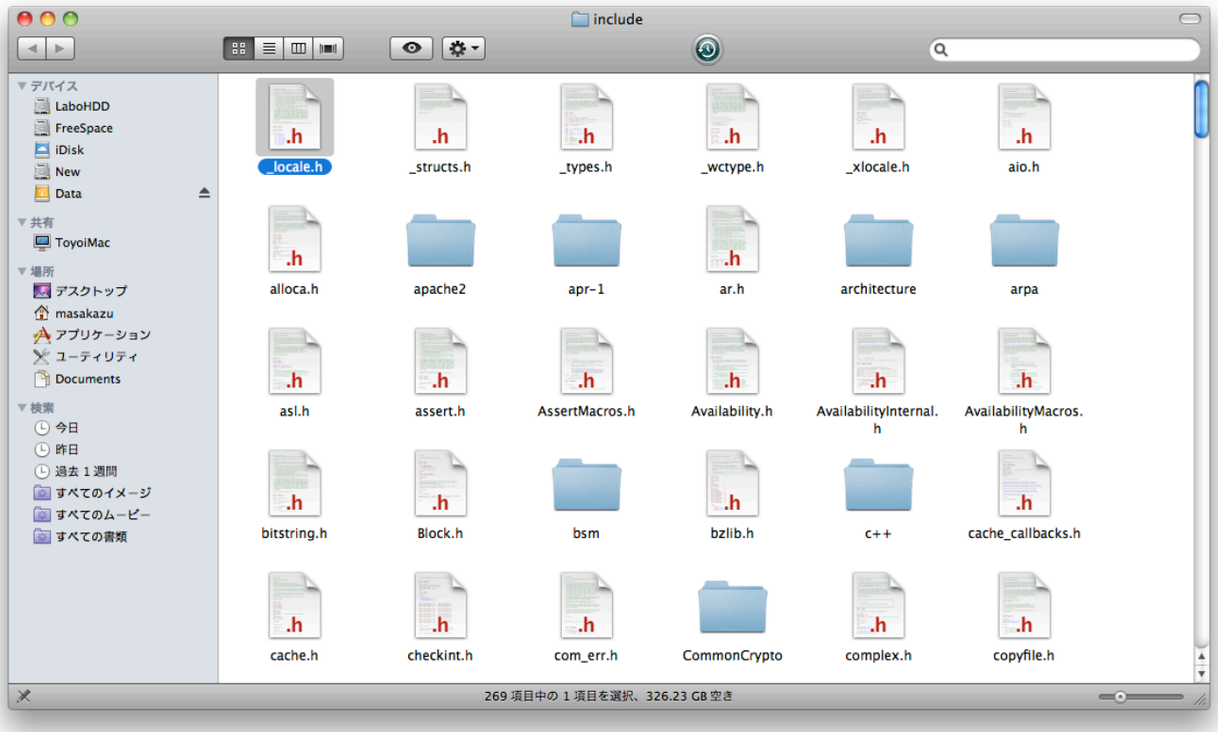
ヘッダーって??

演習4

$ cd /usr/include

$ open .

これら全部ヘッダー →



/usr/local/include/

/Developer/usr/include/

代表的なincludeの場所

libって??

オブジェクトファイルをまとめたファイル=ライブラリをまとめたフォルダ。ライブラリそのものだったりもする。

オブジェクトファイルって？？

ソースからgcc -cすることによって得られる、中間的なファイル。人間には解読できない。実行ファイルと似ているがちがう。

bin,include,lib
ここまでのまとめ

bin: 実行ファイル(binary)。もしくはそれをまとめたフォルダ。
コンパイラ型とインタプリタ型の２種類がある。

include:　関数の方宣言などに使われるヘッダー(例　stdio.h)そのもの。もしくはそれをまとめたフォルダ。

lib:オブジェクトファイルをまとめたファイルのこと(library)。
通常lib???.aという名前になる。

bin,include,lib 実践

演習5

$ cd Type1

$ cc main.c

$ ./a.out

main.c

main.c:1:1 <選択されたシンボルなし>

```c
#include <stdio.h>

void masakazu_hello1(void)
{
    printf("Hello World from masakazu_hello1 \n");
}
void masakazu_hello2(double num)
{
    printf("Hello World from masakazu_hello2 num = %f\n",num);
}

int main(void)
{
    masakazu_hello1();
    masakazu_hello2(3.14);
    return 0;
}
```

Terminal — bash — 82×6

```
Type1;./a.out
Hello World from masakazu_hello1
Hello World from masakazu_hello2 num = 3.140000
Type1;
```

bin,include,lib 実践

演習6

$ cd Type2

$ cc main.c

$ ./a.out

masakazu_funcs.c
を変更する

と、、、

main.c

```
1  #include <stdio.h>
2
3  #include "masakazu_funcs.c"
4
5  int main(void)
6  {
7      masakazu_hello1();
8      masakazu_hello2(3.14);
9      return 0;
10 }
```

masakazu_funcs.c

```
1
2  void masakazu_hello1(void)
3  {
4      printf("Hello World from masakazu_hello1 \n");
5  }
6  void masakazu_hello2(double num)
7  {
8      printf("Hello World from masakazu_hello2 num = %f\n",num);
9  }
10
```

Terminal — bash — 82×6

```
a.out*                  masakazu_funcs.c
Type2;cc main.c
Type2;./a.out
Hello World from masakazu_hello1
Hello World from masakazu_hello2 num = 3.140000
Type2;
```

bin,include,lib 実践

演習7

```
$ cd Type3
$ cc -c masakazu_funcs.c
$ cc main.c masakazu_funcs.o
$ ./a.out
```

objectファイルを作っている。

a.outを作っている。

masakazu_funcs.hを消して、3.14を3に変更すると変なことになる。つまりヘッダーは必ず必要であるということがわかる。

bin,include,lib 実践

演習8

$ cd Type4

$ ar -cr libmasakazu.a masakazu_funcs.o　　libmasakazu.aの作成

$ cc main.c libmasakazu.a　　libmasakazu.aを使って、a.outを作る。

＊$ cc main.c -L./ -lmasakazu　としてもよい

-Lの後ろはそのライブラリのある場所を指定する。今は「.」なので./を指定する。また-Lオプションがつくと、-labcdは直ちにlibabcd.aと読み替えられる。

bin,include,lib 実践

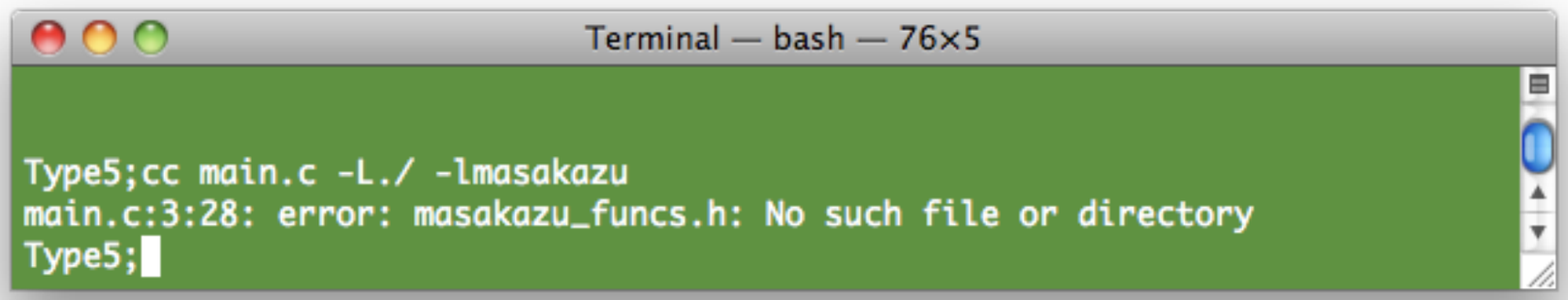
演習9

$ cd Type5

libフォルダ、includeフォルダを作り、その中にそれぞれ

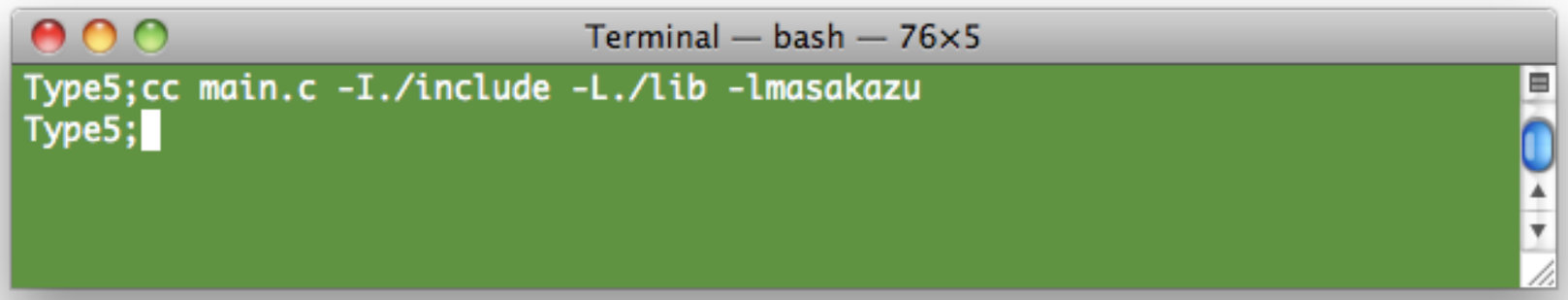
libmasakazu.aとmasakazu_funcs.hを入れる。

$ cc main.c -L./ -lmasakazu



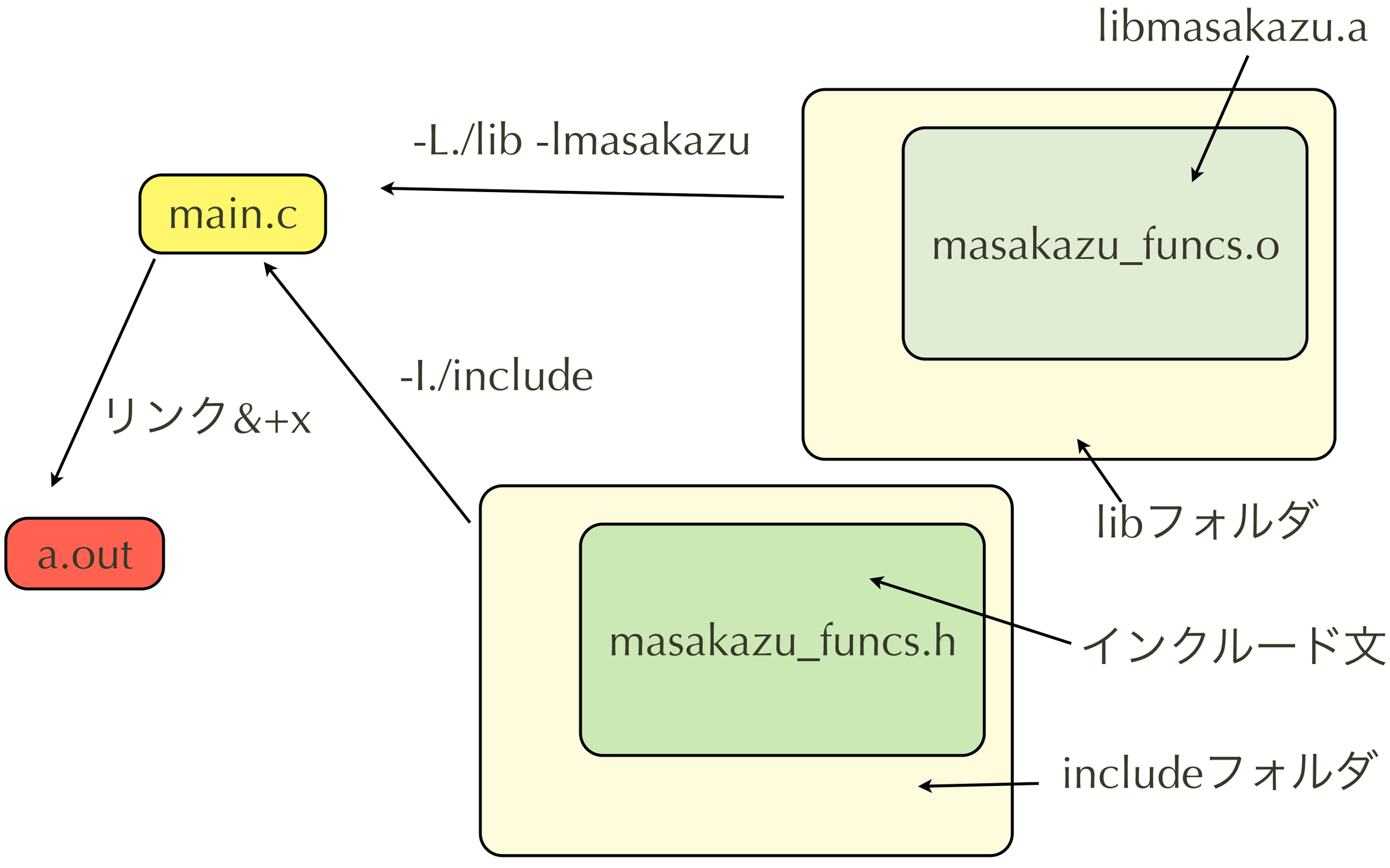
そんなもんないとおこられる。当然。。

$ cc main.c -I./include -L./lib -lmasakazu



今度はOK

bin,include,lib 実践



トイレ休憩

よく使う関数はライブラリ化しよう

Type5-1

プロトタイプ宣言を作る。(myheader.h)

myheader.h

```
1  double  sum3(double a,double b,double c);
2  double  mul3(double a,double b,double c);
3  double  fac3(double a,double b,double c);
4
5
```

cc -cでオブジェクトファイルを作成
ar -crでライブラリを作成

(libmyfunc.a)

cc main.c -L(libのある場所)

-l(ライブラリ名) -I(include

のある場所)

よく使う関数

sum3.c

```
1
2  double  sum3(double a,double b,double c)
3  {
4      return a + b + c;
5  }
6
7
```

fac3.c

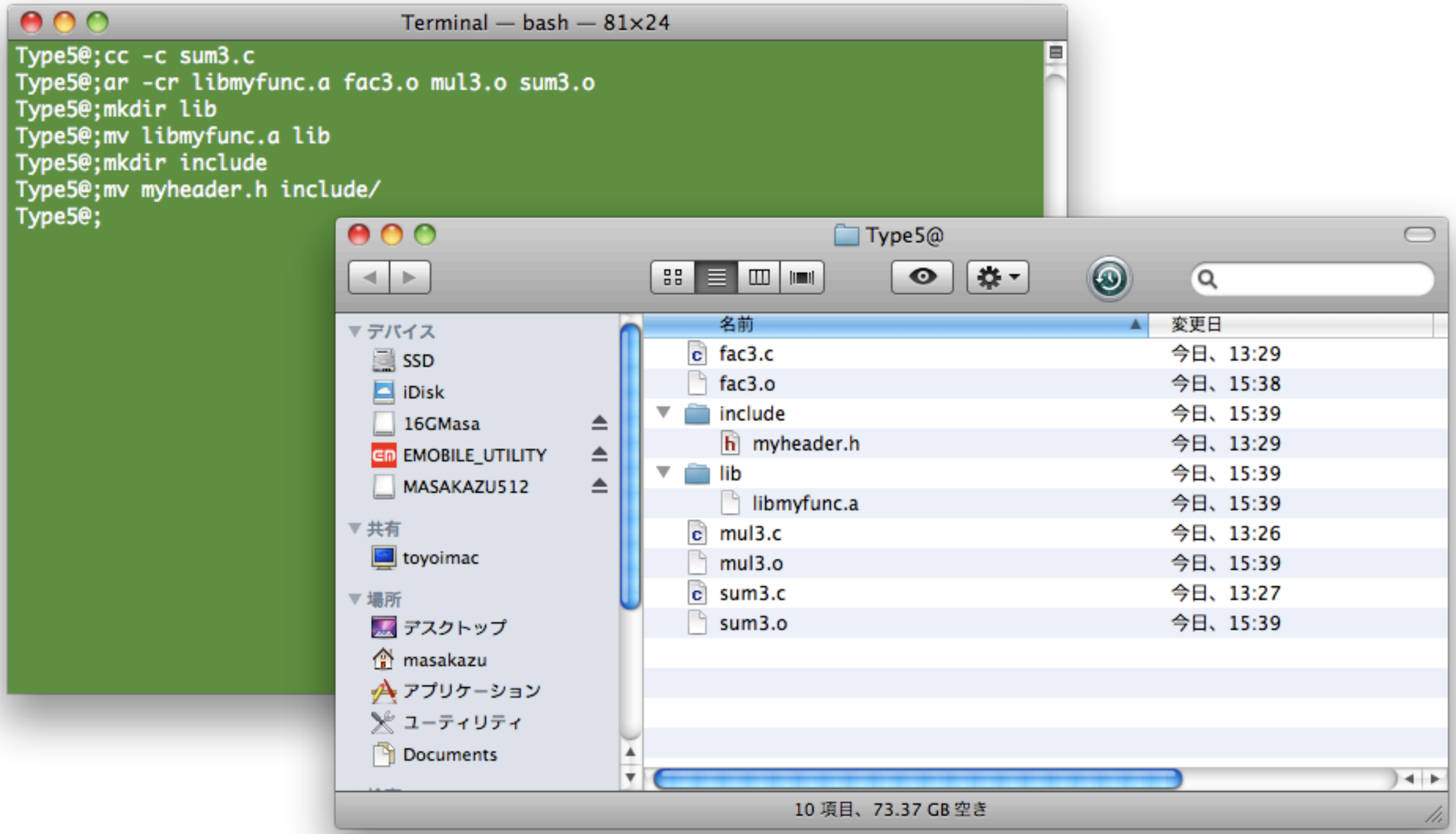
```
1  #include <math.h>
2
3  double  fac3(double a,double b,double c)
4  {
5      return pow(pow(a,b),c);
6  }
7
```

mul3.c

```
1
2  double  mul3(double a,double b,double c)
3  {
4      return a * b * c;
5  }
6
7
```

よく使う関数はライブラリ化しよう

うまくいけばこんな感じに



よく使う関数はライブラリ化しよう

うまくいけばこんな感じに

main.c

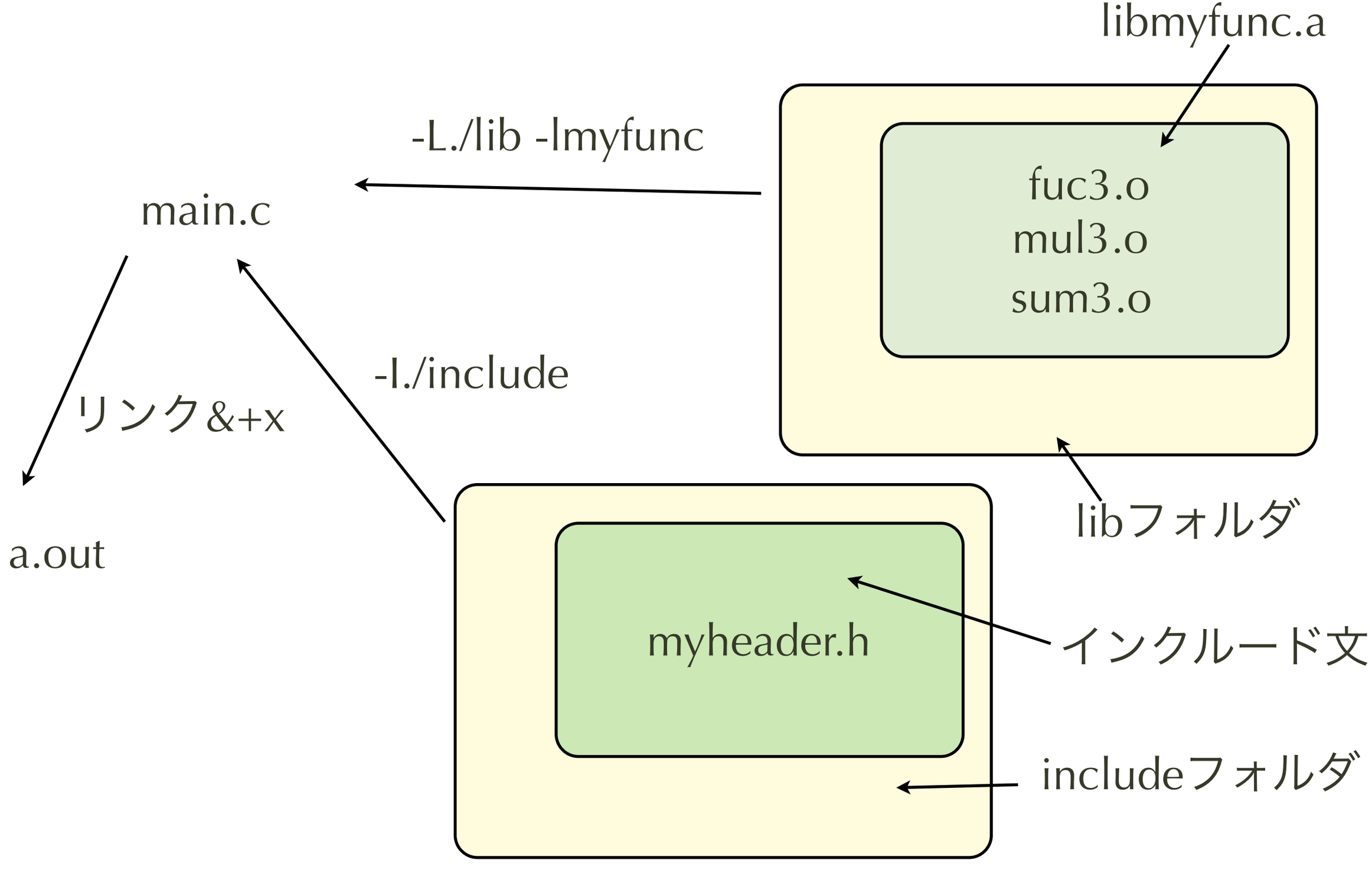
main.c:11:5 main()

```c
#include <stdio.h>
#include <myheader.h>

int main(void)
{
    int a = 2,b = 3,c = 4;

    printf("fac3(%d,%d,%d) = %f\n",a,b,c,fac3(a,b,c));
    printf("mul3(%d,%d,%d) = %f\n",a,b,c,mul3(a,b,c));
    printf("sum3(%d,%d,%d) = %f\n",a,b,c,sum3(a,b,c));

    return 0;
}
```

Terminal — bash — 81×8

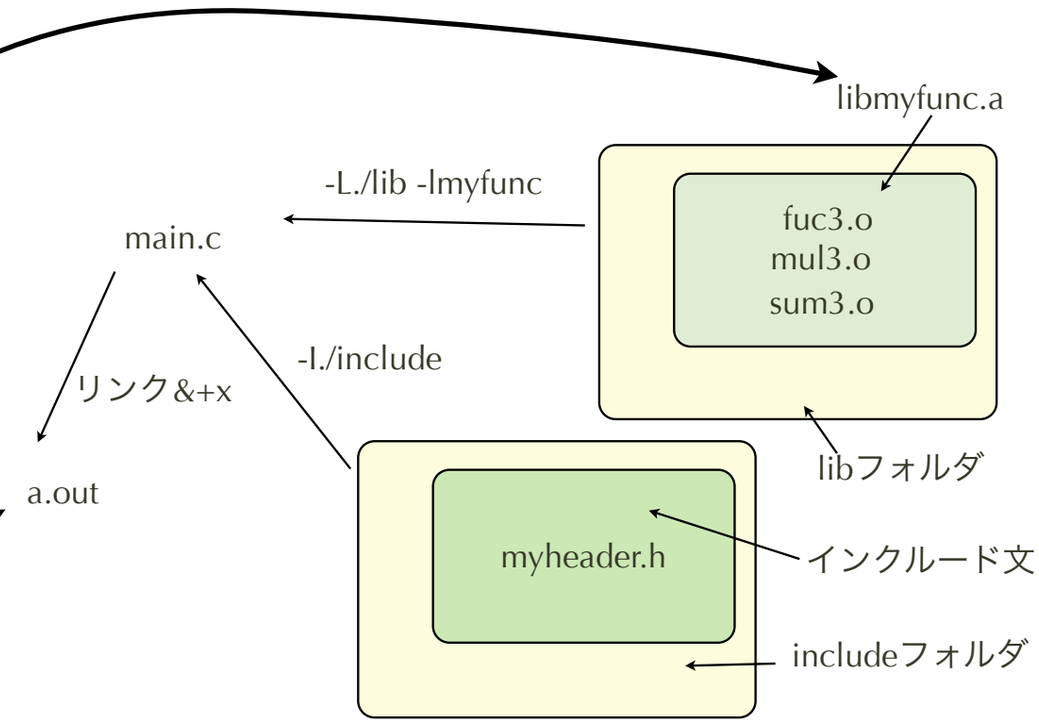
```
Type5@;cc main.c -L./lib/ -lmyfunc -I./include
Type5@;./a.out
fac3(2,3,4) = 4096.000000
mul3(2,3,4) = 24.000000
sum3(2,3,4) = 9.000000
Type5@;
```

bin,include,lib 実践



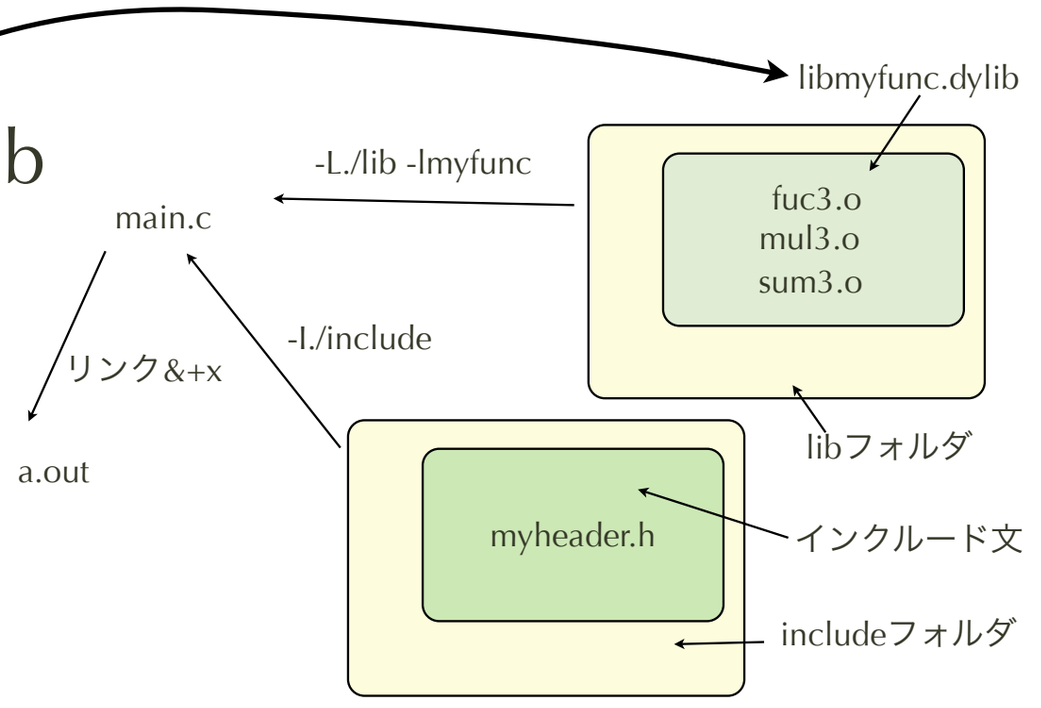
bin,include,lib 補足

静的ライブラリ:ほにゃほにゃ.a



静的実行ファイル

動的ライブラリ:ほにゃほにゃ.dylib



動的実行ファイル

動的ライブラリの例を見せる。

Macの中で実際に lib, bin, includeを探してみよう

C言語とポインタ編

を作ったが割愛・・・

C言語小技

小技集 1

マクロってすごい。

main.c

main.c:16:1 ▸ main()

```c
#include <stdio.h>

#define SQ(i)       ((i) * (i))

#define         MAX(a, b)       (((a) > (b)) ? (a) : (b))
#define         MIN(a, b)       (((a) < (b)) ? (a) : (b))

int main(void)
{
    double  tmp,hoge;

    tmp = 2.0;
    hoge = 3.0;

    printf("SQ(%f) = %f \n",tmp,SQ(tmp));
    printf("MAX(%f,%f) = %f \n",tmp,hoge,MAX(tmp,hoge));
    printf("MIN(%f,%f) = %f \n",tmp,hoge,MIN(tmp,hoge));

    return 0;
}
```

Terminal — bash — 58×7

```
Kowaza2;./a.out
SQ(2.000000) = 4.000000
MAX(2.000000,3.000000) = 3.000000
MIN(2.000000,3.000000) = 2.000000
Kowaza2;
```

$ cc -E main.c

Terminal — bash — 83×23

```
  ;
# 444 "/usr/include/stdio.h" 2 3 4
# 2 "main.c" 2



int main(void)
{
 double tmp,hoge;

 tmp = 2.0;
 hoge = 3.0;

 printf("SQ(%f) = %f \n",tmp,((tmp) * (tmp)));
 printf("MAX(%f,%f) = %f \n",tmp,hoge,(((tmp) > (hoge)) ? (tmp) : (hoge)));
 printf("MIN(%f,%f) = %f \n",tmp,hoge,(((tmp) < (hoge)) ? (tmp) : (hoge)));

 return 0;
}
Kowaza2;
```

マクロはこのように解釈せれているのだ！

小技集2

関数形式マクロ

main.c

main.c:11:6 ▸ main() ▸

```c
#include <stdio.h>
#define     Imax        (10)
#define     Jmax        (20)

#define     u(i,j)      u[(Imax) * (j) + (i)]

int main(void)
{
    int     i,j;
    double  u[Imax * Jmax];

    for(i = 0;i < Imax;i++)
    {
        for(j = 0;j < Jmax;j++)
        {
            u(i,j) = 0.0;
        }
    }
    return 0;
}
```

Terminal — bash — 48×24

```
        const char * , va_list)
  ;
# 444 "/usr/include/stdio.h" 2 3 4
# 2 "main.c" 2



int main(void)
{
 int i,j;
 double u[(10) * (20)];

 for(i = 0;i < (10);i++)
 {
  for(j = 0;j < (20);j++)
  {
   u[((10)) * (j) + (i)] = 0.0;
  }
 }
 return 0;
}
Kowaza3;
```

小技集2

関数形式マクロ

main.c

main.c:11:6 ▸ main() ▸

```c
#include <stdio.h>
#define     Imax        (10)
#define     Jmax        (20)

#define     u(i,j)      u[(Imax) * (j) + (i)]

int main(void)
{
    int     i,j;
    double  u[Imax * Jmax];

    for(i = 0;i < Imax;i++)
    {
        for(j = 0;j < Jmax;j++)
        {
            u(i,j) = 0.0;
        }
    }
    return 0;
}
```

Terminal — bash — 48×24

```
        const char * , va_list)
  ;
# 444 "/usr/include/stdio.h" 2 3 4
# 2 "main.c" 2



int main(void)
{
 int i,j;
 double u[(10) * (20)];

 for(i = 0;i < (10);i++)
 {
  for(j = 0;j < (20);j++)
  {
   u[((10)) * (j) + (i)] = 0.0;
  }
 }
 return 0;
}
Kowaza3;
```

小技集3

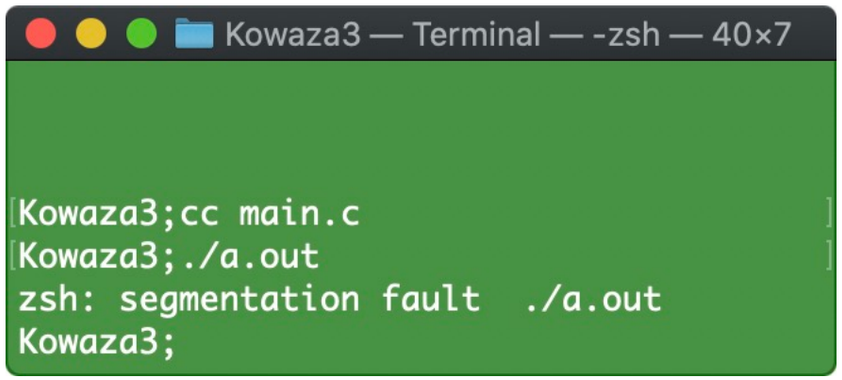
```c
#include <stdio.h>
#include <limits.h>
#define     Imax        (1000)
#define     Jmax        (2000)

double rnd(void)//0~1
{
    static unsigned long x=123456789,y=362436069,z=521288629,w=88675123;
    unsigned long t;
    t=(x^(x<<11));x=y;y=z;z=w;
    return( w=(w^(w>>19))^(t^(t>>8)) ) / (double)(ULONG_MAX);
}

int main(void)
{
    int     i,j;
    double  u[Imax][Jmax];

    for(i = 0;i < Imax;i++)
    {
        for(j = 0;j < Jmax;j++)
        {
            u[i][j] = rnd();
            //printf("u[%05d][%05d] = %15.15f\n",i,j,u[i][j]);
        }
    }
    double wa = 0;
    for(i = 0;i < Imax;i++)
    {
        for(j = 0;j < Jmax;j++)
        {
            wa += u[i][j];
        }
    }
    printf("Sum = %15.15f\n",wa);

    return 0;
}
```

Imaxを100

Jmaxを200だと．．．

Imaxを1000

Jmaxを2000だと．．．



1000*2000*8byte
=16Mbyte

Macのメモリは8G以上
あるのでおかしい？？

小技集4

```c
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#define     Imax        (40000)
#define     Jmax        (20000)

#define         u(i,j)          u[(Imax) * (j) + (i)]

double rnd(void)//0~1
{
    static unsigned long x=123456789,y=362436069,z=521288629,w=88675123;
    unsigned long t;
    t=(x^(x<<11));x=y;y=z;z=w;
    return( w=(w^(w>>19))^(t^(t>>8)) ) / (double)(ULONG_MAX);
}

int main(void)
{
    int     i,j;
    //double    u[Imax][Jmax];
    double *u;
    u = (double *)malloc(sizeof(double) * Imax * Jmax);

    for(i = 0;i < Imax;i++)
    {
        for(j = 0;j < Jmax;j++)
        {
            u(i,j) = rnd();
            //printf("u[%05d][%05d] = %15.15f\n",i,j,u(i,j));
        }
    }
    double wa = 0;
    for(i = 0;i < Imax;i++)
    {
        for(j = 0;j < Jmax;j++)
        {
            wa += u(i,j);
        }
    }
    printf("Sum = %15.15f\n",wa);

    return 0;
}
```

10000*20000*8byte
=1.6Gbyte

40000*20000*8byte
=6.4Gbyte

20000*200000*8byte
=32Gbyte

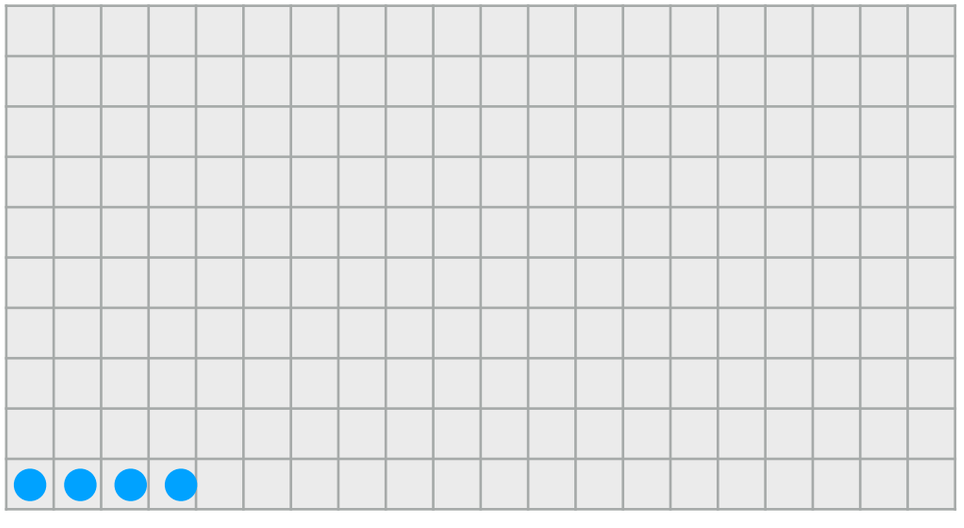
```c
for(i = 0;i < Imax;i++)
{
    for(j = 0;j < Jmax;j++)
    {
        u(i,j) = rnd();
        //printf("u[%05d][%05d] = %15.15f\n",i,j,u(i,j));
    }
}
double wa = 0;
for(i = 0;i < Imax;i++)
{
    for(j = 0;j < Jmax;j++)
    {
        wa += u(i,j);
    }
}
printf("Sum = %15.15f\n",wa);
```

```
Kowaza4;cc main.c
Kowaza4;time ./a.out
Sum = 400006476.598021924495697
./a.out  19.36s user 1.71s system 99% cpu 21.095 total
Kowaza4;
```

メモリ空間の連続性の向き→

```c
for(j = 0;j < Jmax;j++)
{
    for(i = 0;i < Imax;i++)
    {
        u(i,j) = rnd();
        //printf("u[%05d][%05d] = %15.15f\n",i,j,u(i,j));
    }
}
double wa = 0;
 for(j = 0;j < Jmax;j++)
   {
       for(i = 0;i < Imax;i++)
       {
        wa += u(i,j);
    }
}
printf("Sum = %15.15f\n",wa);
```

```
Kowaza4;cc main_waza.c
Kowaza4;time ./a.out
Sum = 400006476.598021924495697
./a.out  5.61s user 1.47s system 97% cpu 7.283 total
Kowaza4;
```



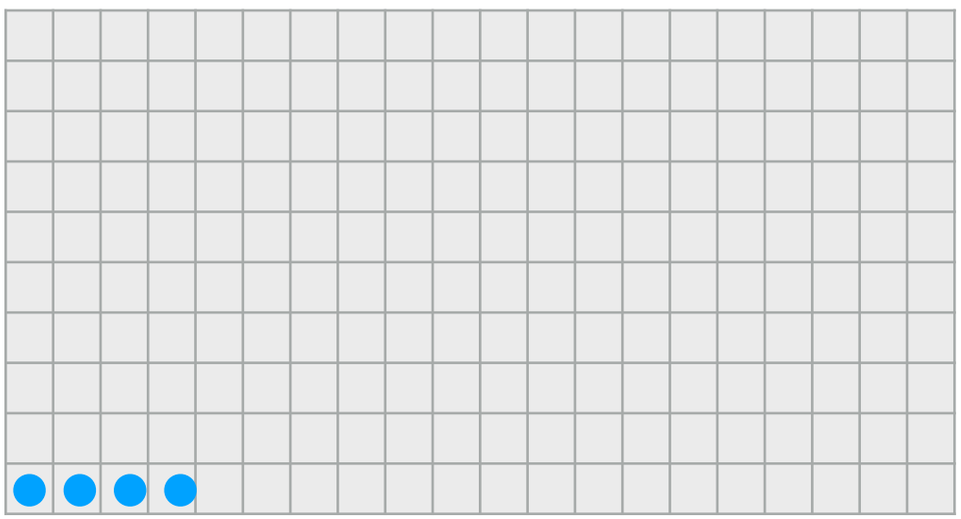
メモリ空間の連続性の向き→

```
for(j = 0;j < Jmax;j++)
{
    for(i = 0;i < Imax;i++)
    {
        u(i,j) = rnd();
        //printf("u[%05d][%05d] = %15.15f\n",i,j,u(i,j));
    }
}
double wa = 0;
 for(j = 0;j < Jmax;j++)
   {
       for(i = 0;i < Imax;i++)
       {
        wa += u(i,j);
    }
}
printf("Sum = %15.15f\n",wa);
```

```
Kowaza4;cc -03 main_waza.c
Kowaza4;time ./a.out
Sum = 400006476.598021924495697
./a.out  2.85s user 1.42s system 94% cpu 4.508 total
```

コンパイラ最適化

```
Kowaza4;cc main_waza.c
Kowaza4;time ./a.out
Sum = 400006476.598021924495697
./a.out  5.61s user 1.47s system 97% cpu 7.283 total
Kowaza4;
```



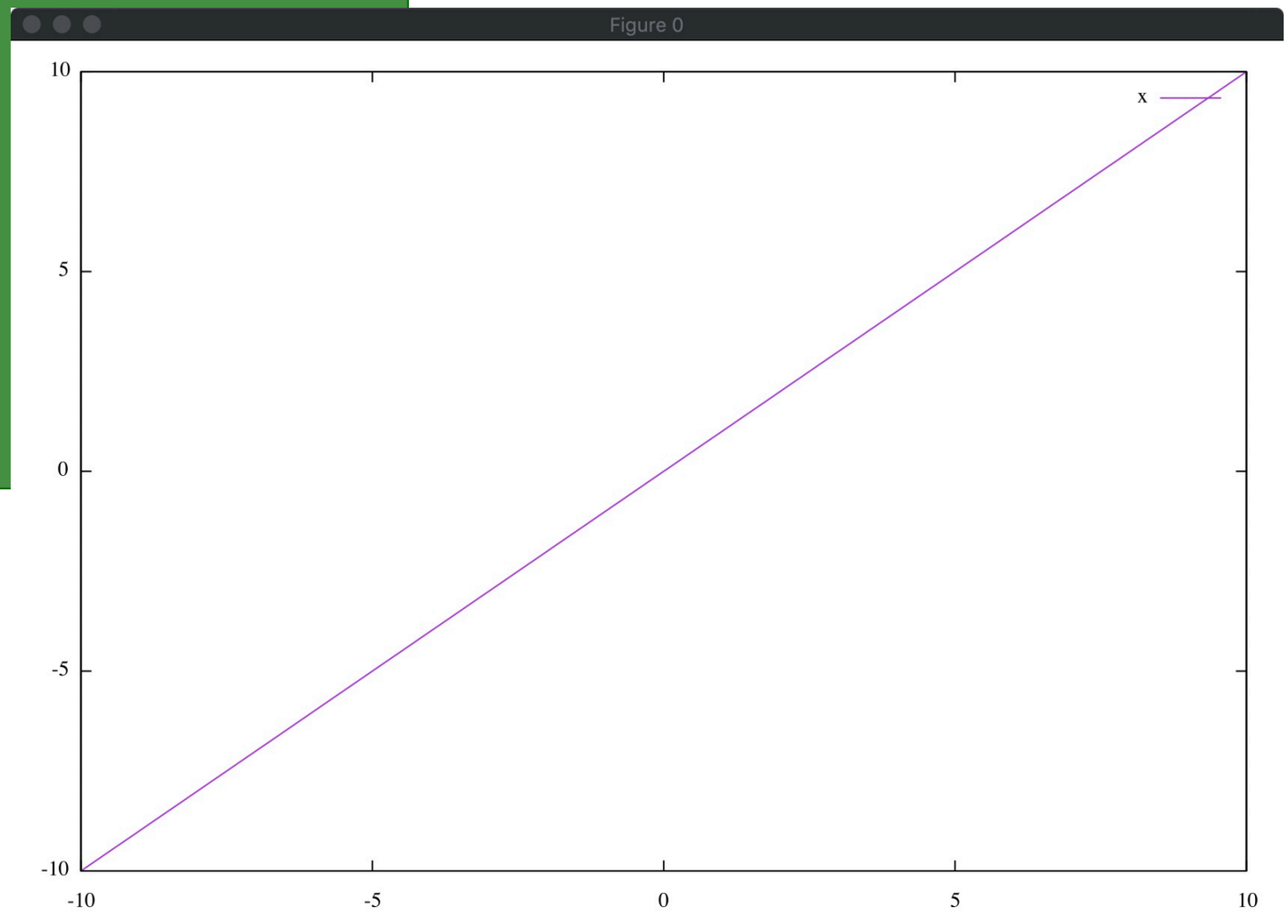
メモリ空間の連続性の向き→

お昼休憩の前に．．．

gnuplotが動くかチェック

masakazu — Terminal — gnuplot — 79×26

```
[~;gnuplot

        G N U P L O T
        Version 5.2 patchlevel 7    last modified 2019-05-29

        Copyright (C) 1986-1993, 1998, 2004, 2007-2018
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'aqua'
[gnuplot> plot x
gnuplot>
```

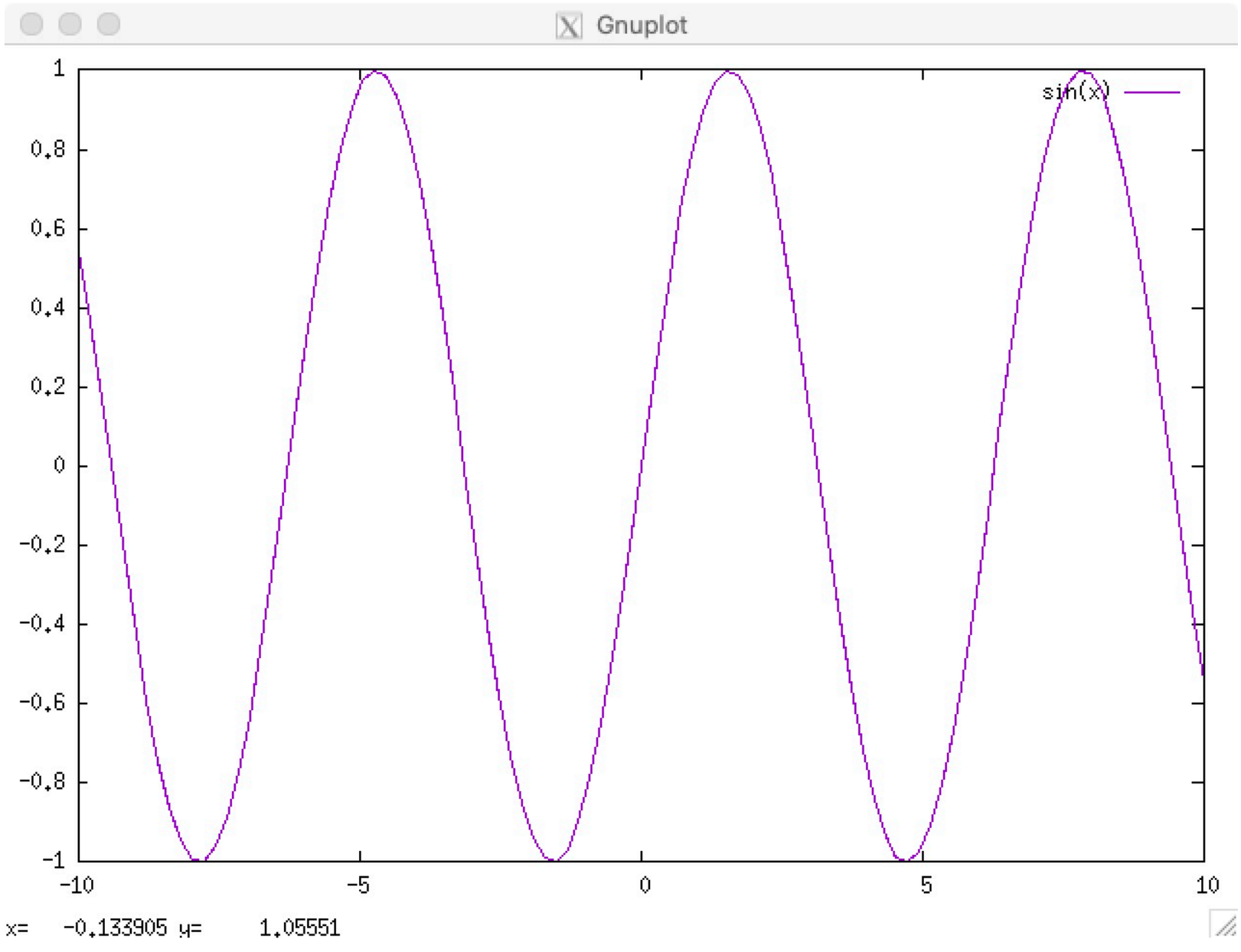


出ない人は挙手

gnuplotが動くかチェック

```
Common;cc 4_Gnuplot_3.c
Common;./a.out
Common;
```

出ない人は挙手

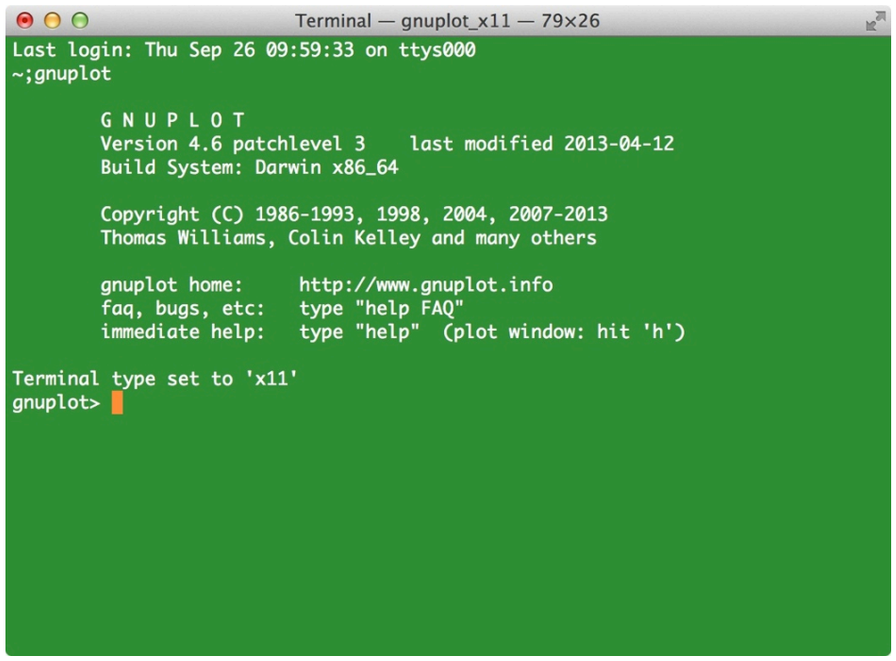


お昼休憩

～gnuplot編～

gnuplotに慣れよう

$: gnuplot　と打つ.



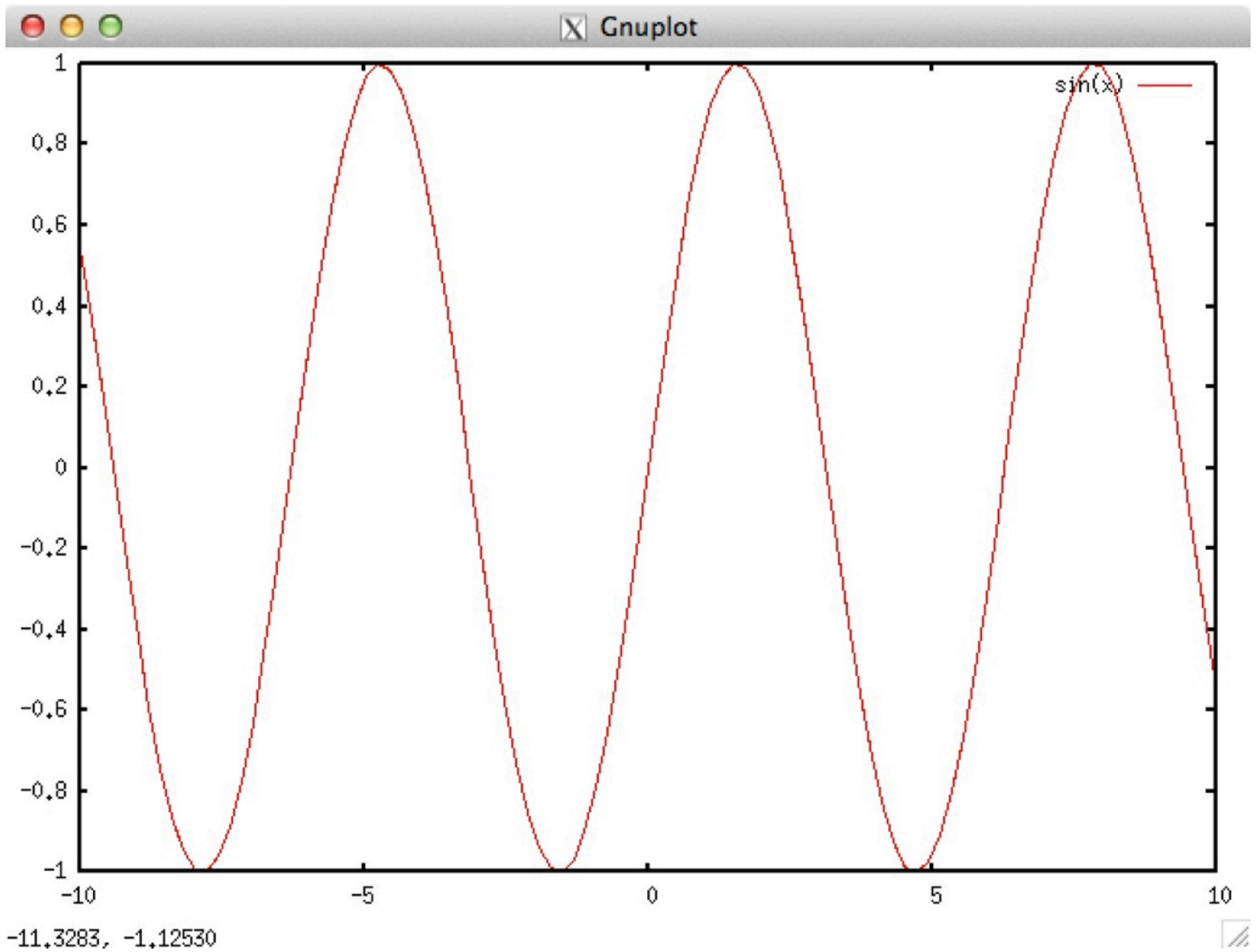
私の環境ではこう.　　違う環境ではこう.

細かいことは気にしないこと.

（緑の画面にすることは後で出来る.）

gnuplotに慣れよう

$: plot sin(x) と打つ.



こんな画面が出ればOK! ⇒　出ない人は挙手.

gnuplotに慣れよう

“ Exercise” が出たら手と頭を動かすこと（本演習での約束！！）

Exercise！. 以下のグラフを描いみてみよ.

$y = x^2$

$y = x^8$

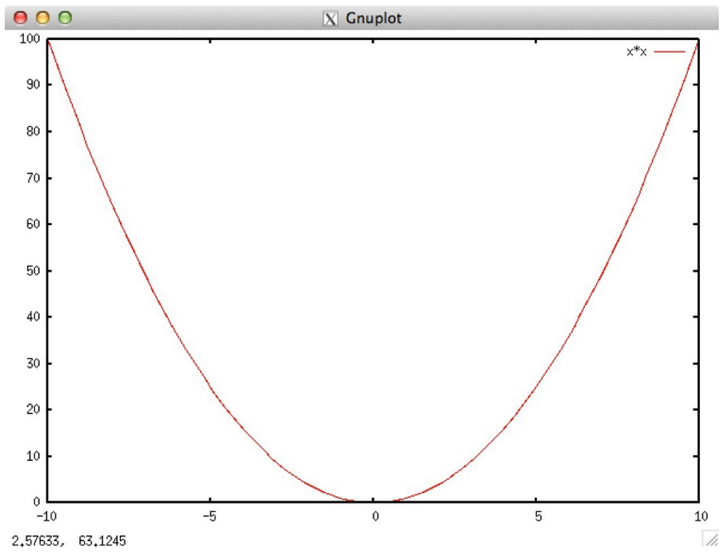
$y = e^x$

gnuplotに慣れよう

“ Exercise” が出たら手と頭を動かすこと（本演習での約束！！）

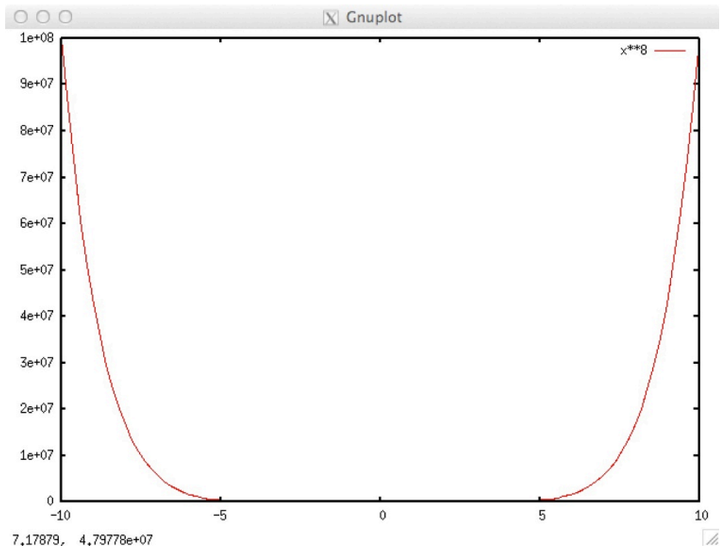
Exercise！. 以下のグラフを描いみてみよ.

$y = x^2$



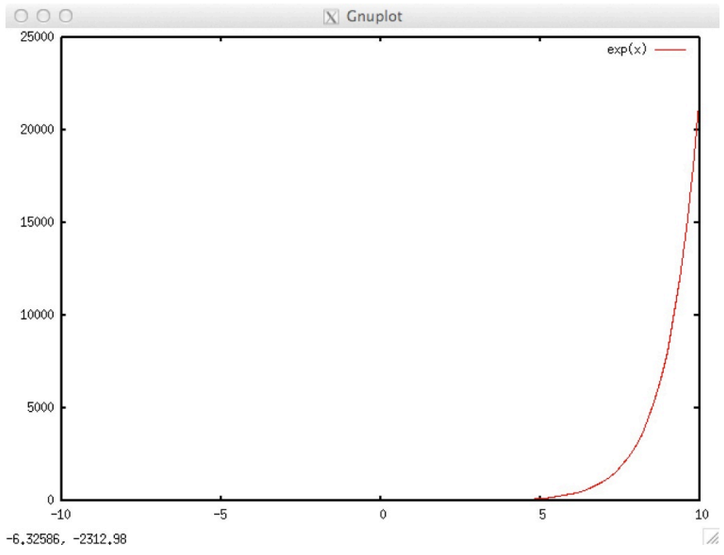
x*x

$y = x^8$



x**8

$y = e^x$

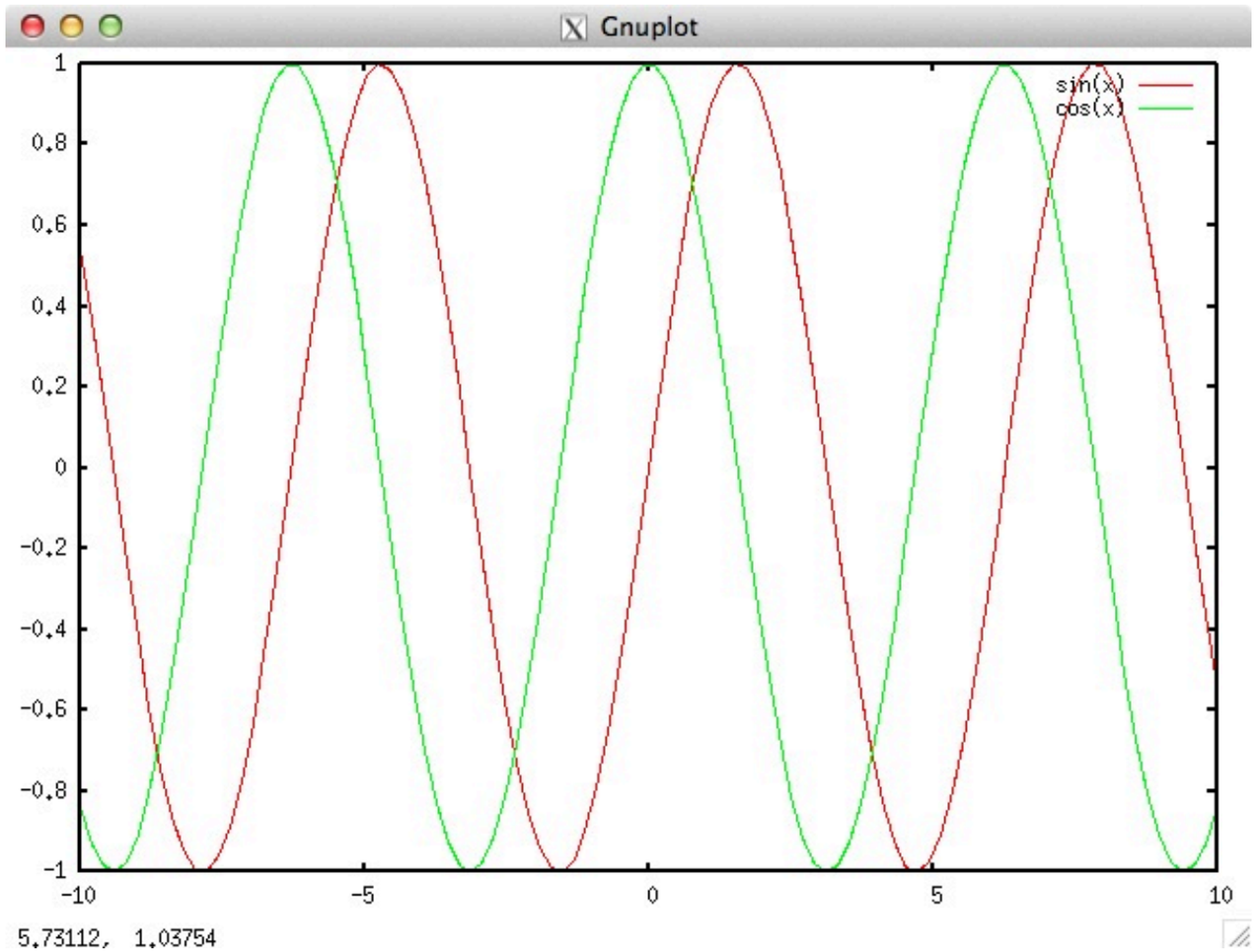


exp(x)

gnuplotに慣れよう

グラフの指定について

$: plot sin(x) , cos(x)　　カンマでつなぐと．．．

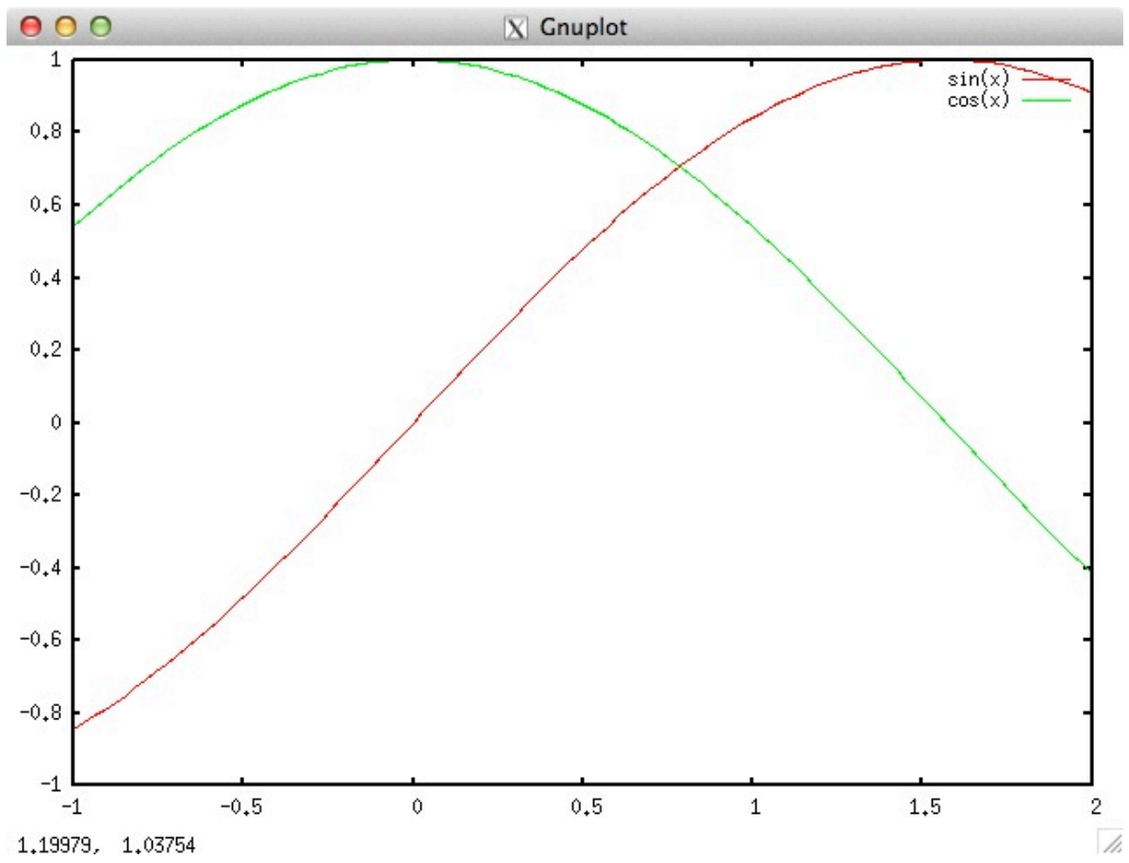


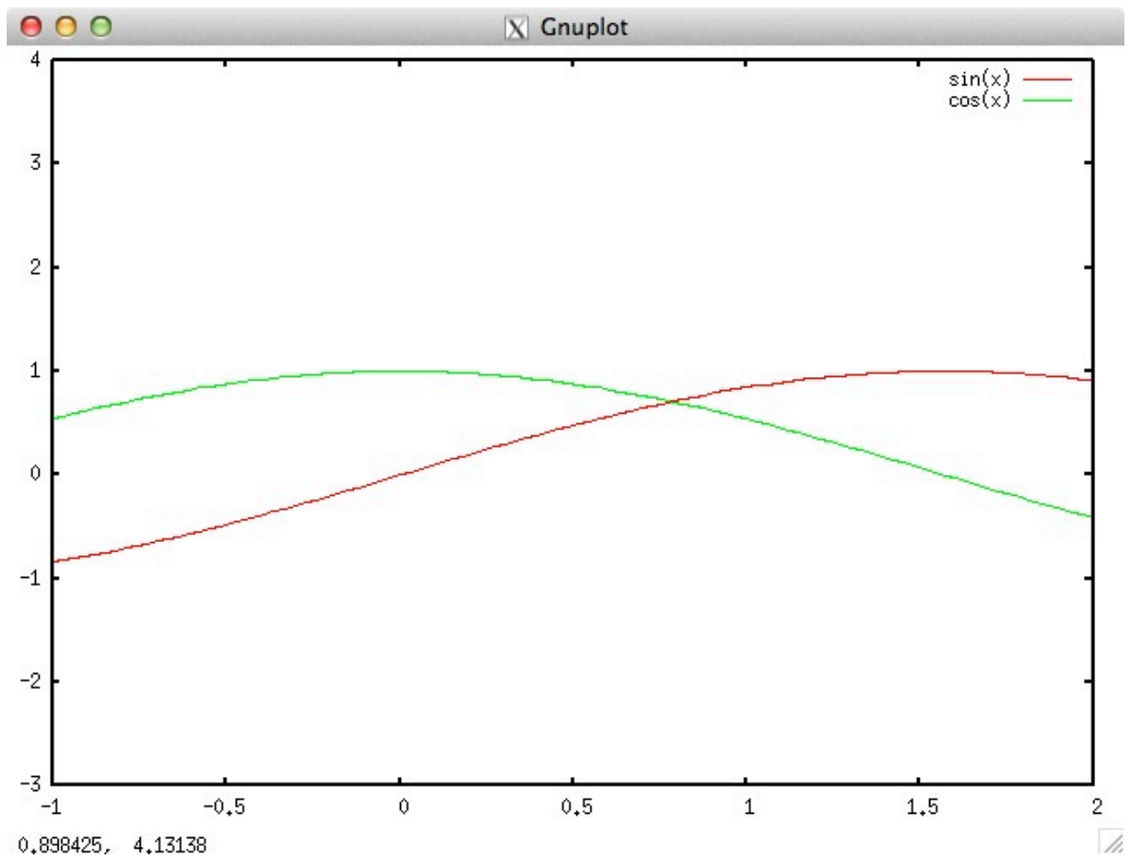
gnuplotに慣れよう

グラフの指定について

$: plot [-1:2] sin(x),cos(x)　　$x \in [-1, 2]$

$: plot [-1:2][-3:4] sin(x),cos(x)　　$x \in [-1, 2]\ y \in [-3, 4]$

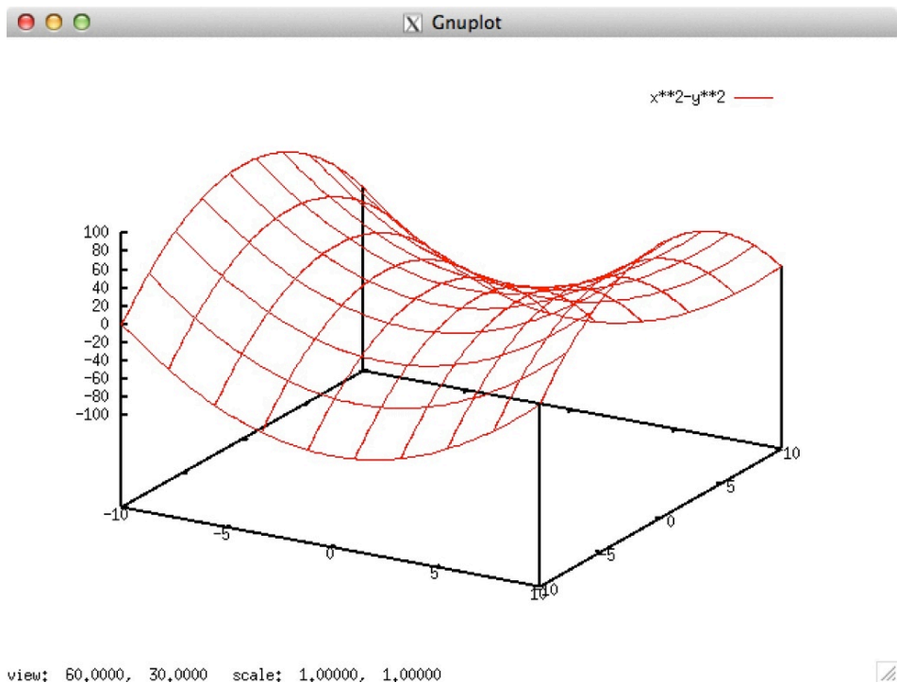


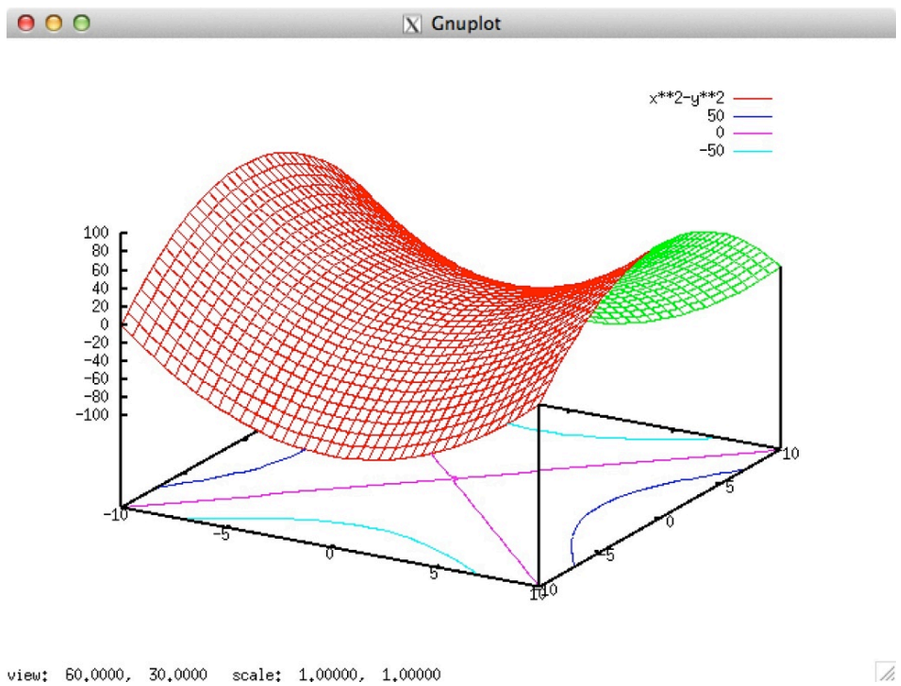


gnuplotに慣れよう

グラフの指定について

$: splot x**2-y**2





細かく指定することで，色々なグラフを描くことができる．

⇒　設定などはgoogleなどで検索をかける！

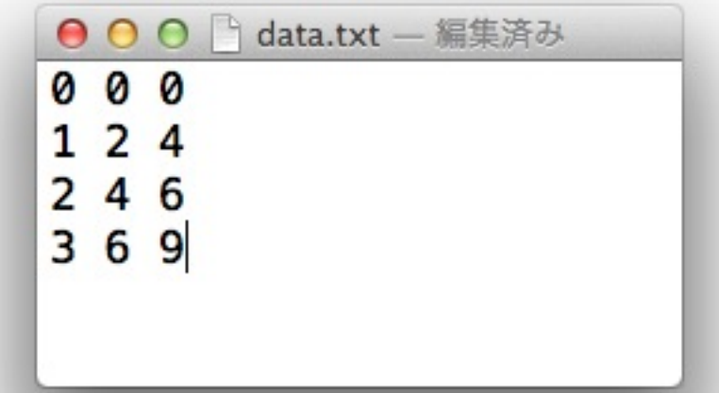
```
gnuplot> splot x**2-y**2
gnuplot> set hidden3d          ← 隠線消去を指定
gnuplot> set contour           ← 等高線描画を指定
gnuplot> set isosamples 40,40  ← メッシュを細かく
gnuplot> replot                ← 再描画
```

gnuplotに慣れよう

計算データの描画について

1．空のファイルを作成してファイル名をdata.txtにする.

2．下の例を参考に“x y1 y2”形式で手で入力する.

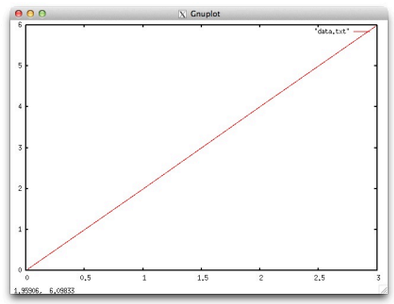


3．gnuplotで可視化する.　　$: plot "data.txt"

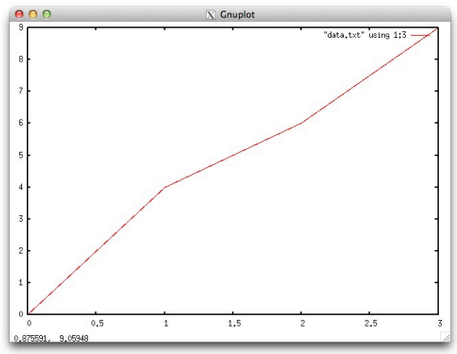
gnuplotに慣れよう

計算データの描画について

$: plot "data.txt" with lines　　⇒　折れ線でつなぐことができる.



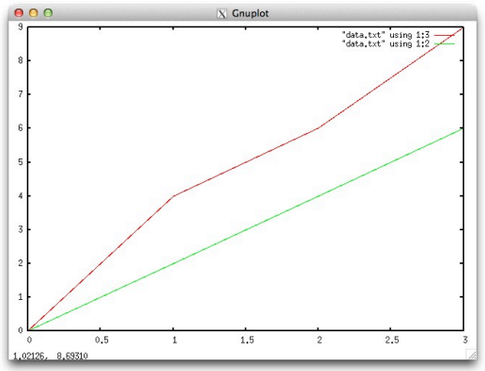
$: plot "data.txt" using 1:3 with lines



⇒　折れ線でつなぐことができる.

⇒　１列目と３列目のデータを
つかう.

$: plot "data.txt" using 1:3 with lines, "data.txt" using 1:2 with lines



⇒　２つのグラフを出すことができる.

gnuplotに慣れよう

計算データの描画について

Tips!!
長ったらしいコマンドは一文字に置き換えられる場合がある.

$: plot "data.txt" using 1:3 with lines, "data.txt" using 1:2 with lines

⇔

$: plot "data.txt" u 1:3 w l, "data.txt" u 1:2 w l

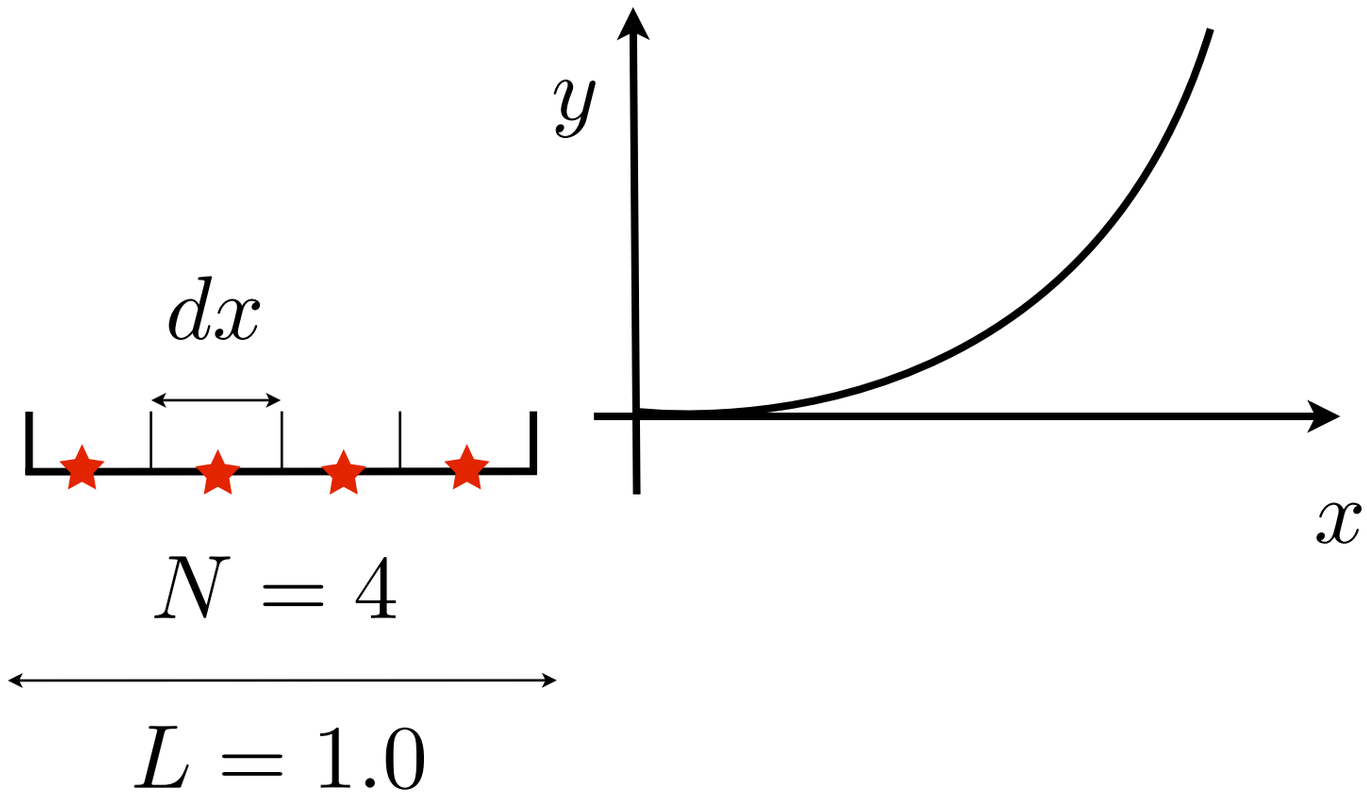
どちらでも好きな方で

gnuplotに慣れよう

計算データは手で作成するよりも計算機で計算したほうが楽.

4_Gnuplot_1.cをコンパイル＆実行

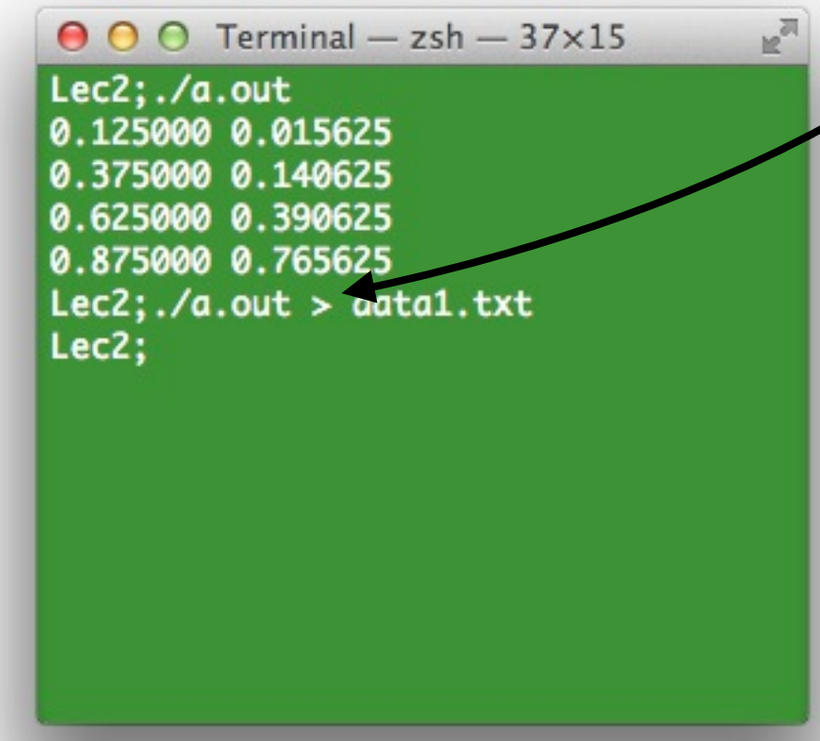
```c
#include <stdio.h>
int
main(void)
{
    double x,y;
    double L = 1.0;          //x length
    int    N = 4;            //Bunkatu
    double dx = L / N;       //kizami
    int    i;

    for(i = 0;i < N;i ++)
    {
        x = (i + 0.5) * dx;
        y = x * x;

        printf("%f %f\n",x,y);
    }

    return 0;
}
```

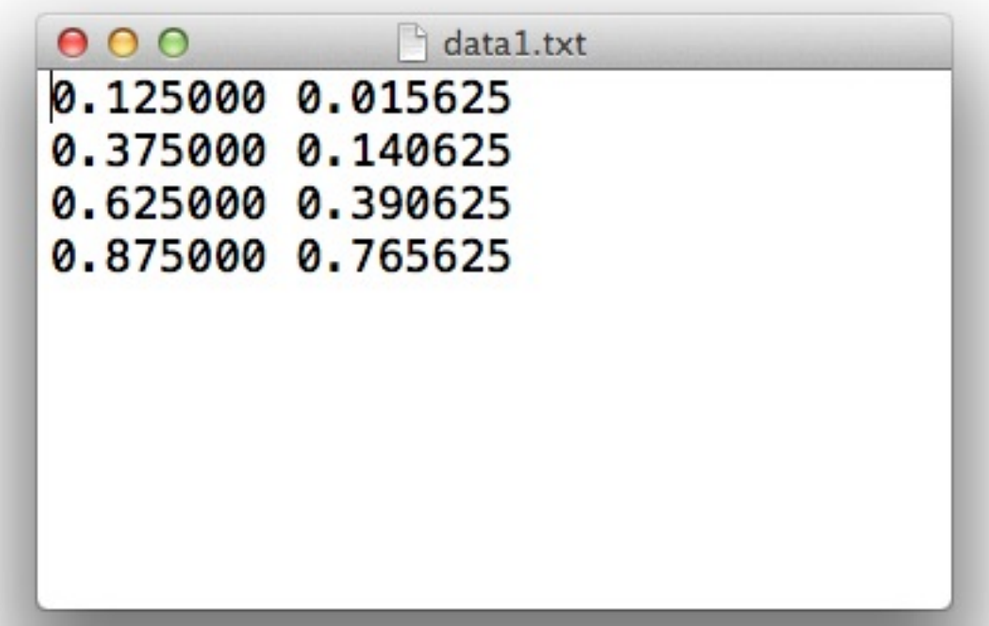


gnuplotに慣れよう

計算データは手で作成するよりも計算機で計算したほうが楽.

3．計算されたデータをリダイレクト“ > ”でテキストに保存.





4． gnuplotで可視化する.

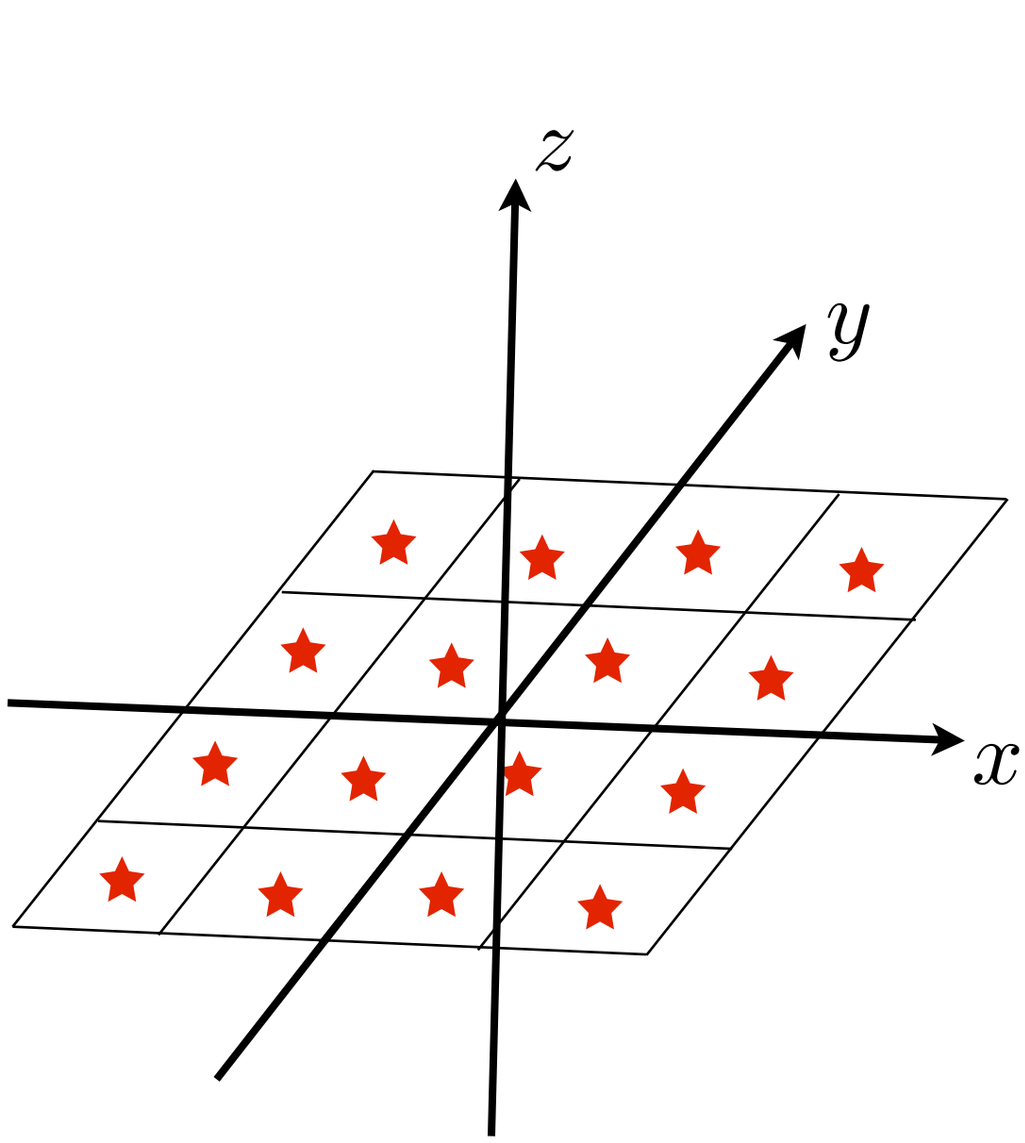
Exercise ！

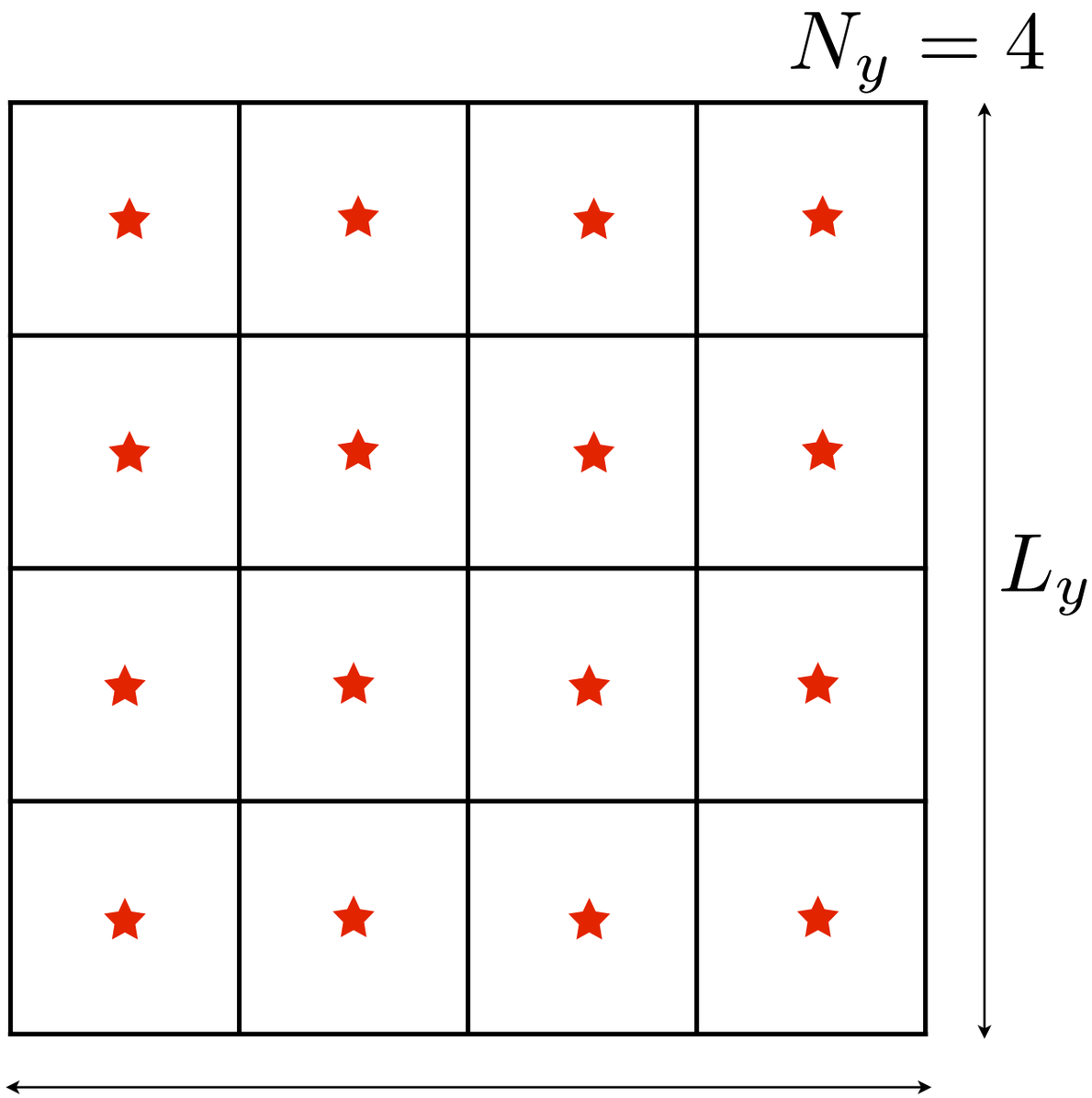
Nの値やLの値を変更し，可視化せよ.

gnuplotに慣れよう

２変数関数ならどのようにやるのか？

4_Gnuplot_2.cをコンパイル＆実行する.

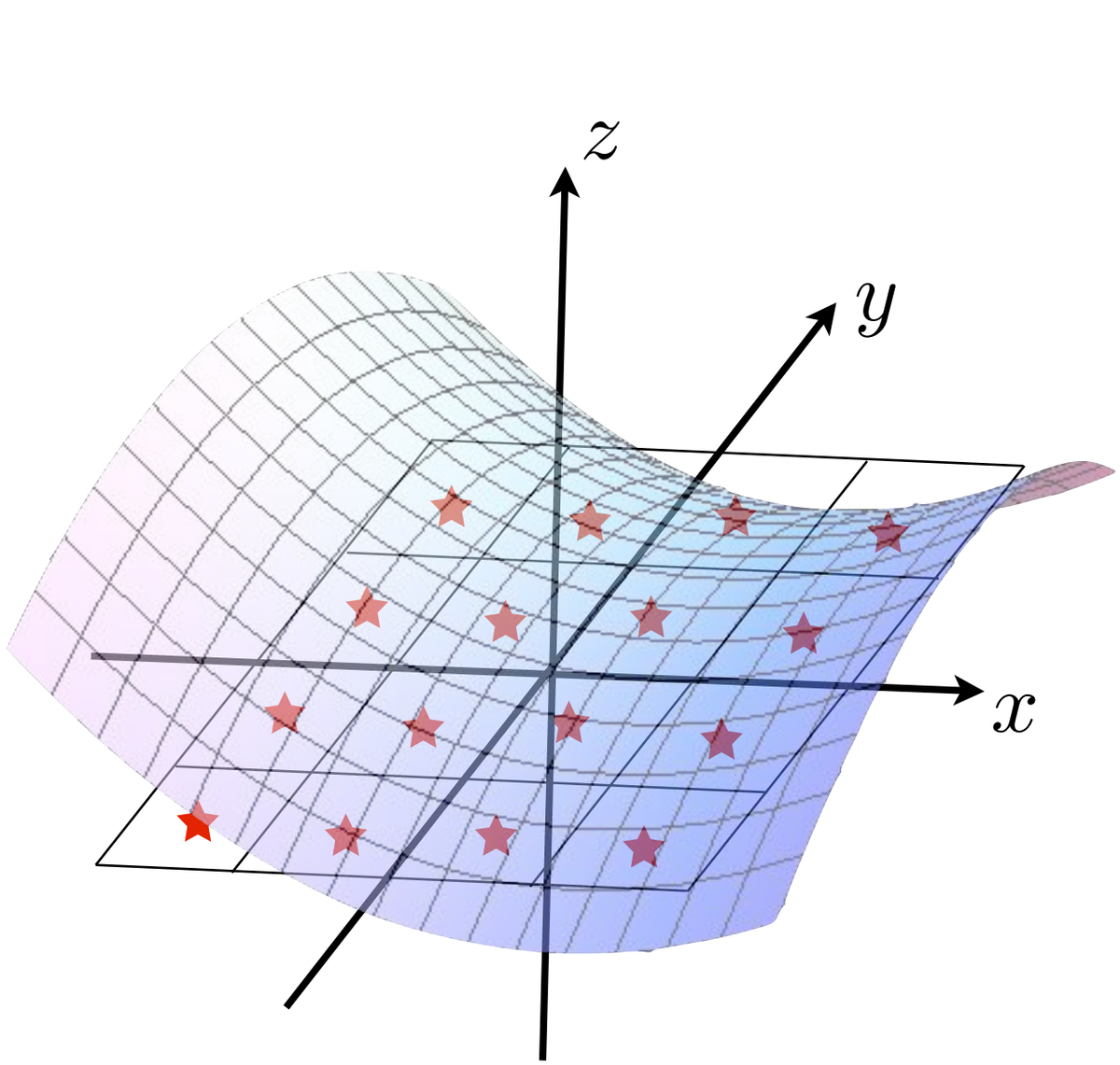


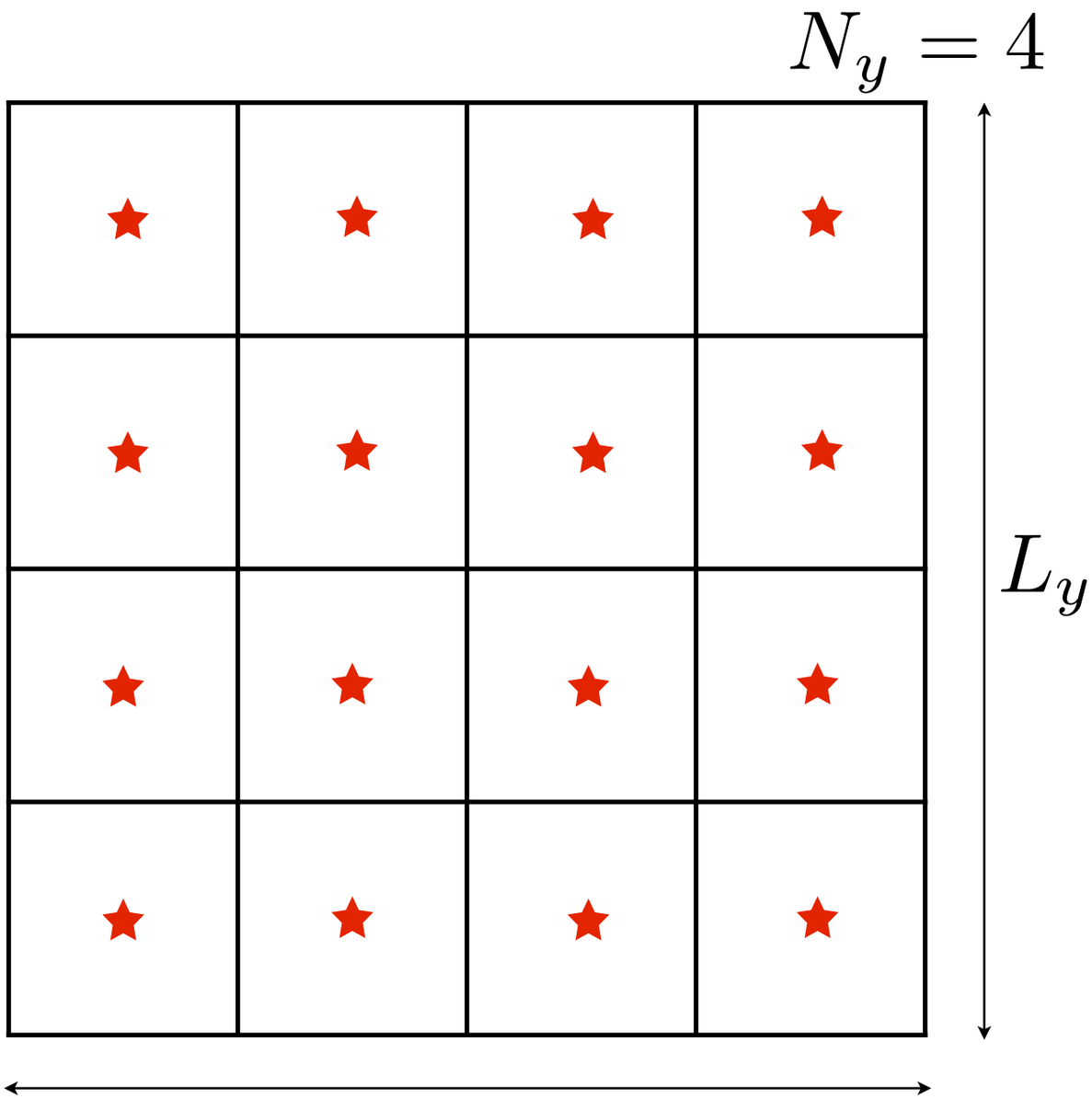


gnuplotに慣れよう

２変数関数ならどのようにやるのか？

4_Gnuplot_2.cをコンパイル＆実行する.

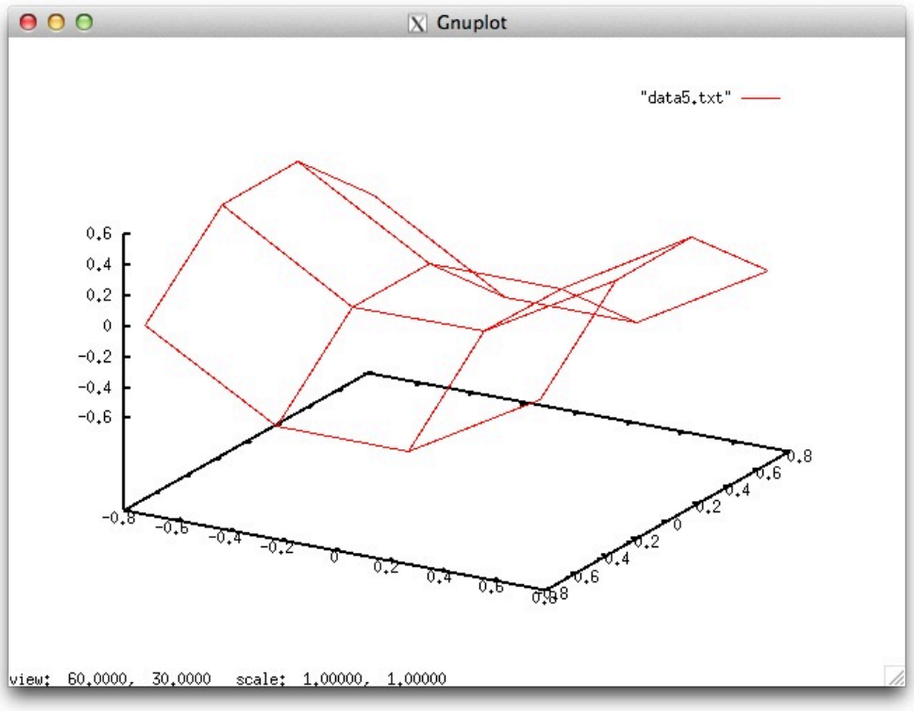




gnuplotに慣れよう

```c
#include <stdio.h>
#include <math.h>

int
main(void)
{
    double x,y,z;
    double Lx = 2.0;        //x length
    double Ly = 2.0;        //y length
    int    Nx = 4;          //x Bunkatu
    int    Ny = 4;          //y Bunkatu
    double dx = Lx / Nx;    //x kizami
    double dy = Ly / Ny;    //y kizami

    for(int i = 0;i < Nx;i ++)
    {
        for(int j = 0;j < Ny;j ++)
        {
            x = (i + 0.5) * dx - Lx / 2;
            y = (j + 0.5) * dy - Ly / 2;
            z = x * x - y * y;
            printf("%f %f %f \n",x,y,z);
        }
        printf("\n");
    }
    return 0;
}
```

4_Gnuplot_2.c

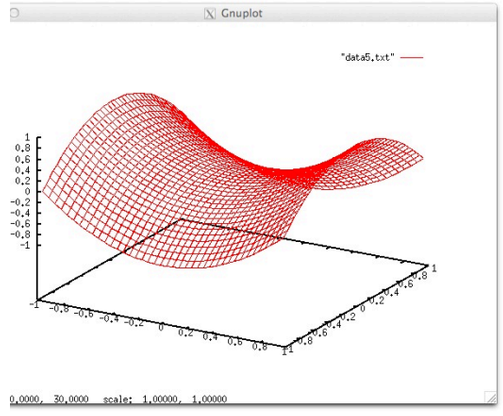


gnuplot 4x4

data5.txt

```
-0.750000 -0.750000 0.000000
-0.750000 -0.250000 0.500000
-0.750000 0.250000 0.500000
-0.750000 0.750000 0.000000

-0.250000 -0.750000 -0.500000
-0.250000 -0.250000 0.000000
-0.250000 0.250000 0.000000
-0.250000 0.750000 -0.500000

0.250000 -0.750000 -0.500000
0.250000 -0.250000 0.000000
0.250000 0.250000 0.000000
0.250000 0.750000 -0.500000

0.750000 -0.750000 0.000000
0.750000 -0.250000 0.500000
0.750000 0.250000 0.500000
0.750000 0.750000 0.000000
```

計算データ



gnuplot 40x40

gnuplotに慣れよう

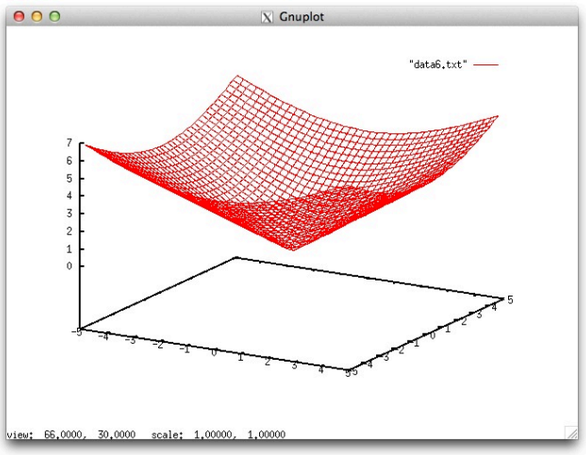
Exercise！. 以下のグラフを描いみてみよ.

$z = \sqrt{x^2 + y^2}$　　$z = xy$　　$z = \cos\sqrt{x^2 + y^2}$

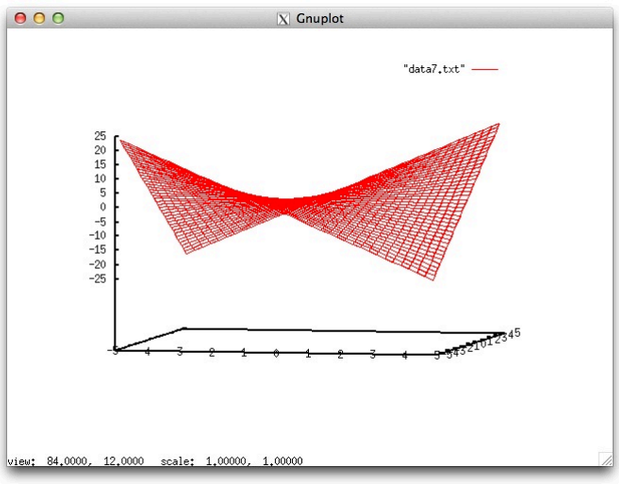
gnuplotに慣れよう

Exercise！. 以下のグラフを描いみてみよ.

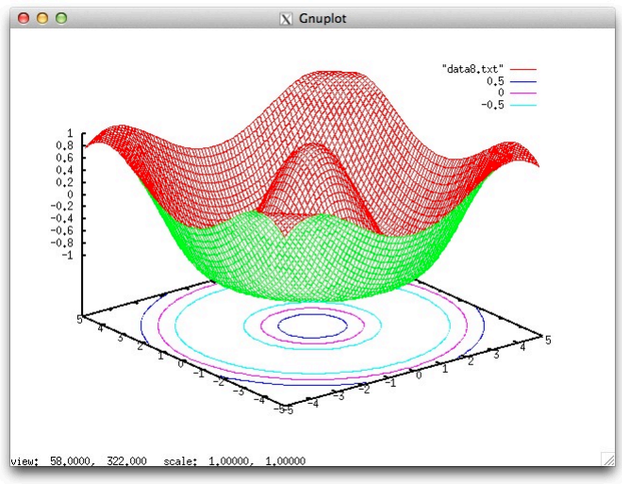
$z = \sqrt{x^2 + y^2}$



$z = xy$



$z = \cos\sqrt{x^2 + y^2}$



gnuplotに慣れよう

空間変数“x”時間変数“ t ”を含むデータの可視化.

$$z = f(x, y) \quad \Rightarrow \quad z = f(x, t)$$

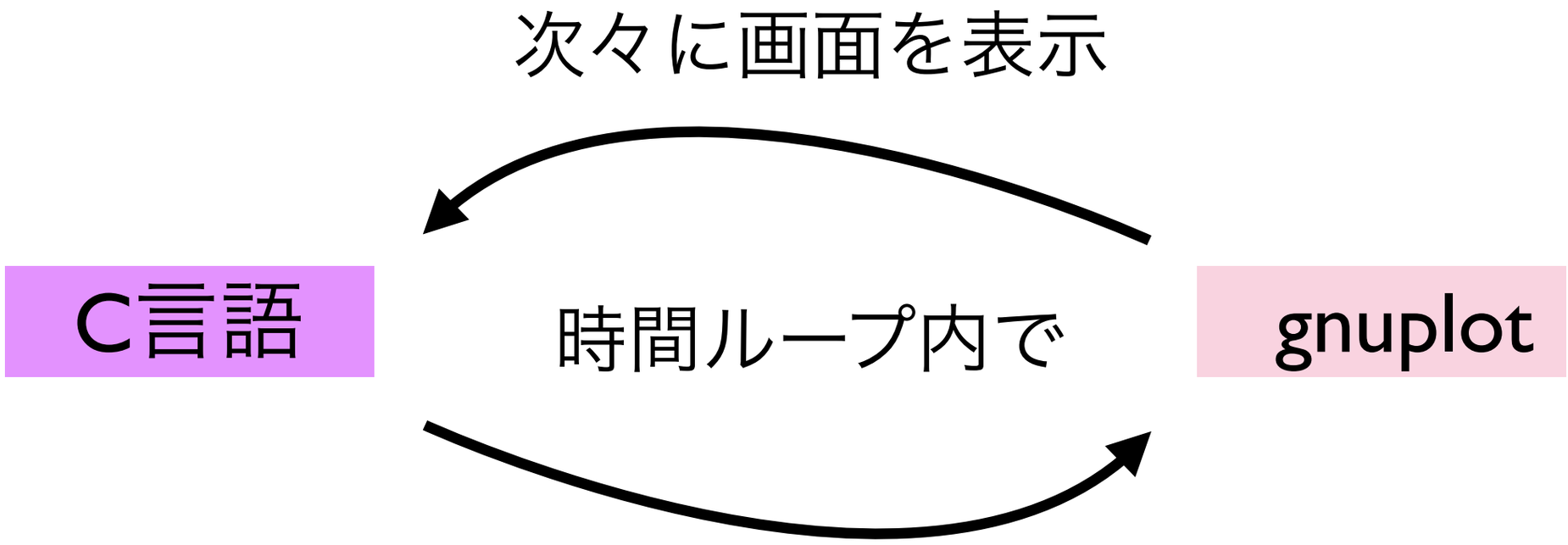
これだと芸がない. . . .

gnuplotに慣れよう

空間変数“x”時間変数“ t ”を含むデータの可視化.

$$z = f(x, y) \quad \Rightarrow \quad z = f(x, t)$$

これだと芸がない. . . .



gnuplotに慣れよう

C言語から直接制御

```c
#include <stdio.h>

int
main(void)
{
    FILE *gp;//For gnuplot

    gp = popen("gnuplot -persist","w");
    fprintf(gp, "set terminal x11\n");
    fprintf(gp, "plot sin(x)\n");
    fflush(gp);
    pclose(gp);

    return 0;
}
```

この構文がミソ！

4_Gnuplot_3.c

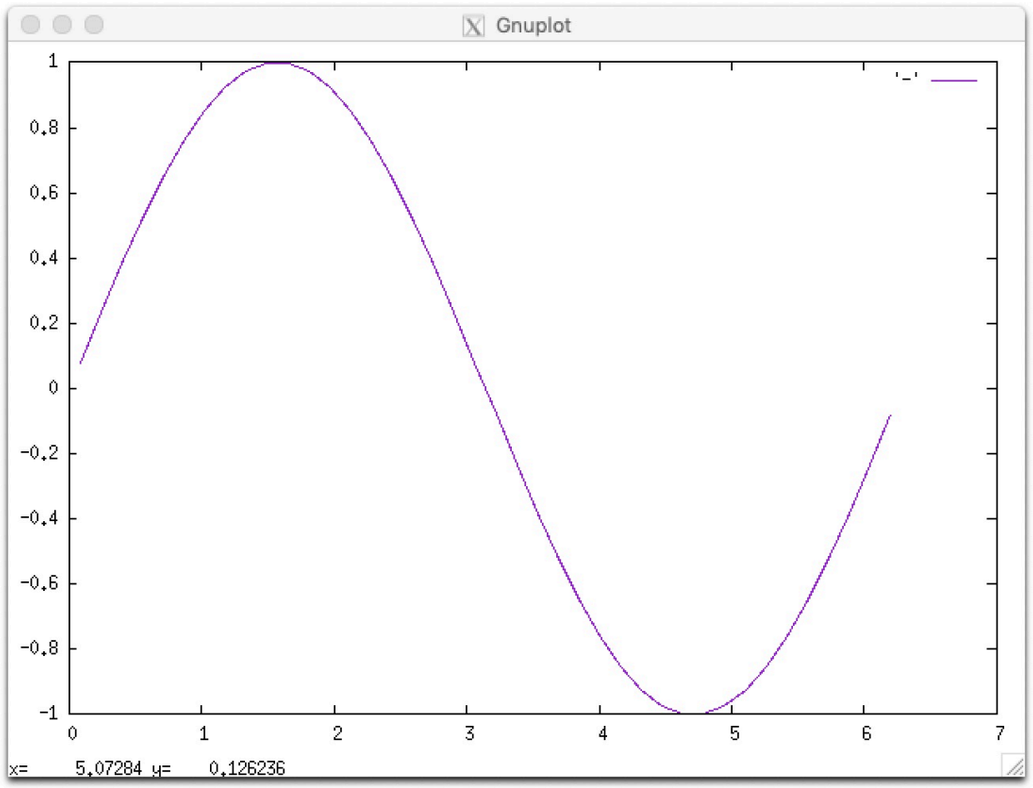
gnuplotに慣れよう

C言語から直接制御

```c
#include <stdio.h>
#include <math.h>

int
main(void)
{
    int i;
    int N = 40;
    double x,y;
    double L = 2 * 3.14159265358979;
    double dx = L / N;

    FILE *gp;//For gnuplot
    gp = popen("gnuplot -persist","w");

    fprintf(gp, "plot '-' with lines \n");//これがミソ

    for(i = 0;i < N;i ++)
    {
        x = (i + 0.5) * dx;
        y = sin(x);
        fprintf(gp,"%f %f\n", x, y);// データの書き込み
    }
    fprintf(gp,"e\n"); // データの書き込み終了

    fflush(gp);
    pclose(gp);

    return 0;
}
```

データを次々と計算し，

それをgnuplotに食わせる．



4_Gnuplot_4.c

gnuplotに慣れよう

C言語から直接制御

```c
#include <stdio.h>
#include <math.h>

int
main(void)
{
    int i;
    int i_time;
    int N = 40;
    double x,y;
    double L = 4 * 3.14159265358979;
    double dx = L / N;
    double dt = 0.05;

    FILE *gp;//For gnuplot
    gp = popen("gnuplot -persist","w");
     fprintf(gp, "set terminal x11\n");

    for(i_time = 0;;i_time ++)
    {
        fprintf(gp, "set yrange [-1.2:1.2]\n");
        fprintf(gp, "plot '-' with lines \n");
        for(i = 0;i < N;i ++)
        {
            x = (i + 0.5) * dx;
            y = sin(x - i_time * dt);
            fprintf(gp,"%f %f\n", x, y);
        }
        fprintf(gp,"e\n");
          fflush(gp);
    }
    pclose(gp);

    return 0;
}
```

4_Gnuplot_5.c

“データを次々と計算し,
それをgnuplotに食わせる. ”

を時間ループ内で繰り返す

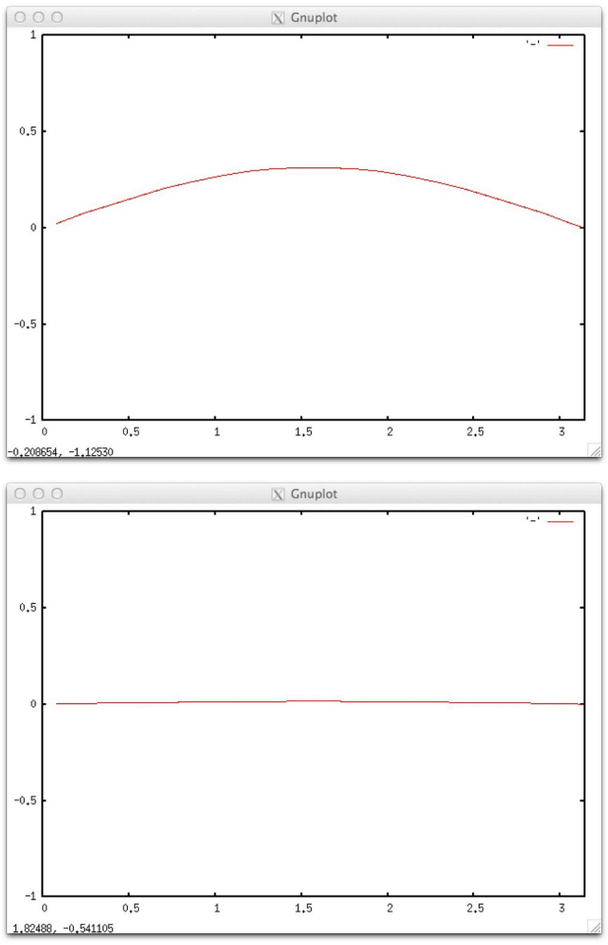
Tips!
無限ループを止めるには

“Ctrl + C”

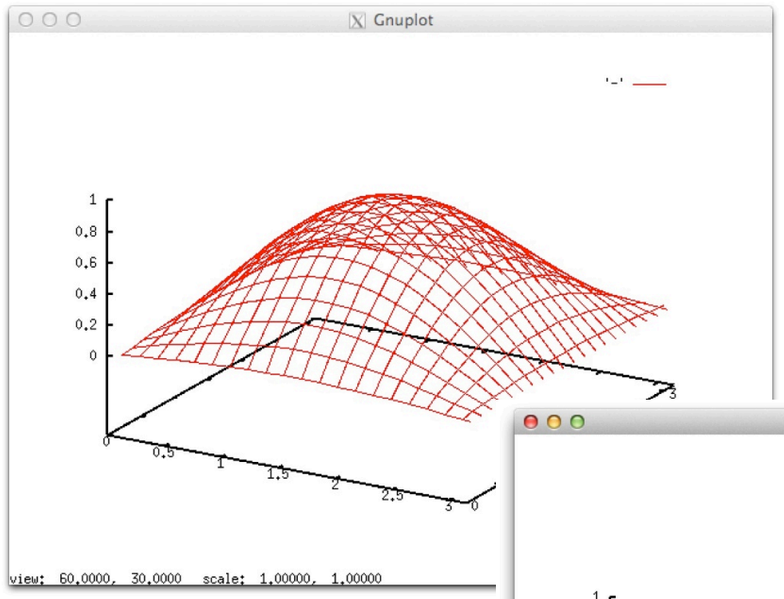
gnuplotに慣れよう

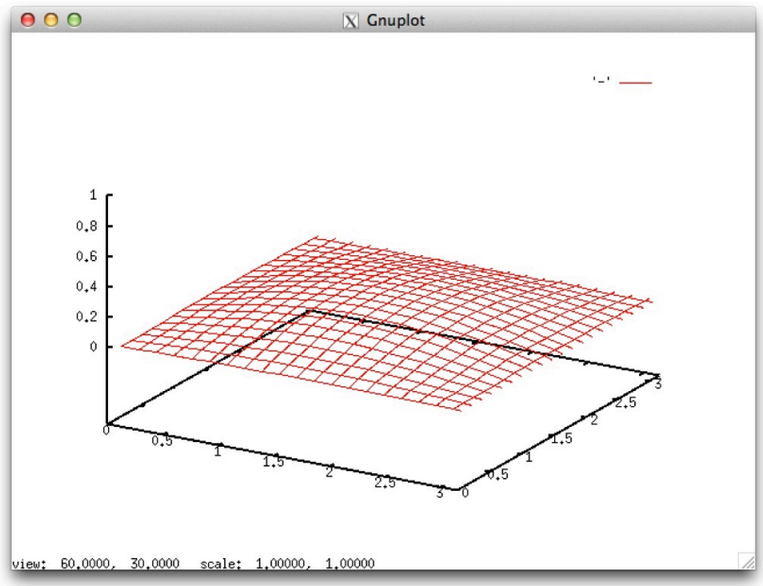
Exercise！. 以下のグラフを描いてみよ.

$$z = e^{-t} \sin x$$



$$z = e^{-t} \sin x \sin y$$





常微分方程式の
数値解法入門

常微分方程式の数値解法

Euler法とは

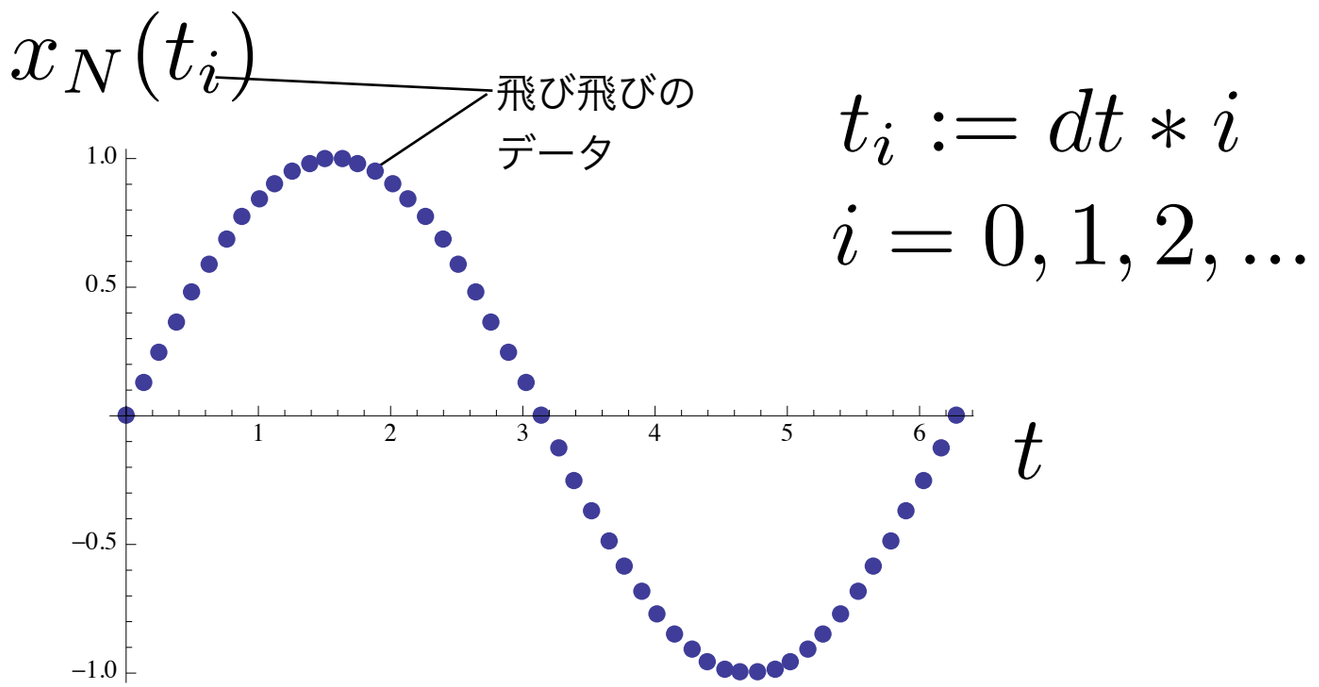
$$\dot{x}(t) = \cos(t)$$

$$x(0) = 0$$

問題

$$x_s(t) = \sin(t)$$

厳密解



$$t_i := dt * i$$

$$i = 0, 1, 2, ...$$

数値解

$x_N(t_i)$　が満たすべき条件を決定する.

常微分方程式の数値解法

Euler法とは

$x_N(t_i)$　が満たすべき条件を決定する.

この変形が
Euler法の心

$$\dot{x}(t) = \cos(t)$$

$$x(0) = 0$$

問題

常微分方程式の数値解法

Euler法とは

$x_N(t_i)$　が満たすべき条件を決定する.

この変形が
Euler法の心

$$\dot{x}(t) = \cos(t)$$

$$x(0) = 0$$

問題

$$\frac{x_N(t_{i+1}) - x_N(t_i)}{dt} \approx \cos(t_i)$$

$$x_N(t_0) = x(0) = 0$$

常微分方程式の数値解法

Euler法とは

$x_N(t_i)$　が満たすべき条件を決定する.

この変形が
Euler法の心

$$\dot{x}(t) = \cos(t)$$

$$x(0) = 0$$

問題

$$\frac{x_N(t_{i+1}) - x_N(t_i)}{dt} \approx \cos(t_i)$$

$$x_N(t_0) = x(0) = 0$$

$$f'(t) := \lim_{\delta t \to 0} \frac{f(t+\delta t) - f(t)}{\delta t}$$

常微分方程式の数値解法

Euler法とは

$x_N(t_i)$　が満たすべき条件を決定する.

この変形が
Euler法の心

$$\dot{x}(t) = \cos(t)$$

$$x(0) = 0$$

問題

$$\frac{x_N(t_{i+1}) - x_N(t_i)}{dt} \approx \cos(t_i)$$

$$x_N(t_0) = x(0) = 0$$

$$f'(t) := \lim_{\delta t \to 0} \frac{f(t+\delta t) - f(t)}{\delta t}$$

$$x_N(t_{i+1}) \approx x_N(t_i) + dt\cos(t_i)$$

現時点の　$x_N(t_i)$　がわかればちょっと先の $x_N(t_{i+1})$も大体わかる.

常微分方程式の数値解法

Euler法とは

$$x_N(t_{i+1}) \approx x_N(t_i) + dt\cos(t_i)$$

$$x_N(t_0) = 0$$

```c
#include <stdio.h>
#include <math.h>

int
main(void)
{
    int i;
    double dt = 0.01;
    double xN,xN_new;

    FILE *gp;
    gp = popen("gnuplot -persist","w");
    fprintf(gp, "set terminal x11\n");
    fprintf(gp, "set yrange[-1.1:1.1] \n");
    fprintf(gp, "plot '-' with lines \n");

    xN = 0;
    for (i = 0; i < 1000; i++)
    {
        fprintf(gp,"%f %f\n", i * dt, xN);// データの書き込み
        xN_new = xN + dt * cos(i * dt);//計算
        xN = xN_new;
    }
    fprintf(gp,"e\n");

    fflush(gp);
    pclose(gp);

    return 0;
}
```

小さな数を設定.

gnuplotのおまじない.

初期値設定.

まず表示

漸化式

次のループのため更新

gnuplotのおまじない.

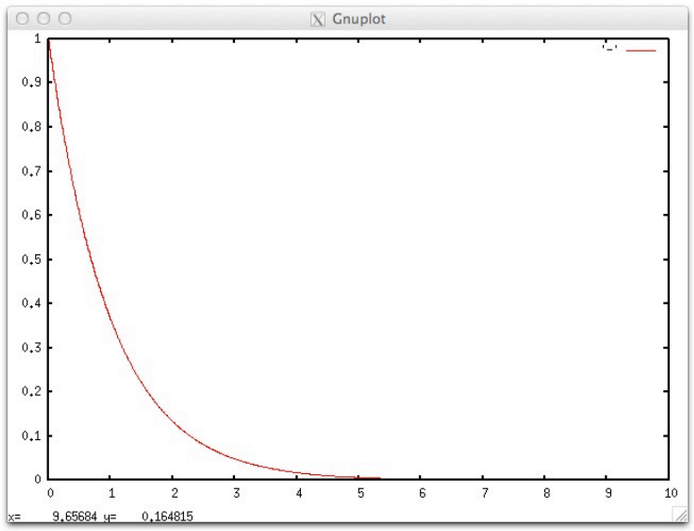
コンパイル＆実行しよう

5_ODE_1.c

常微分方程式の数値解法

Exercise！計算＆可視化をせよ．また厳密解と比較せよ．

$$\dot{x}(t) = -x(t)$$
$$x(0) = 1$$
$$(x_s(t) = e^{-t})$$



5_ODE_2.c

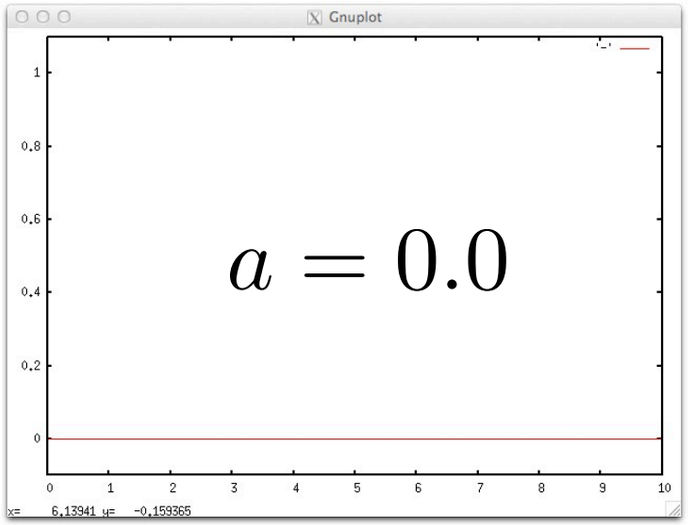
$$\dot{x}(t) = x(t)$$
$$x(0) = 0.01$$
$$(x_s(t) = 0.01e^{t})$$

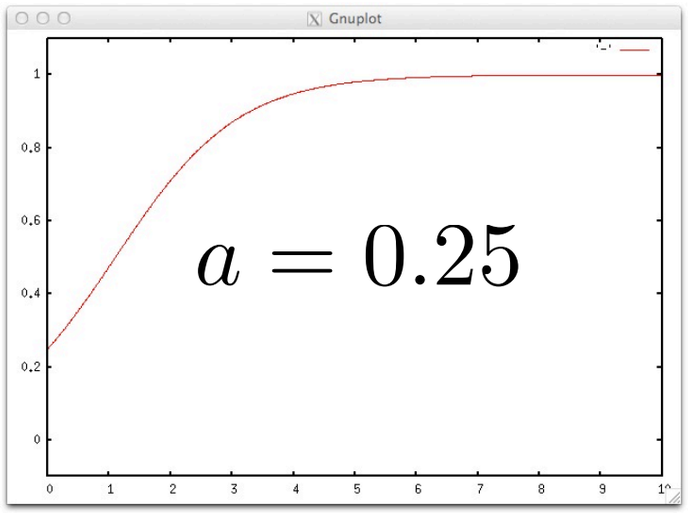
5_ODE_3.c

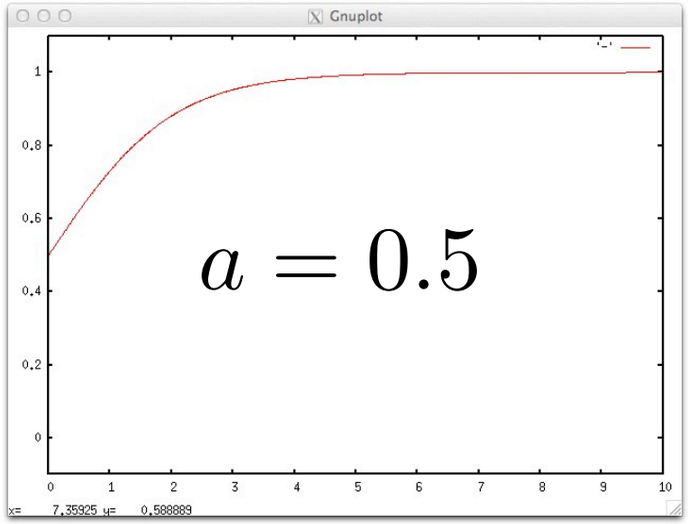
$$\dot{x}(t) = x(t)(1 - x(t))$$
$$x(0) = a \quad 0 \leq a \leq 1$$
$$(x_s(t) = ae^{t}/(1 - a + ae^{t}))$$

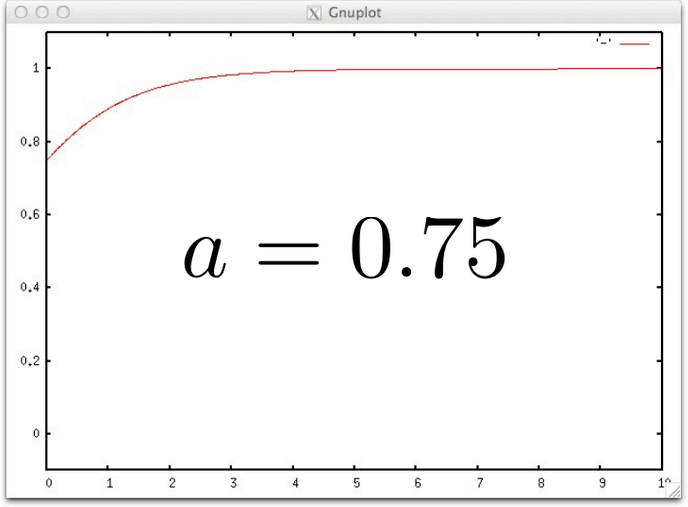
5_ODE_4.c

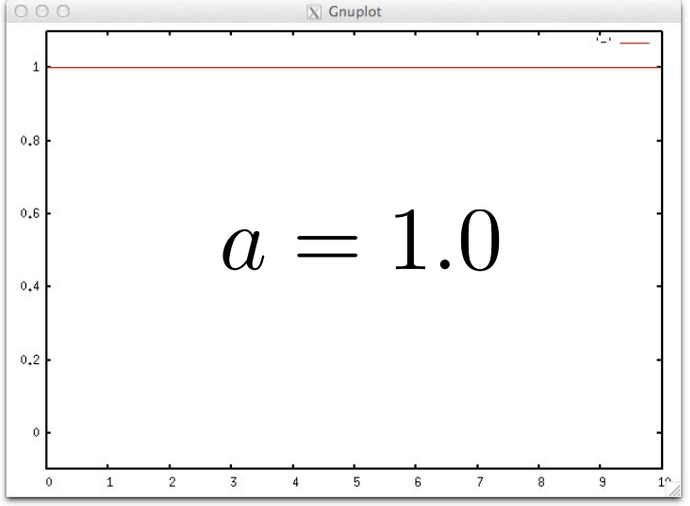
常微分方程式の数値解法











$$\dot{x}(t) = x(t)(1 - x(t))$$

$$x(0) = a$$

$$(x_s(t) = ae^t/(1 - a + ae^t))$$

a=0のとき以外はいつも1に向かう.なぜだろう?

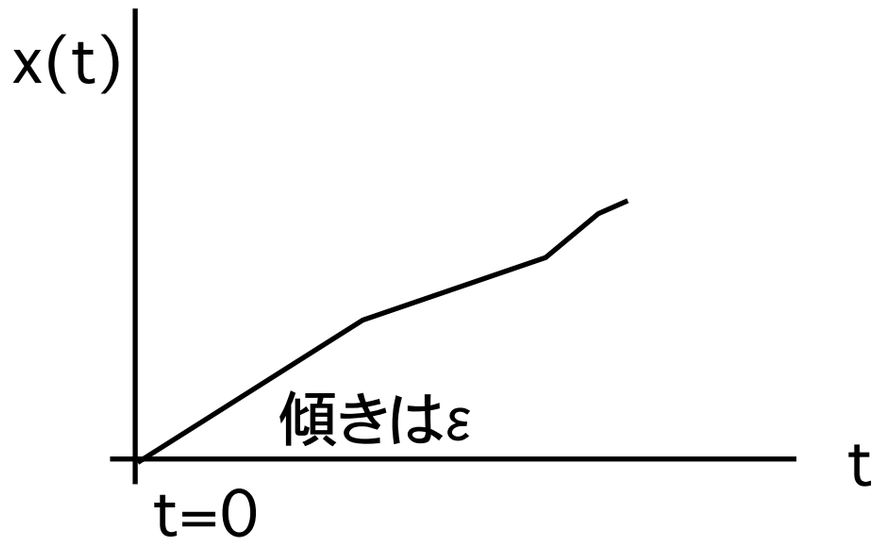
なぜだろう？

$$\dot{x}(t) = x(t)(1 - x(t))$$

$x(0) = \epsilon$　と置く.ただし　$\epsilon > 0$ の小さい数とする.

$t = 0$　を代入する.

$$\dot{x}(0) = \epsilon(1 - \epsilon) = \epsilon - \epsilon^2 \simeq \epsilon > 0$$



つまり時刻t=0での傾きは正なので,ちょっと先の未来では,少なくとも値は増える.

つまり, a=0から離れていく!

小さの勝負表

	ε	ε^2
ε=0.1	0.1	0.01
ε=0.01	0.01	0.0001
ε=0.001	0.001	0.000001
ε=0.0001	0.0001	0.00000001

一兆円(10^12)の前では100万円(10^6)はゴミでしょ?

常微分方程式の数値解法

多変数の場合

$$\dot{x}(t) = -y(t)$$

$$\dot{y}(t) = x(t)$$

$$x(0) = 1$$

$$y(0) = 0$$

問題

離散化

$$\frac{x_N(t_{i+1}) - x_N(t_i)}{dt} = -y_N(t_i)$$

$$\frac{y_N(t_{i+1}) - y_N(t_i)}{dt} = x_N(t_i)$$

$$x_N(t_0) = 1 \qquad y_N(t_0) = 0$$

C Code

```c
#include <stdio.h>
#include <math.h>

int
main(void)
{
    int i;
    double dt = 0.01;
    double xN,xN_new;
    double yN,yN_new;

    xN = 1.0;
    yN = 0.0;
    for (i = 0; i < 1000; i++)
    {
        printf("%f %f %f\n",i * dt, xN, yN);
        xN_new = xN + dt * (-yN);
        yN_new = yN + dt * (xN);
        xN = xN_new;
        yN = yN_new;
    }

    return 0;
}
```

5_ODE_5.c

常微分方程式の数値解法

$$\dot{x}(t) = -y(t)$$

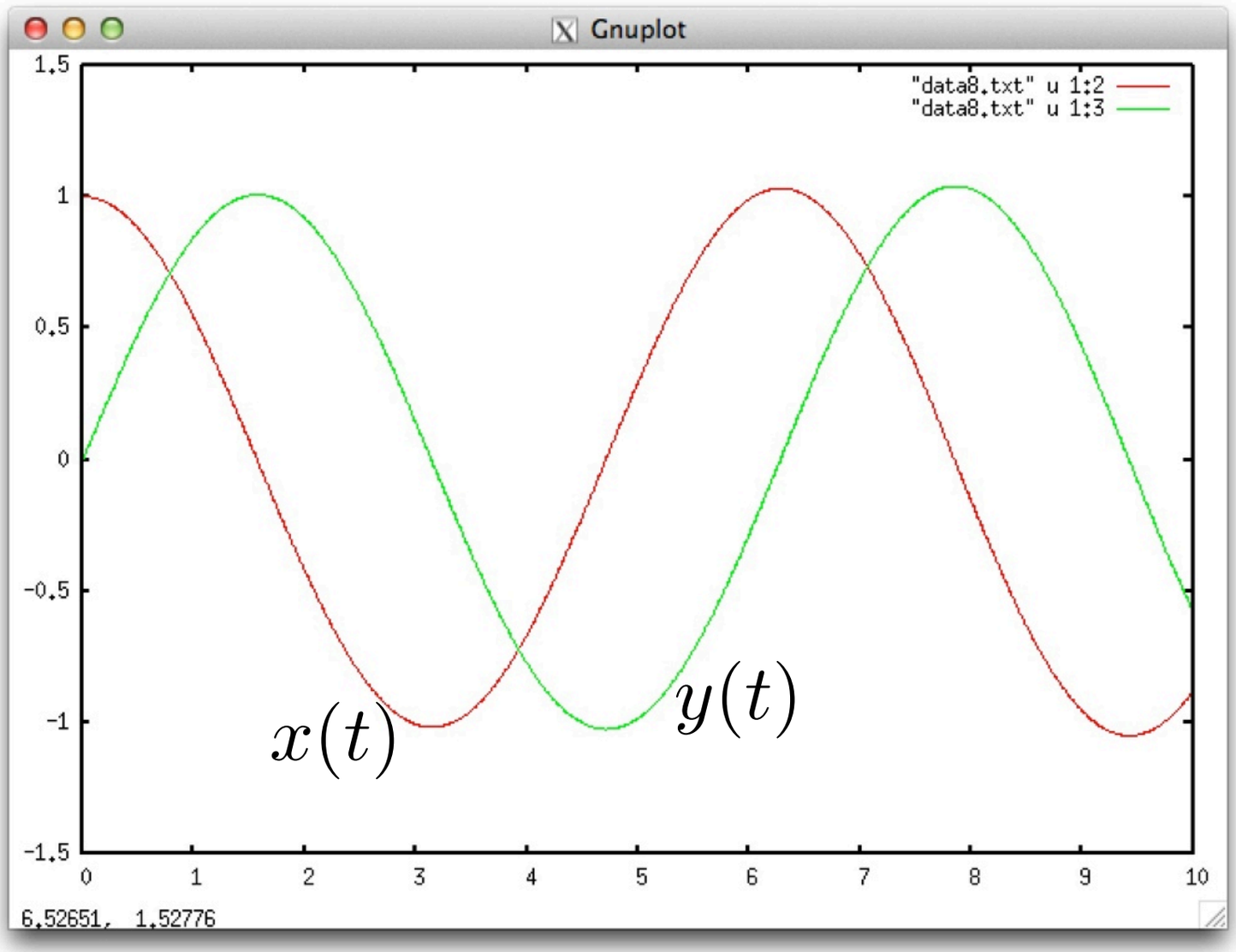
$$\dot{y}(t) = x(t)$$

$$x(0) = 1$$

$$y(0) = 0$$

問題

多変数の場合



常微分方程式の数値解法

誤差について

$$\dot{x}(t) = -y(t)$$
$$\dot{y}(t) = x(t)$$
$$x(0) = 1$$
$$y(0) = 0$$

厳密解

$$x_s(t) = \sin t$$
$$y_s(t) = \cos t$$

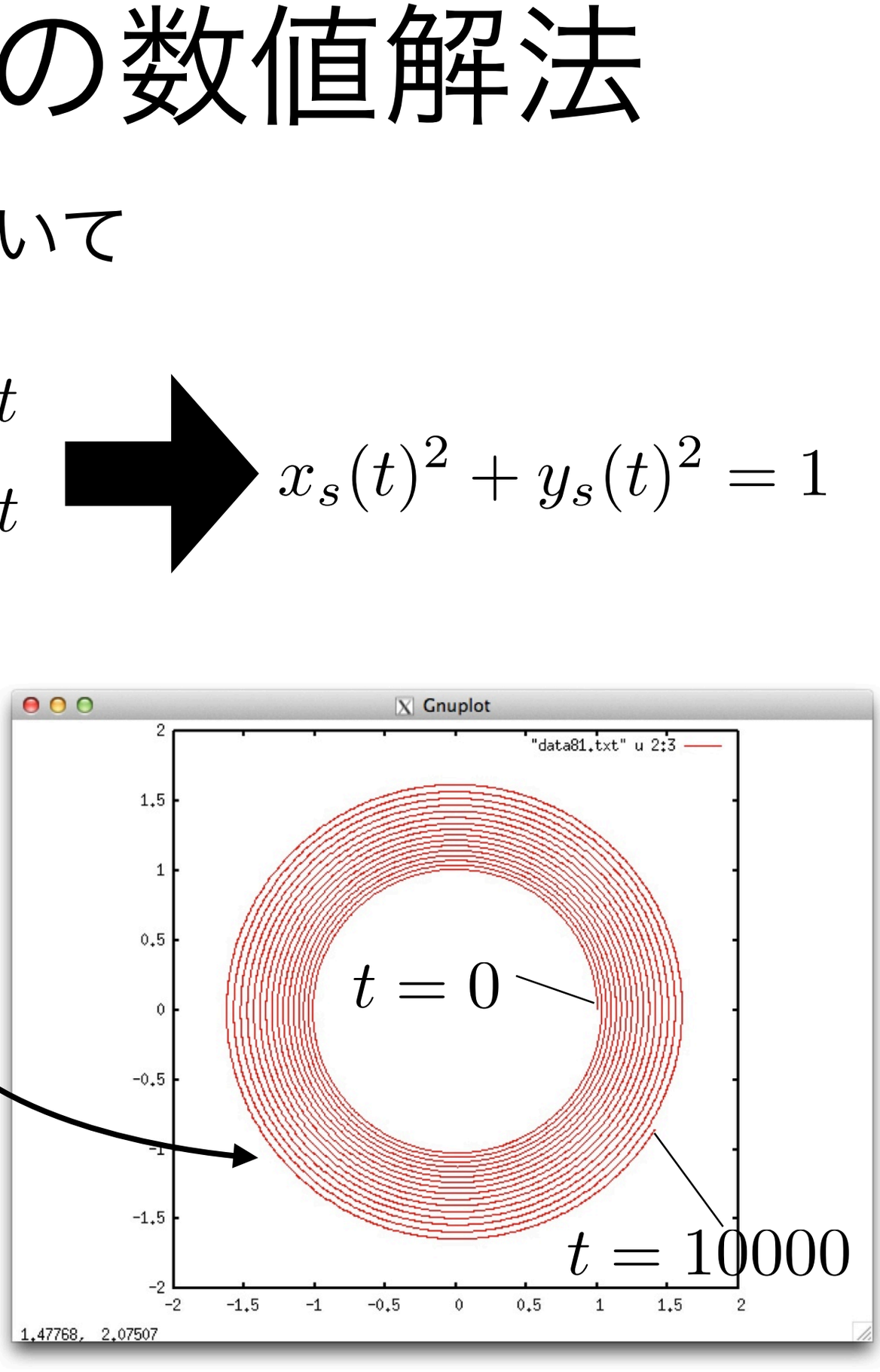
$$x_s(t)^2 + y_s(t)^2 = 1$$

問題

数値計算では軌道が膨らんでいる.

$$f'(t) := \lim_{\delta t \to 0} \frac{f(t+\delta t) - f(t)}{\delta t}$$

を大雑把に計算したことが原因！



常微分方程式の数値解法

誤差について

$$f'(t) := \lim_{\delta t \to 0} \frac{f(t+\delta t) - f(t)}{\delta t}$$

を大雑把に計算したことが原因！

この式を厳密に成り立たせるためには $dt \to 0$ でなければならない.

とりあえず小さなdtをセットしておき，大きくしても解の挙動が変わらないものを選ぶ.

数値計算で得られた解は，与えられた方程式の解にきちんとなっているか確認せねばならない.

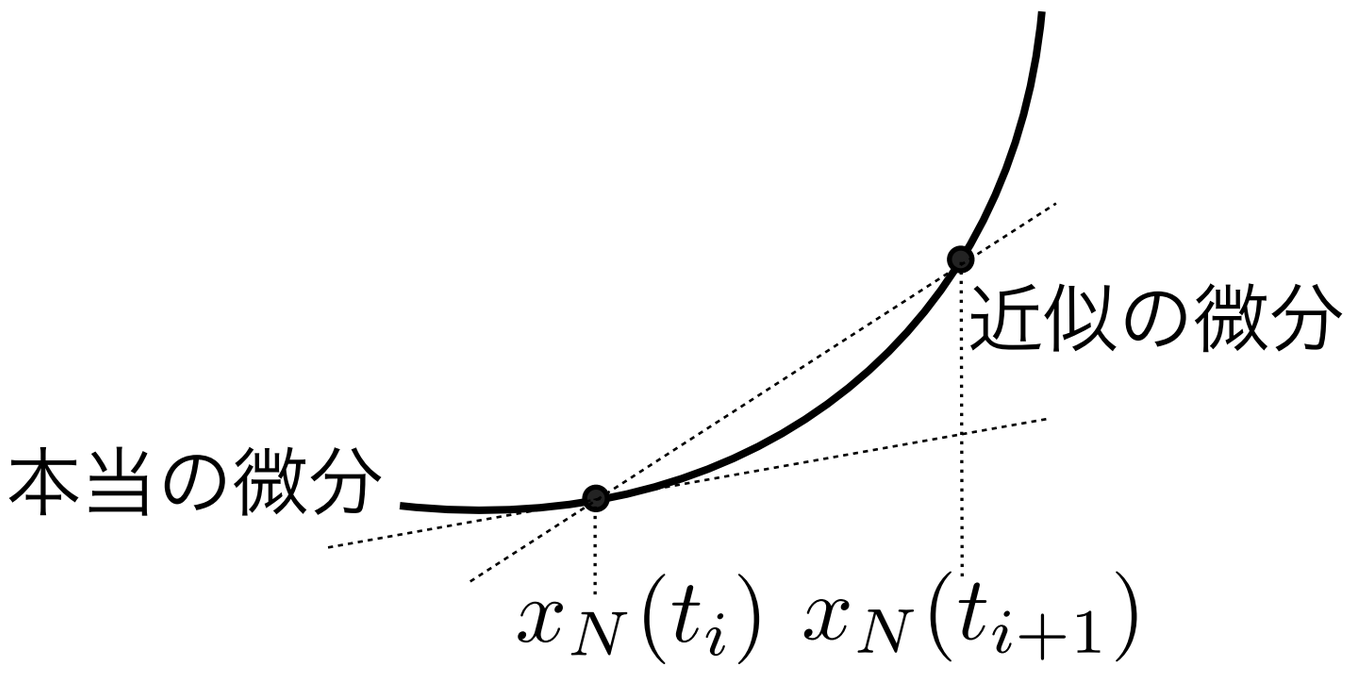
厳密解がわかっている問題でまずしっかりチェックして，未知な問題に取り組むことが大切！

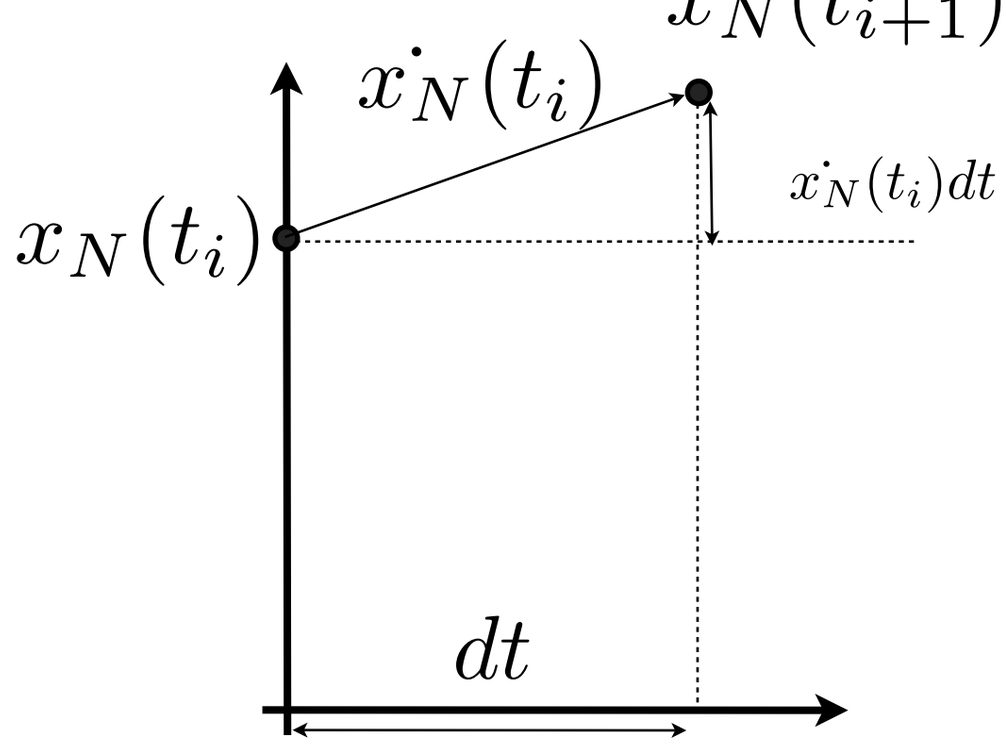
常微分方程式の数値解法まとめ

$$\dot{x}(t) = f(x, t)$$

オイラー法とは 微分を 1 次の差分で近似する.

$$\frac{x_N(t_{i+1}) - x_N(t_i)}{dt} \approx f(x_N(t_i), t_i)$$





でも毎回，問題が変わるたびにコードを書くのは面倒．．．

秋山謹製 常微分方程式計算ライブラリ(+アルファ)

mybasic8.4にいく

```
Common;cd mybasic8.4
mybasic8.4;ls
.DS_Store
CG_Solve.c
Data2FFT.c
DoMake*
Dtpr.c
Embedded_Explicit_Runge_Kutta_Multiple.c
Euler.c
ExtendDirichlet.c
ExtendNeumann.c
ExtendPeriodic.c
Fitting.c
GenMatrix.c
GetParam.c
GetParam2.c
Initialize.c
LuDcmp.c
LuSolve.c
Makefile*
Makefile.GENERIC*
Makefile_intel*
Mean.c*
MxmnCheck.c*
QuickSort.c
Rainbow.c*
Runge.c
Runge_n_dim.c
Sign.c*
Sum.c*
TEST_PROGRAM/
basic.h
fftsg_h.c
libbasic_intel.a
mt19937-64.c
mt19937ar.c
myc++*
mycc*
sgiimage.c*
timer.h
```

```
mybasic8.4;make
cc -O2 -DDOUBLE -c CG_Solve.c
cc -O2 -DDOUBLE -c Data2FFT.c
cc -O2 -DDOUBLE -c Dtpr.c
cc -O2 -DDOUBLE -c Embedded_Explicit_Runge_Kutta_Multiple.c
cc -O2 -DDOUBLE -c Euler.c
cc -O2 -DDOUBLE -c ExtendDirichlet.c
cc -O2 -DDOUBLE -c ExtendNeumann.c
cc -O2 -DDOUBLE -c ExtendPeriodic.c
cc -O2 -DDOUBLE -c fftsg_h.c
cc -O2 -DDOUBLE -c Fitting.c
cc -O2 -DDOUBLE -c GenMatrix.c
cc -O2 -DDOUBLE -c GetParam.c
cc -O2 -DDOUBLE -c GetParam2.c
cc -O2 -DDOUBLE -c Initialize.c
cc -O2 -DDOUBLE -c LuDcmp.c
cc -O2 -DDOUBLE -c LuSolve.c
cc -O2 -DDOUBLE -c Mean.c
cc -O2 -DDOUBLE -c mt19937ar.c
cc -O2 -DDOUBLE -c mt19937-64.c
cc -O2 -DDOUBLE -c MxmnCheck.c
cc -O2 -DDOUBLE -c QuickSort.c
cc -O2 -DDOUBLE -c Rainbow.c
cc -O2 -DDOUBLE -c Runge_n_dim.c
cc -O2 -DDOUBLE -c Runge.c
cc -O2 -DDOUBLE -c Sign.c
cc -O2 -DDOUBLE -c Sum.c
Done
```

libbasic.aというライブラリと
basic.hというヘッダーができる.

できない人は挙手

Type6に移動して，今作ったlibbasic.aとbasic.hをD&Dでコピーする.
(もともと入っていますが，必ず置き換えてください．)

main1.cをコンパイル＆実行する.

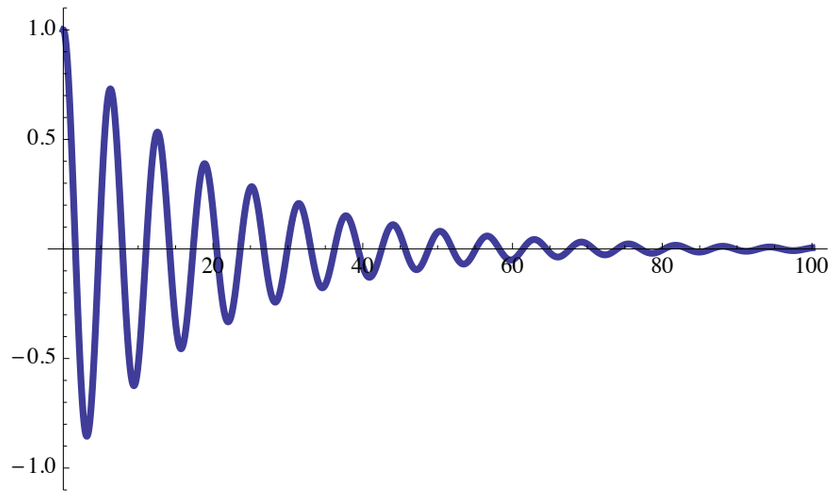
```
Type6;cc main.c libbasic.a
Type6;./a.out

6 outputs of genrand64_real2()
0.786820954867802
0.250480340688029
0.710671228978655
0.946667800960970
0.019271058195814
0.404902144816168
Type6;
```

ヒント：午前のスライドを思い出して

バネ・ダンパ系の問題

$$\frac{d^2x}{dt^2} + 2\gamma\frac{dx}{dt} + \omega_0^2 x = 0$$

$$x(0) = x_0 \quad x'(0) = p_0$$



解のグラフはこんな感じになる.

$0 < \gamma < \omega_0$ のとき，厳密解は

$$x(t) = e^{-\gamma t}(x_0 \cos\omega t + \frac{p_0 + \gamma x_0}{\omega}\sin\omega t)$$

$$p(t) = e^{-\gamma t}(p_0 \cos\omega t - \frac{p_0\gamma + \omega_0^2 x_0}{\omega}\sin\omega t)$$

ただし

$$p(t) := x'(t)$$

$$\omega := \sqrt{\omega_0^2 - \gamma^2}$$

問題は一階の微分方程式系で以下のようにもかける

$$\frac{dp}{dt} = -2\gamma p - \omega_0^2 x$$
$$\frac{dx}{dt} = p$$

本ソルバは問題を使用する場合は必ず，連立の一階の常微分方程式系に書き換える必要がある.

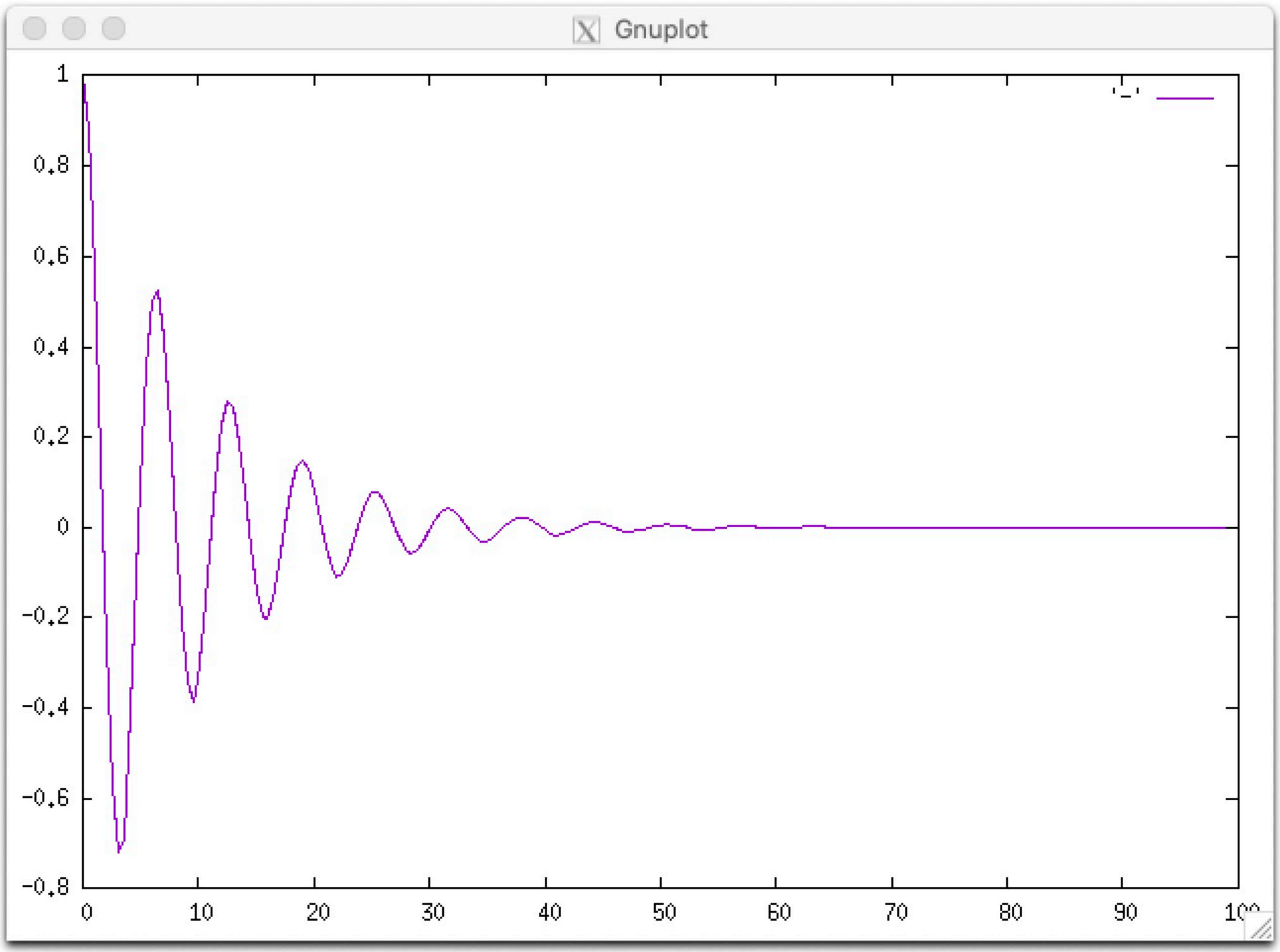
$$\boldsymbol{x}'(t) = \boldsymbol{f}(\boldsymbol{x}(t), t)$$

プログラムの流れ

main2.c

```c
#include<stdio.h>
#include<math.h>
#include"./basic.h"

#define     N           (2)
#define     omega_0     (1.0)
#define     gamma       (0.1)
#define     omega       (sqrt(omega_0 * omega_0 - gamma * gamma))

void
kansu_f(int dim,double *x,double *x_dot,double time)
{
    x_dot[0] = - 2 * gamma * x[0] - omega_0 * omega_0 * x[1];
    x_dot[1] = x[0];
}

int main(void)
{
    int i_time = 0;

    //For Runge Kutta Method
    double  u[N], work_u[5 * N];

    double  p0 = 0,x0 = 1;
    double  p,x,t,dt = 0.5;

    p = p0;
    x = x0;

    u[0] = p;
    u[1] = x;

    t = i_time * dt;

    for(i_time = 1;i_time * dt < 100 ; i_time++)
    {
        printf("%f %f\n",t,x);
        t = i_time * dt;
        Runge_n_dim(N,u,t,dt,work_u,kansu_f);
        p = u[0];
        x = u[1];

    }
    return 0;
}
```

main2_gnuplot.c



本当にあっているかを厳密解と比較

main3.c

```
double  p0 = 0,x0 = 1;
double  p,x;
double  genmitu_p,genmitu_x;
double  t;
double  err_p,err_x;
double  max_err_p = 0.0,max_err_x = 0.0;

p = p0;
x = x0;

u[0] = p;
u[1] = x;

t = i_time * dt;

genmitu_p = exp(- gamma * t) * (p0 * cos (omega * t) - (p0 * gamma + x0 * omega_0 * omega_0) / omega * sin (omega * t));
genmitu_x = exp(- gamma * t) * (x0 * cos (omega * t) + (p0 + gamma * x0) / omega * sin (omega * t));

for(i_time = 1;i_time * dt < 100 ; i_time++)
    {
        t = i_time * dt;
        err_p = fabs(p - genmitu_p);
        err_x = fabs(x - genmitu_x);
        if(err_p > max_err_p)
        {
            max_err_p = err_p;
            printf("max_err_p %15.15lf\n",max_err_p);
        }
        if(err_x > max_err_x)
        {
            max_err_x = err_x;
            printf("max_err_x %15.15lf\n",max_err_x);
        }
        Runge_n_dim(N,u,t,dt,work_u,kansu_f);
        p = u[0];
        x = u[1];
        genmitu_p = exp(- gamma * t) * (p0 * cos (omega * t) - (p0 * gamma + x0 * omega_0 * omega_0) / omega * sin (omega * t));
        genmitu_x = exp(- gamma * t) * (x0 * cos (omega * t) + (p0 + gamma * x0) / omega * sin (omega * t));
    }
    return 0;
}
```

４次精度の所以

dt=1.0 のときの出力

```
Type6;cc main3.c libbasic.a
Type6;./a.out
max_err_p 0.006424345176904
max_err_x 0.004361442387233
max_err_x 0.014741263439546
max_err_p 0.017573909644766
max_err_p 0.021183985329894
max_err_x 0.026719793298931
max_err_p 0.023189349662586
max_err_p 0.029437946866847
max_err_x 0.030083890367140
max_err_p 0.032336105425555
```

p については，厳密解との最大誤差は0.0323

表を作ってみよう

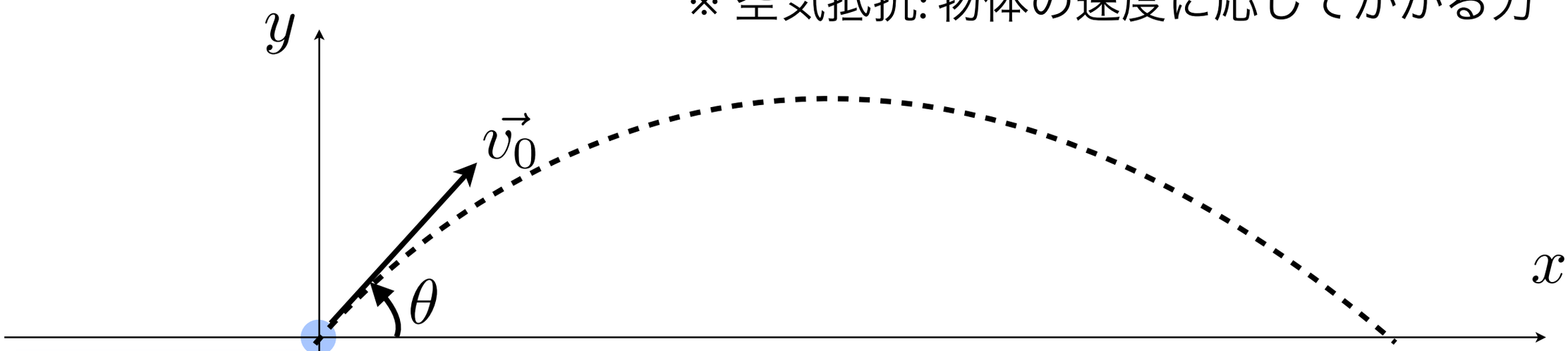
dt	最大誤差	0の数
1E+00	0.0323	2
1E-01	0.000003089740248	6
1E-02	0.000000000306660	10
1E-03	0.000000000000034	14
1E-04	0.000000000000007	15

C言語の実数の精度は高々15桁なので，この辺で終わり

練習問題

空気抵抗のある斜方投射

※ 空気抵抗: 物体の速度に応じてかかる力



$$m\ddot{x}(t) = -\gamma\dot{x}(t)$$

$$m\ddot{y}(t) = -mg - \gamma\dot{y}(t)$$

$$\dot{y}(0) = v_0 \sin\theta$$

$$\dot{x}(0) = v_0 \cos\theta$$

$$x(0) = 0$$

$$y(0) = 0$$

反応拡散方程式
数値解法入門

問題をさらに簡単な問題に分けて考える

偏微分方程式

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u + f(u)$$

問題をさらに簡単な問題に分けて考える

偏微分方程式

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u + f(u)$$

$u = u(t, x)$ 空間１次元

$u = u(t, x, y)$ 空間２次元

$u = u(t, x, y, z)$ 空間３次元

問題をさらに簡単な問題に分けて考える

偏微分方程式

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u + f(u)$$

$u = u(t, x)$ 空間１次元

$u = u(t, x, y)$ 空間２次元

$u = u(t, x, y, z)$ 空間３次元

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u$$

$u = u(t, x)$

偏微分方程式

問題をさらに簡単な問題に分けて考える

偏微分方程式

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u + f(u)$$

$u = u(t, x)$ 空間１次元

$u = u(t, x, y)$ 空間２次元

$u = u(t, x, y, z)$ 空間３次元

$$\frac{\partial u}{\partial t} = D_u \nabla^2 u$$

$u = u(t, x)$

偏微分方程式

$$\frac{\partial u}{\partial t} = f(u) \qquad u = u(t, x)$$

偏微分方程式なんだけど空間に依存しないなら．．．

$$\frac{du}{dt} = f(u) \quad u = u(t)$$

常微分方程式になる.

反応拡散方程式＝

常微分方程式＋拡散方程式

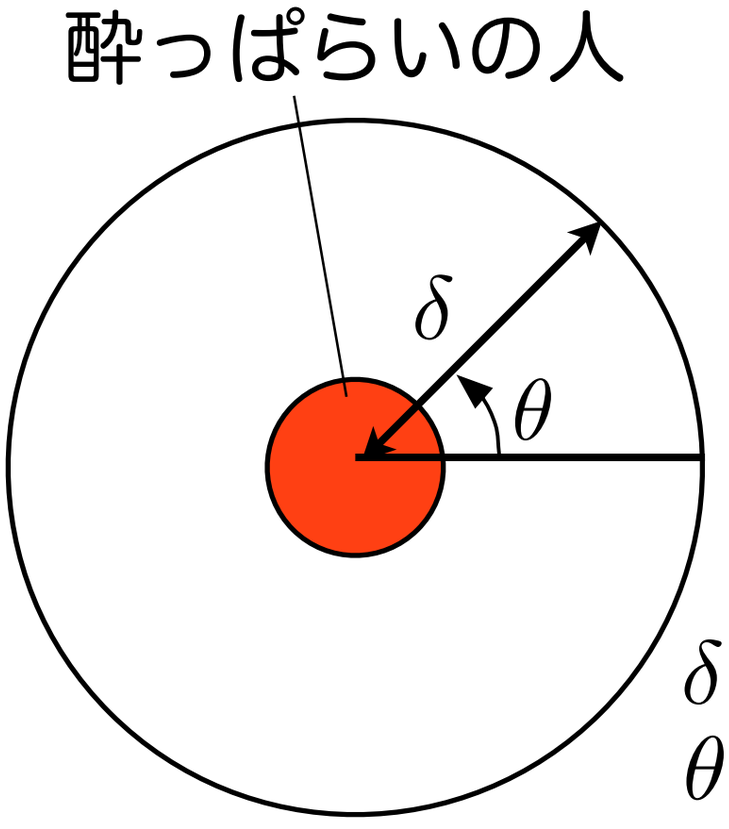
※大きなくくりでは偏微分方程式

拡散方程式の
数値解法入門

そもそも拡散方程式とは?

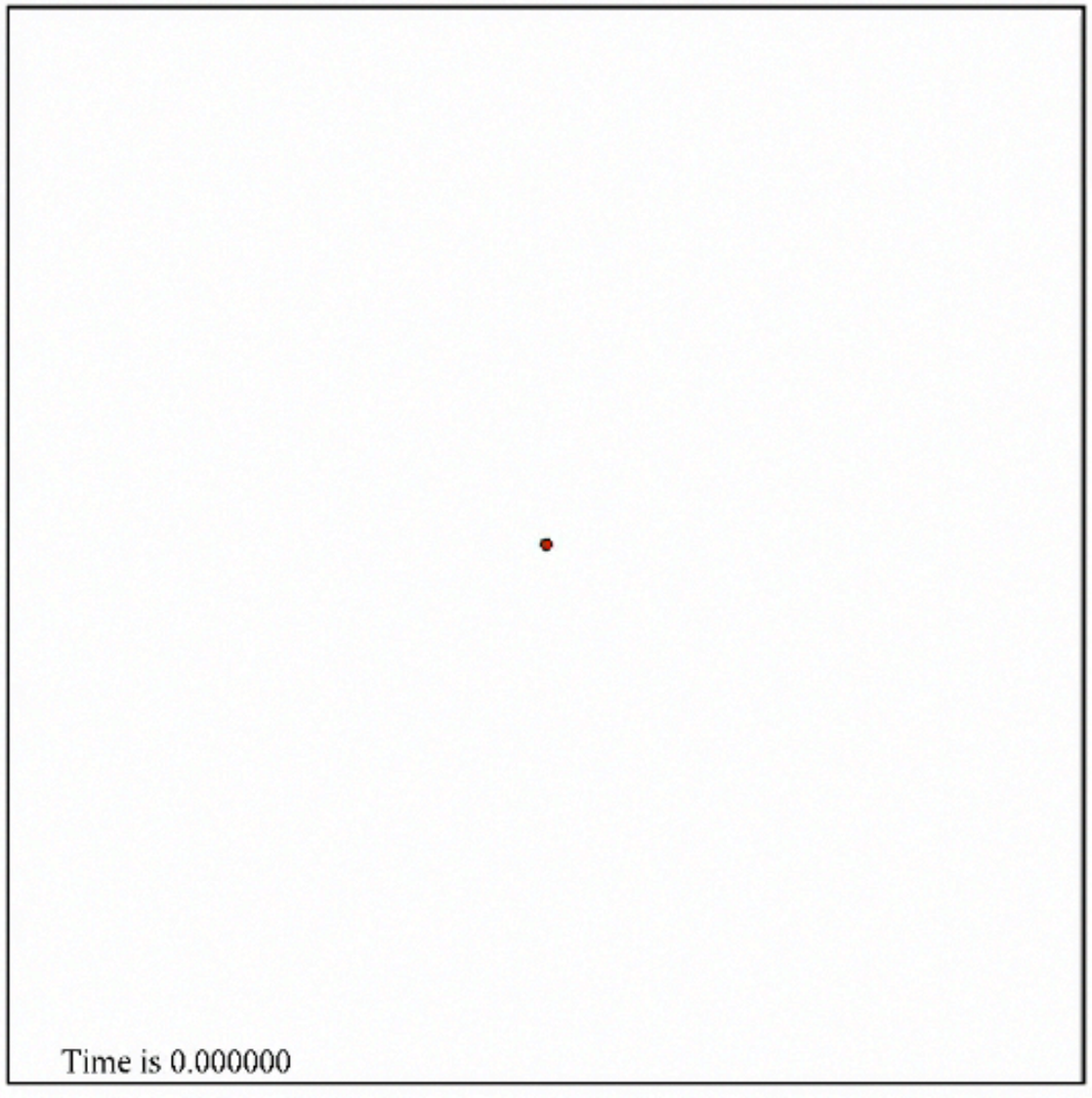
酔歩（酔っ払い歩き）

酔っぱらいの人の運動方程式を求めよう！



δ: 定数
θ: ランダム

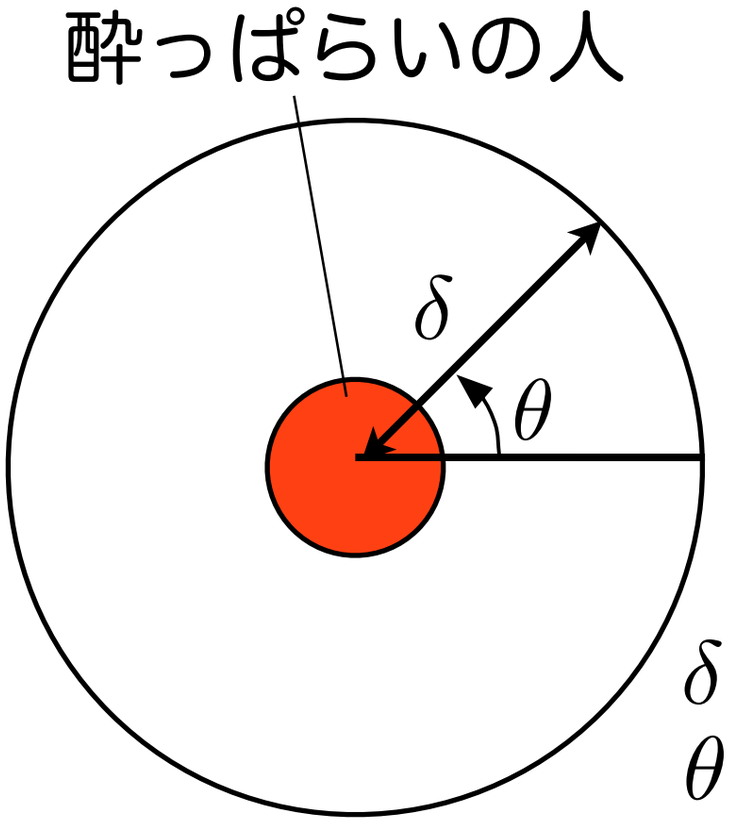
酔っぱらいの人は角度がランダムで一定の距離だけふらふらするはず.



酔っぱらいの人が１人の場合

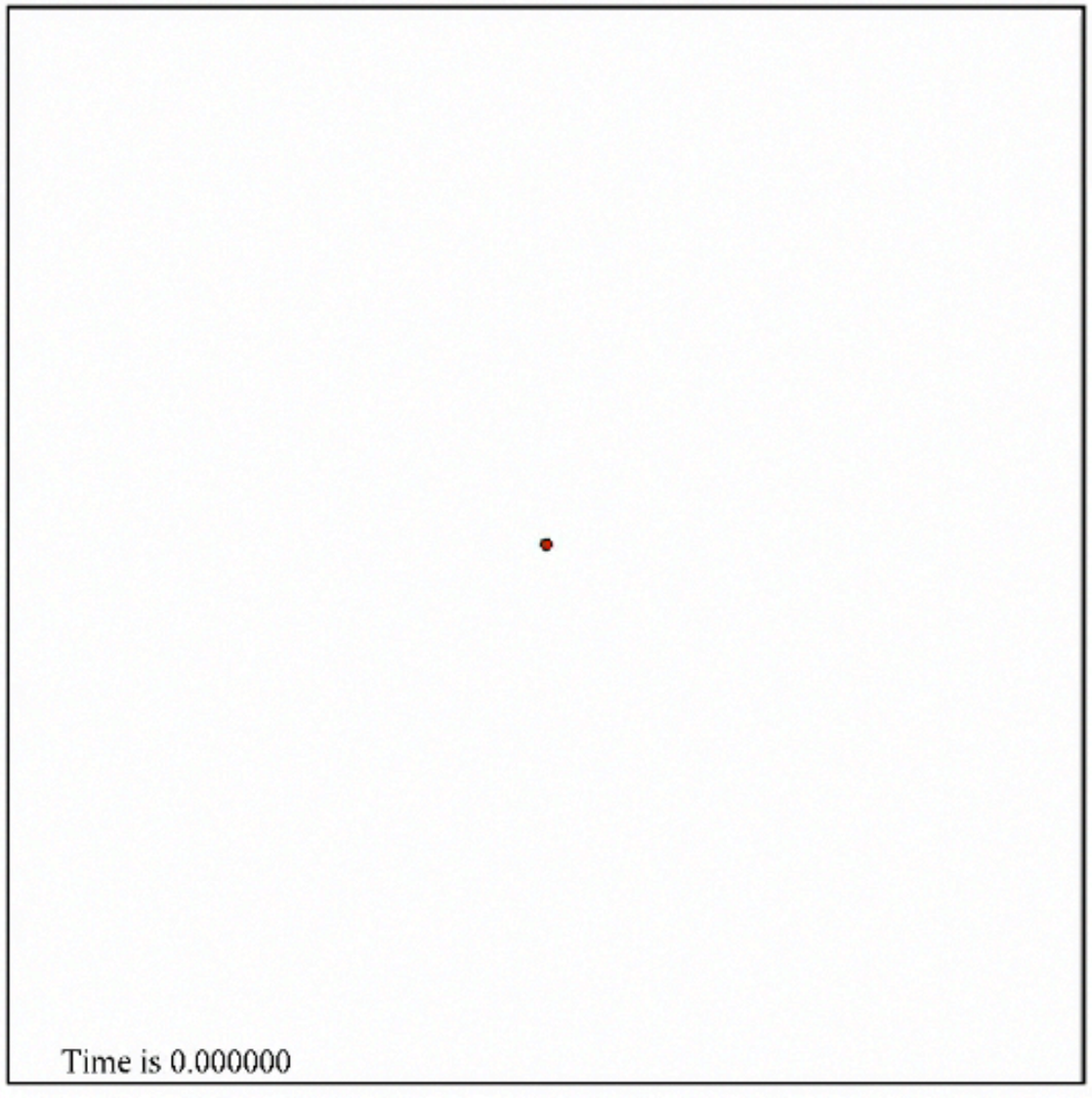
酔歩（酔っ払い歩き）

酔っぱらいの人の運動方程式を求めよう！



δ: 定数
θ: ランダム

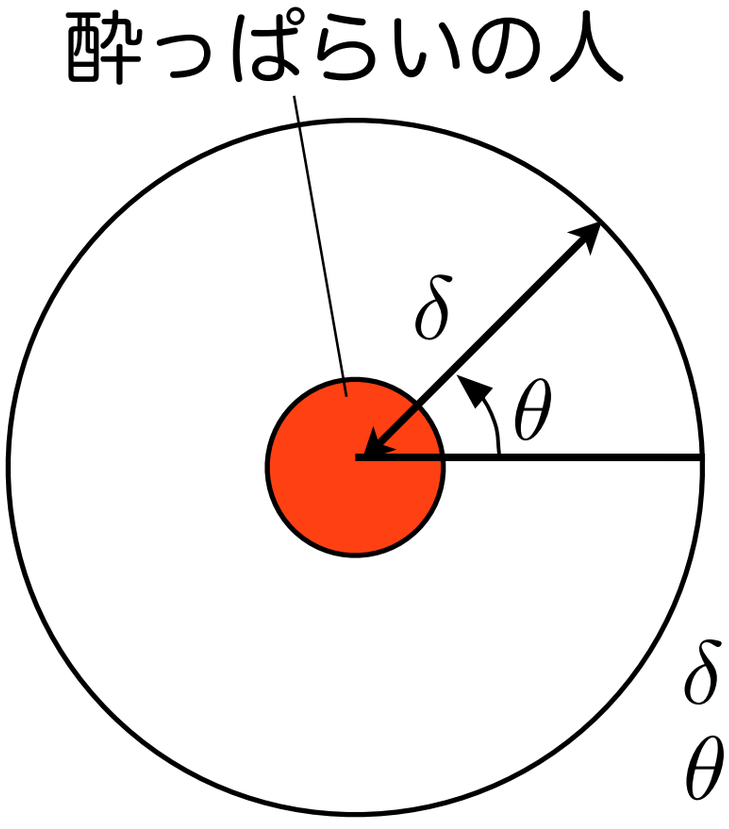
酔っぱらいの人は角度がランダムで一定の距離だけふらふらするはず.



酔っぱらいの人が１人の場合

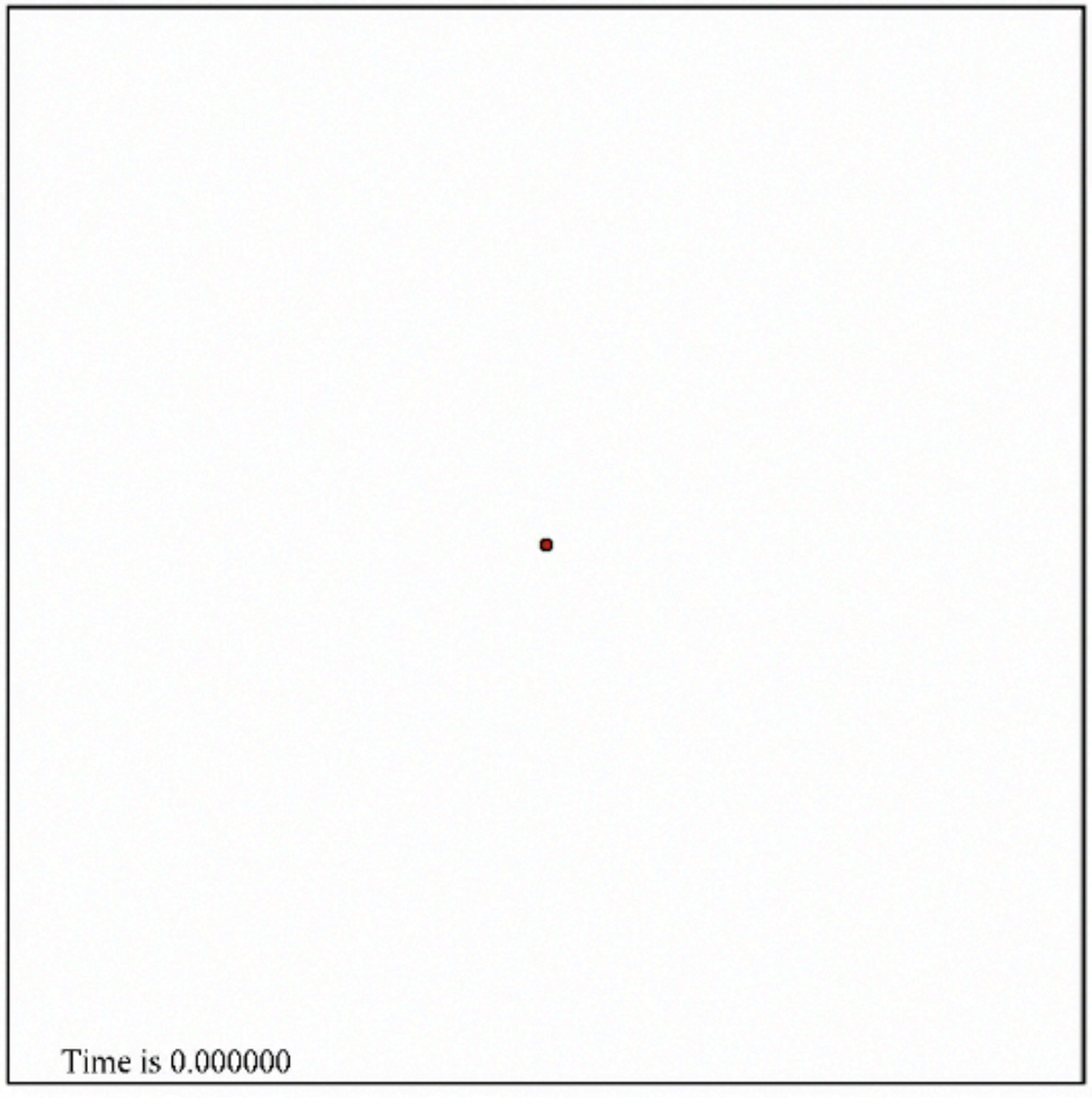
酔歩（酔っ払い歩き）

酔っぱらいの人の運動方程式を求めよう！



δ: 定数
θ: ランダム

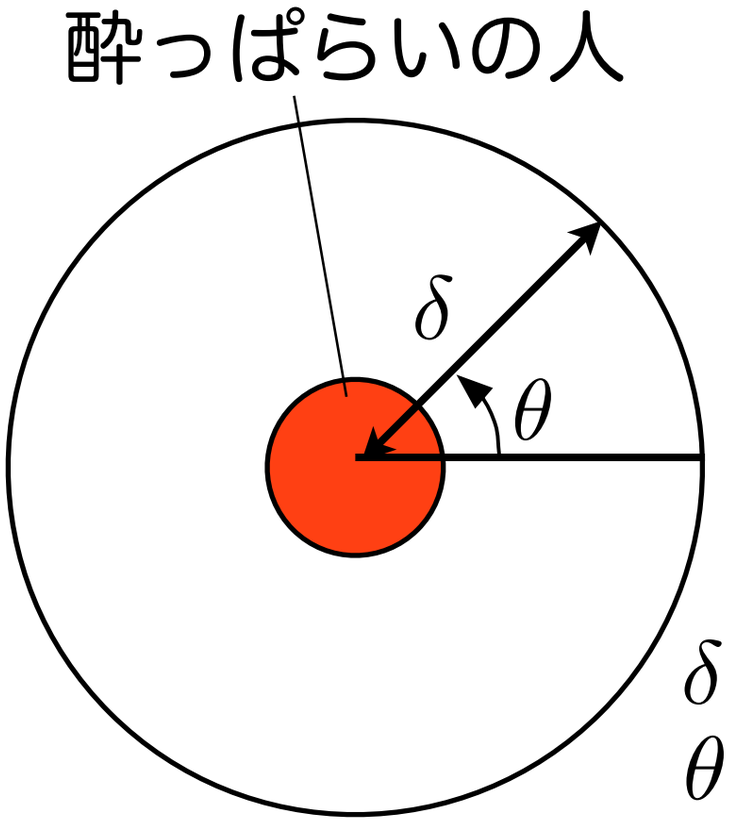
酔っぱらいの人は角度がランダムで一定の距離だけふらふらするはず.



酔っぱらいの人が１００人の場合

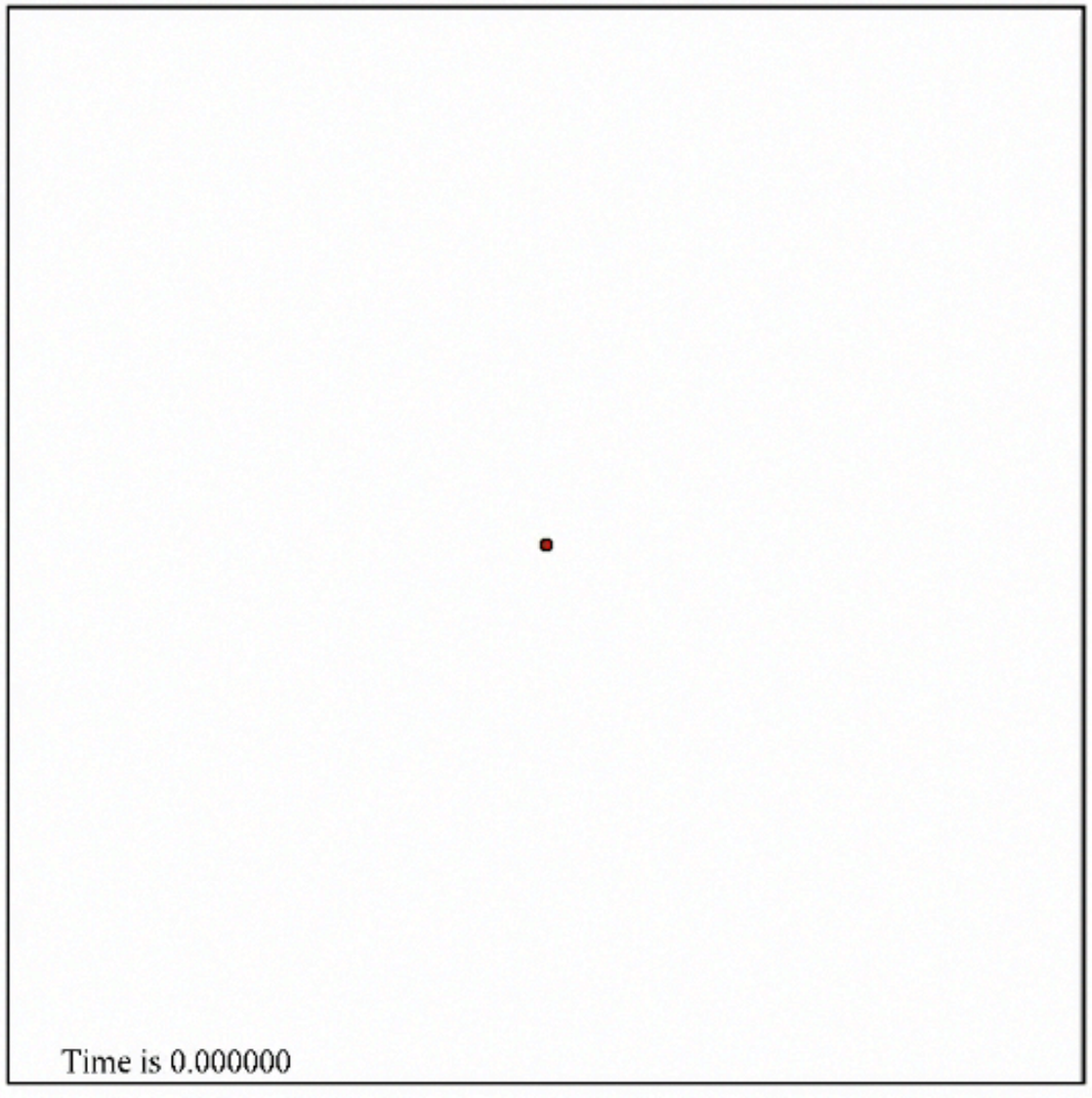
酔歩（酔っ払い歩き）

酔っぱらいの人の運動方程式を求めよう！



δ: 定数
θ: ランダム

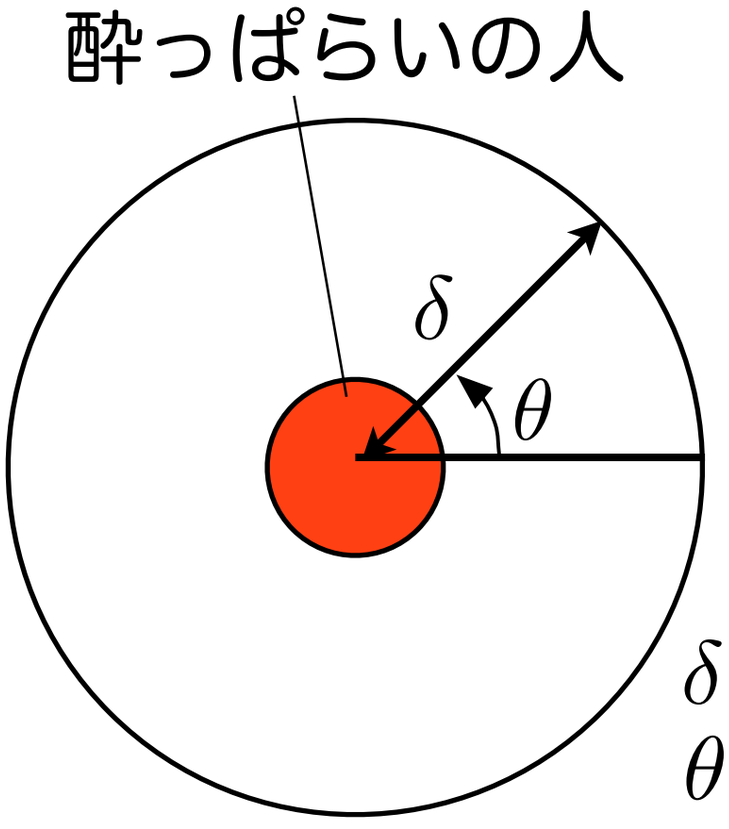
酔っぱらいの人は角度がランダムで一定の距離だけふらふらするはず.



酔っぱらいの人が１００人の場合

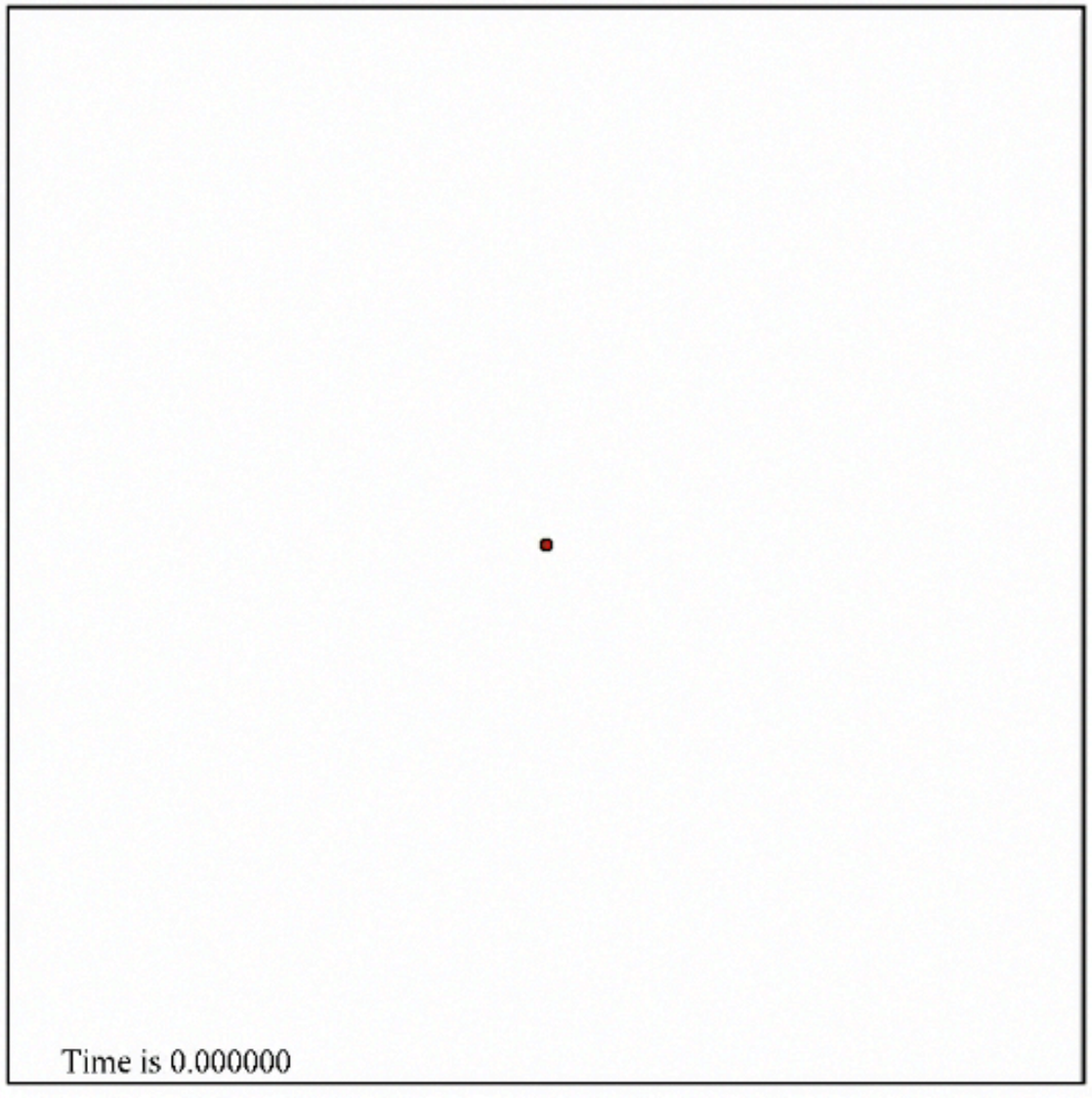
酔歩（酔っ払い歩き）

酔っぱらいの人の運動方程式を求めよう！



δ: 定数
θ: ランダム

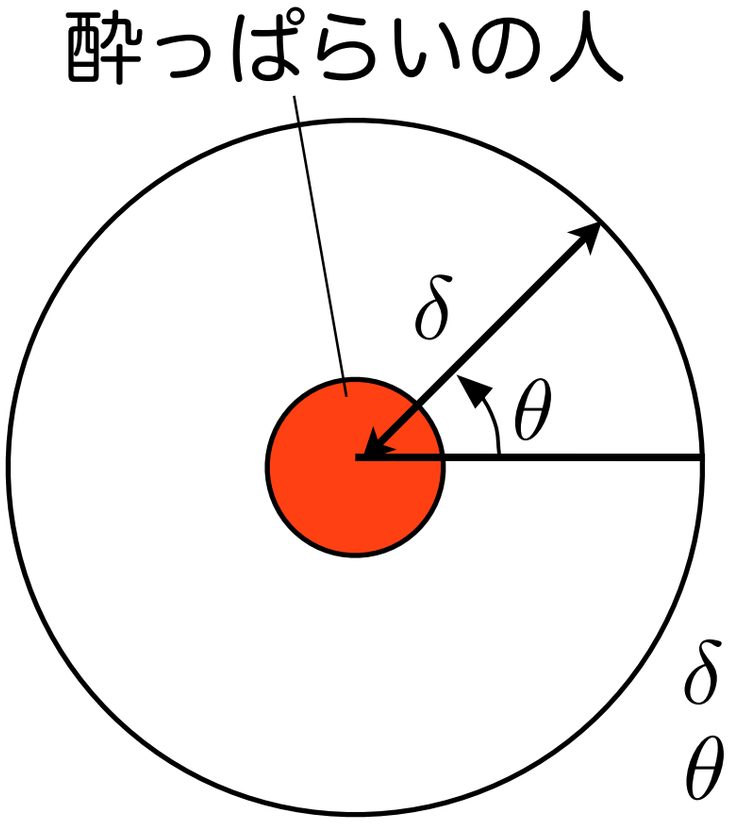
酔っぱらいの人は角度がランダムで一定の距離だけふらふらするはず.



酔っぱらいの人が１００００人の場合

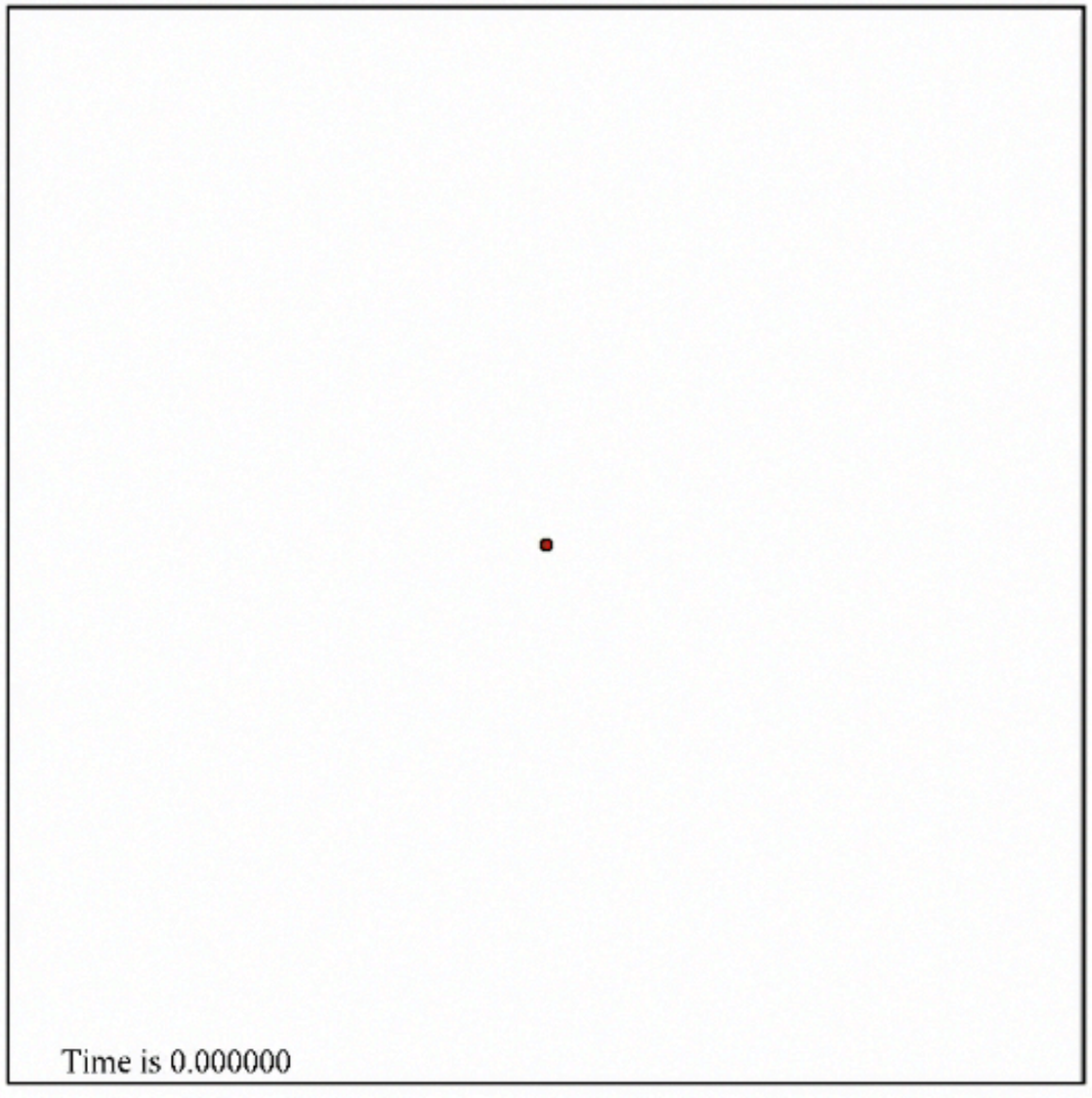
酔歩（酔っ払い歩き）

酔っぱらいの人の運動方程式を求めよう！



δ: 定数
θ: ランダム

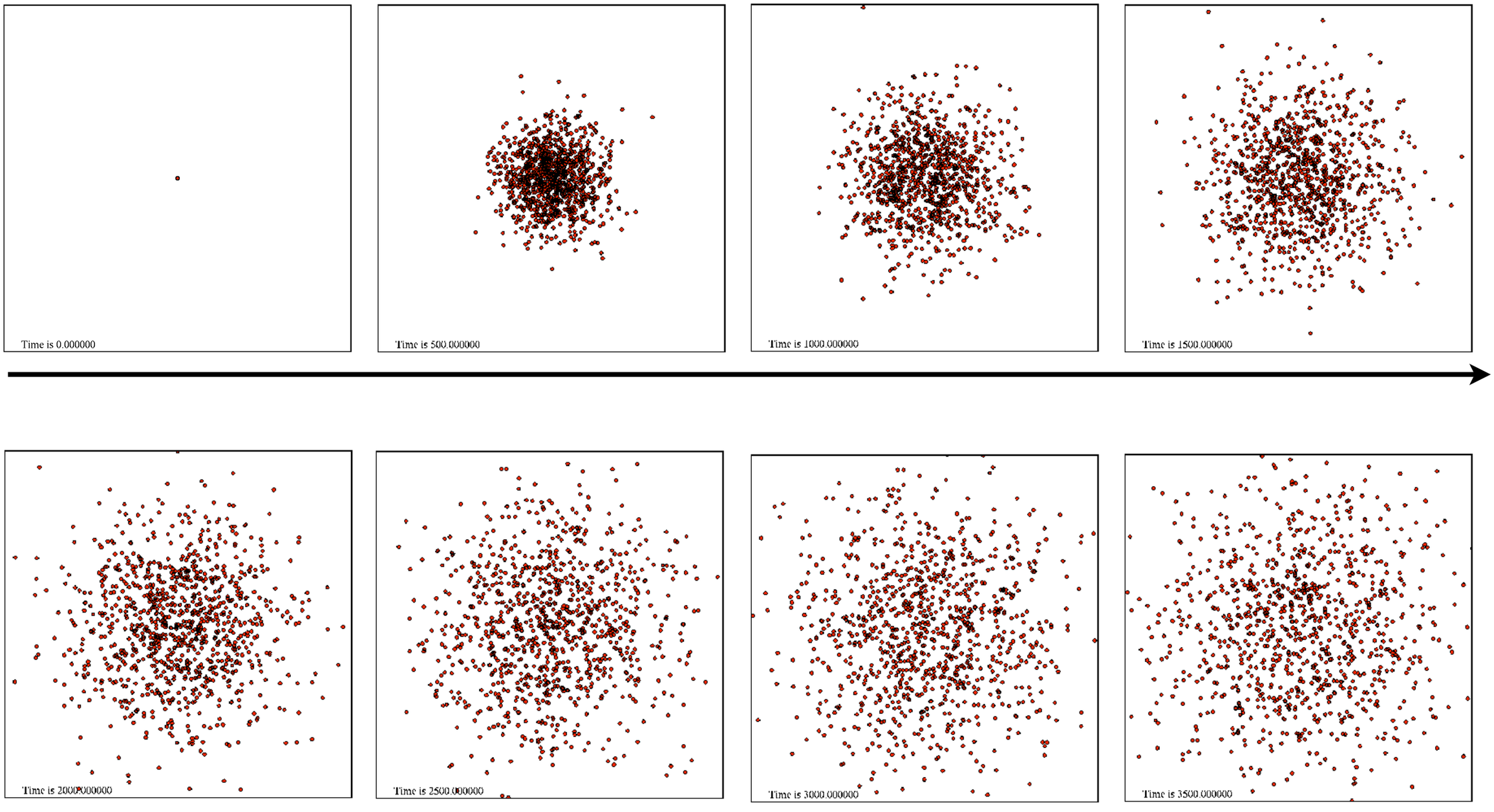
酔っぱらいの人は角度がランダムで一定の距離だけふらふらするはず.



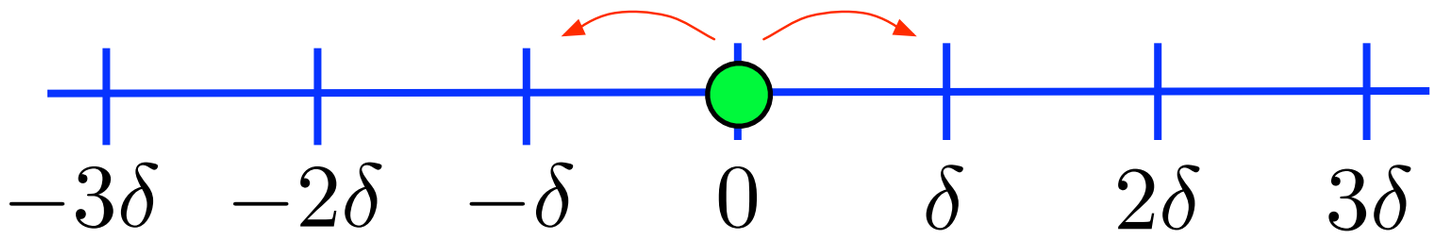
酔っぱらいの人が１００００人の場合

酔歩（酔っ払い歩き）

※酔っぱらいは意図して外に向かっているのではない



どういう方程式で記述できるか？



(1) 各粒子は τ 秒ごとに距離 δ だけ右か左に移動する.

(2) 各ステップで左右に行く確率はそれぞれ 1/2 であり,
前のステップでどちらに動いたかは記憶していない.

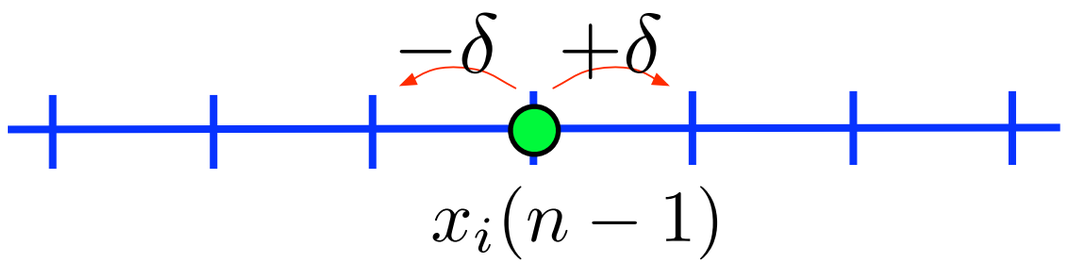
(3) 各粒子は他の粒子と独立に動き, 相互作用することはない.

※ここから資料の一部は小林亮先生（広島大学）にいただきました.

粒子の位置の平均

初期時刻に I 個の全粒子が原点にいる場合を考える.

第 i 番目の粒子の第 n ステップにおける位置： $x_i(n)$



$$x_i(n) = x_i(n-1) \pm \delta$$ ⟸ 確率 1/2

n 第　ステップにおける粒子の平均位置：　$\langle x(n) \rangle = \frac{1}{I} \sum_{i=1}^{I} x_i(n)$

粒子数が十分大きいとき　$\langle x(n) \rangle$ と　$\langle x(n-1) \rangle$　の関係は

$$\langle x(n) \rangle = \frac{1}{I} \sum_{i=1}^{I} (x_i(n-1) \pm \delta) = \frac{1}{I} \sum_{i=1}^{I} x_i(n-1) = \langle x(n-1) \rangle$$

どういう方程式で記述できるか？

$$\langle x(0) \rangle = 0 \quad \Rightarrow \quad \langle x(n) \rangle = 0$$

第nステップにおける粒子の位置の分散：　$\langle x(n)^2 \rangle$

$$\langle x(n)^2 \rangle = \frac{1}{I} \sum_{i=1}^{I} x_i(n)^2$$

$$x_i(n)^2 = x_i(n-1)^2 \pm 2\delta x_i(n-1) + \delta^2$$

$$\langle x(n)^2 \rangle = \langle x(n-1)^2 \rangle + \delta^2 = \langle x(n-2)^2 \rangle + 2\delta^2$$
$$= \cdots = \langle x(0)^2 \rangle + n\delta^2$$

$$\boxed{\langle x(n)^2 \rangle = n\delta^2}$$

粒子の広がり具合

時間変数 t を導入する.　　$t = \tau n$

$$\left\langle x(n)^2 \right\rangle = n\delta^2 \Rightarrow \left\langle x(t)^2 \right\rangle = \frac{\delta^2}{\tau} t$$

$$D = \frac{\delta^2}{2\tau} \quad \text{とおくと} \quad \left\langle x(t)^2 \right\rangle = 2Dt$$

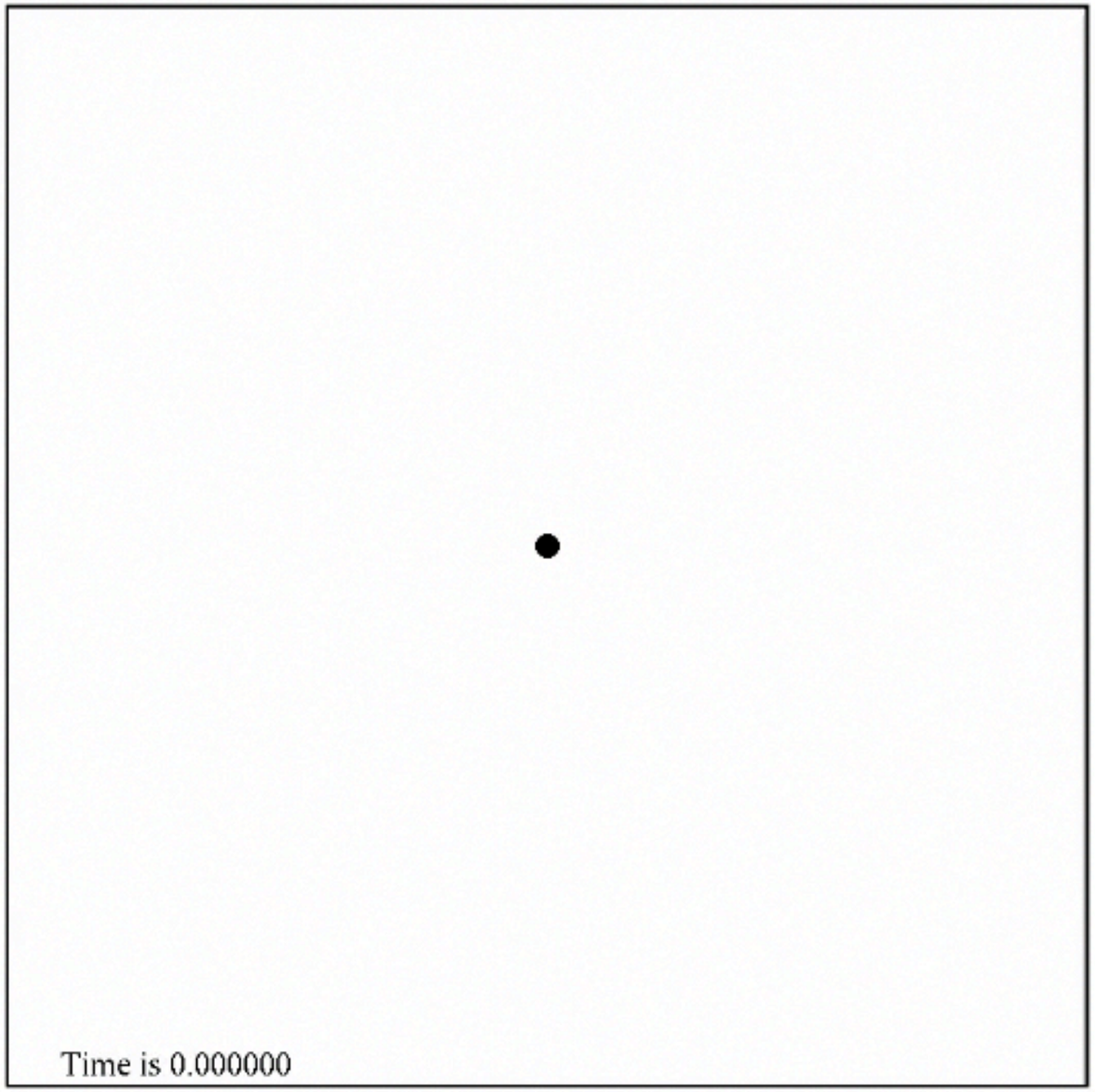
標準偏差　$\sigma(t)$ ⇒ 粒子の広がり具合の指標

$$\sigma(t) = \sqrt{2Dt}$$

D ：拡散係数

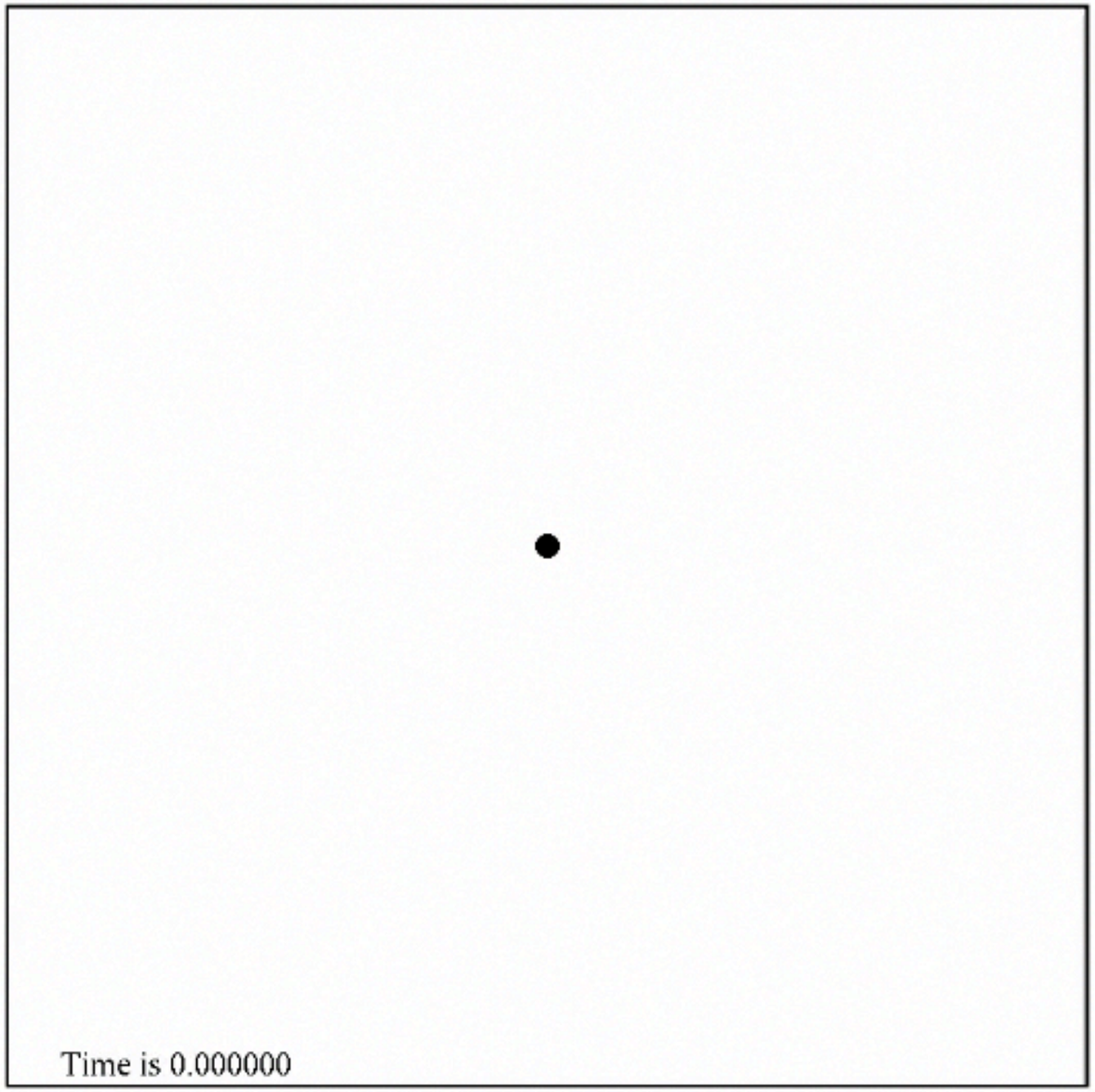
$$[D] = L^2 T^{-1}$$

確かに酔っ払いは．．．



○ 半径$\sqrt{2Dt}$の円

確かに酔っ払いは．．．



○ 半径$\sqrt{2Dt}$の円

電車の中で臭い人が．．．

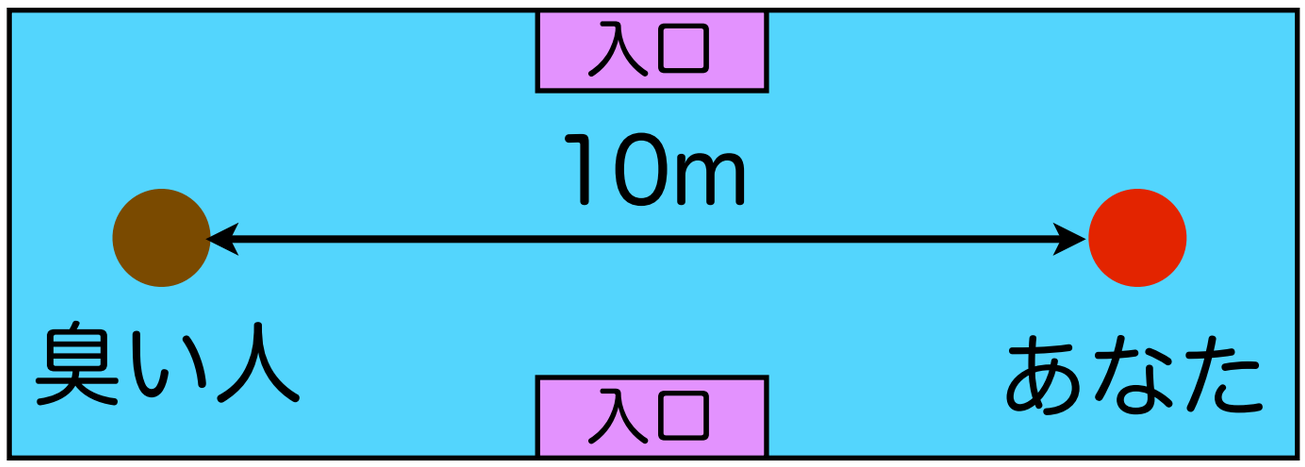
考えてみよう

空気中の微粒子の拡散係数

$$\simeq 10^{-5}\mathrm{m}^2/\mathrm{sec}$$

であることが知られている.

サイズ 10m 程度の電車に臭い人が乗ってきた.
この臭いはどれくらいの時間をかけて，あなたまで届くだろうか？



電車の中で臭い人が．．．

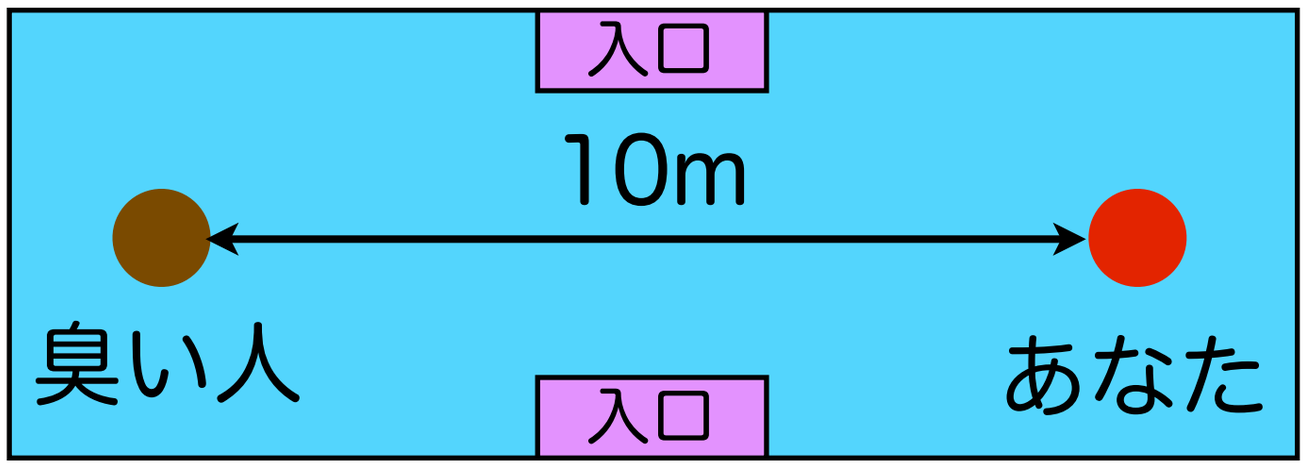
考えてみよう

空気中の微粒子の拡散係数

$$\simeq 10^{-5}\mathrm{m}^2/\mathrm{sec}$$

であることが知られている.

サイズ 10m 程度の電車に臭い人が乗ってきた.
この臭いはどれくらいの時間をかけて，あなたまで届くだろうか？



電車の中で臭い人が．．．

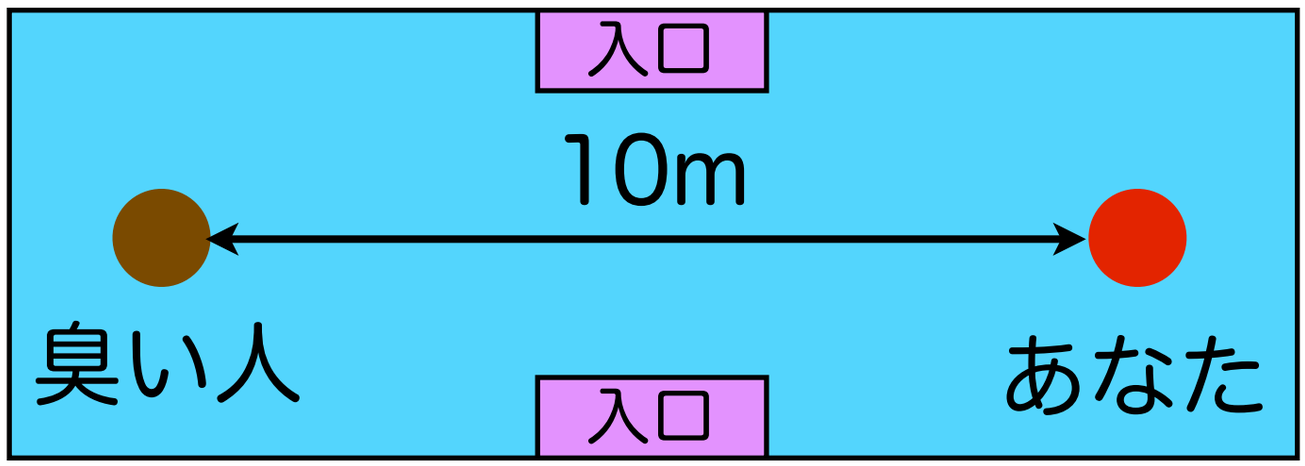
考えてみよう

空気中の微粒子の拡散係数

$$\simeq 10^{-5}\mathrm{m}^2/\mathrm{sec}$$

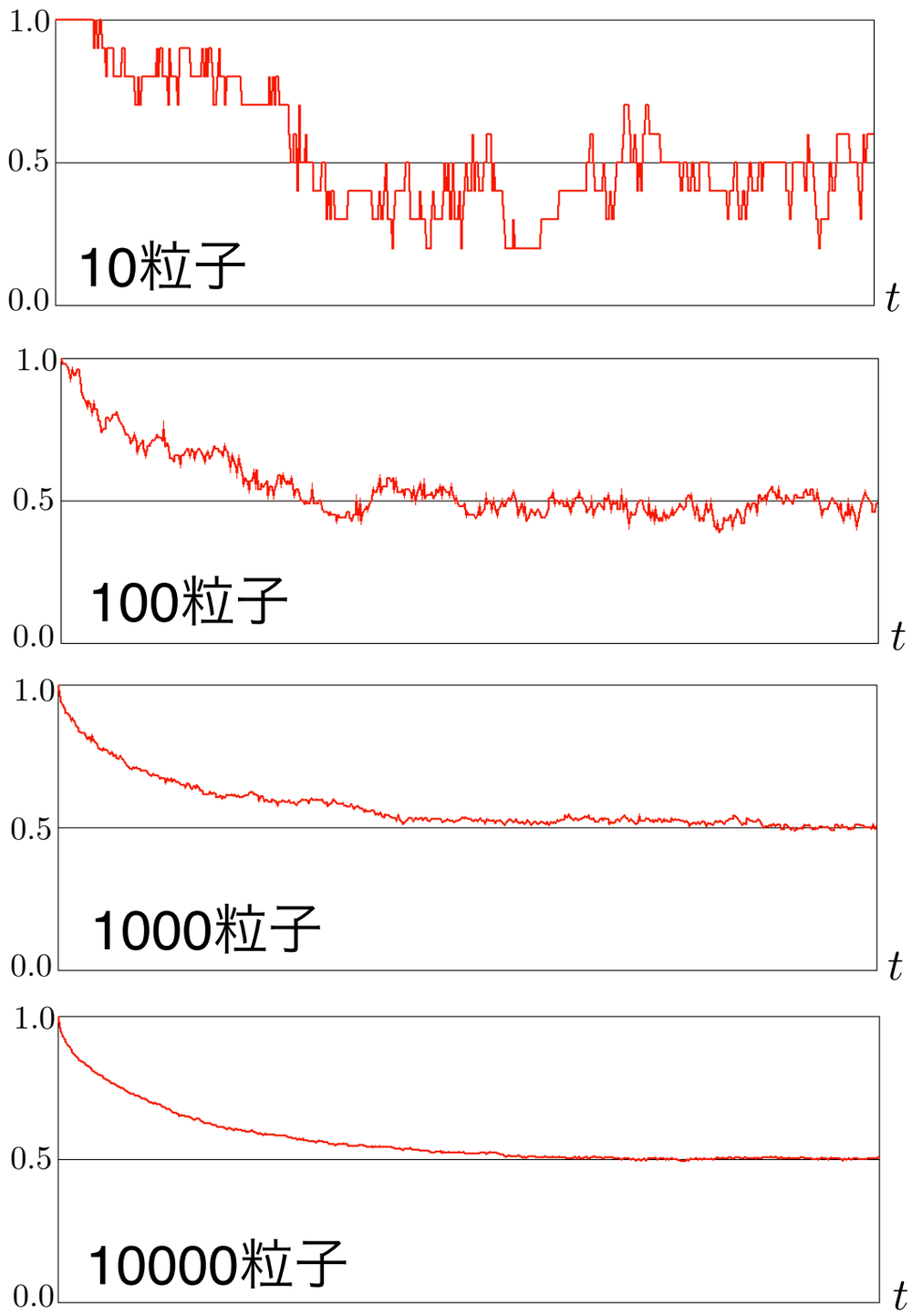
であることが知られている.

サイズ 10m 程度の電車に臭い人が乗ってきた.
この臭いはどれくらいの時間をかけて，あなたまで届くだろうか？



$$t = \frac{\sigma(t)^2}{2D} = \frac{10^2}{2 \times 10^{-5}} = 5 \times 10^6 \mathrm{sec} \simeq 2 \text{ month}$$

粒子の数を増やすと．．．


1.0
0.5
0.0
10粒子
t
1.0
0.5
0.0
100粒子
t
1.0
0.5
0.0
1000粒子
t
1.0
0.5
0.0
10000粒子
t

決定論の拡散方程式

を導出しよう.

思考実験

t=0

…

無限に長い針金

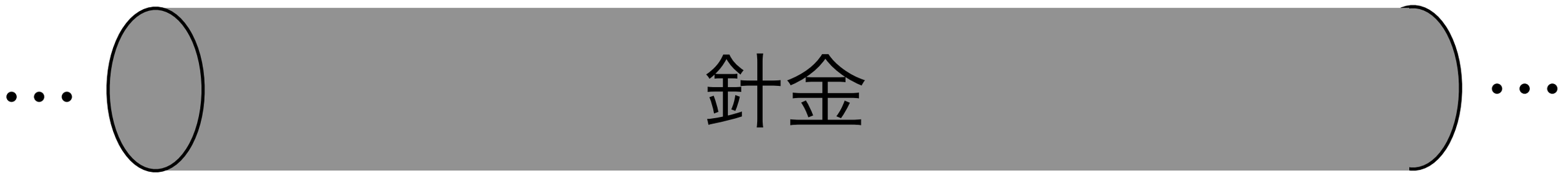
…

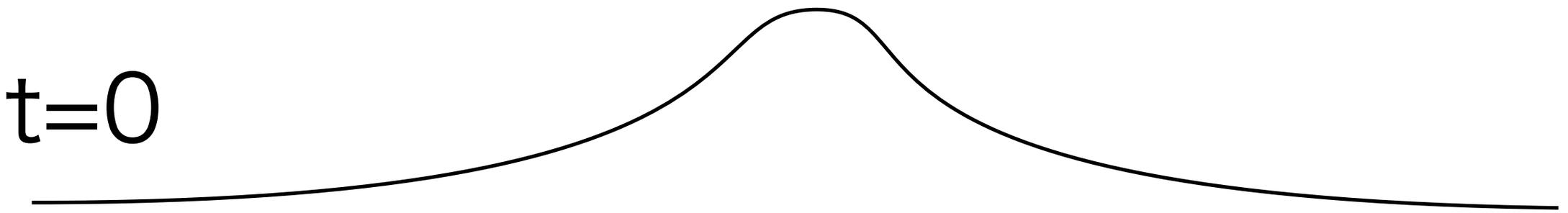
…

針金

…

思考実験

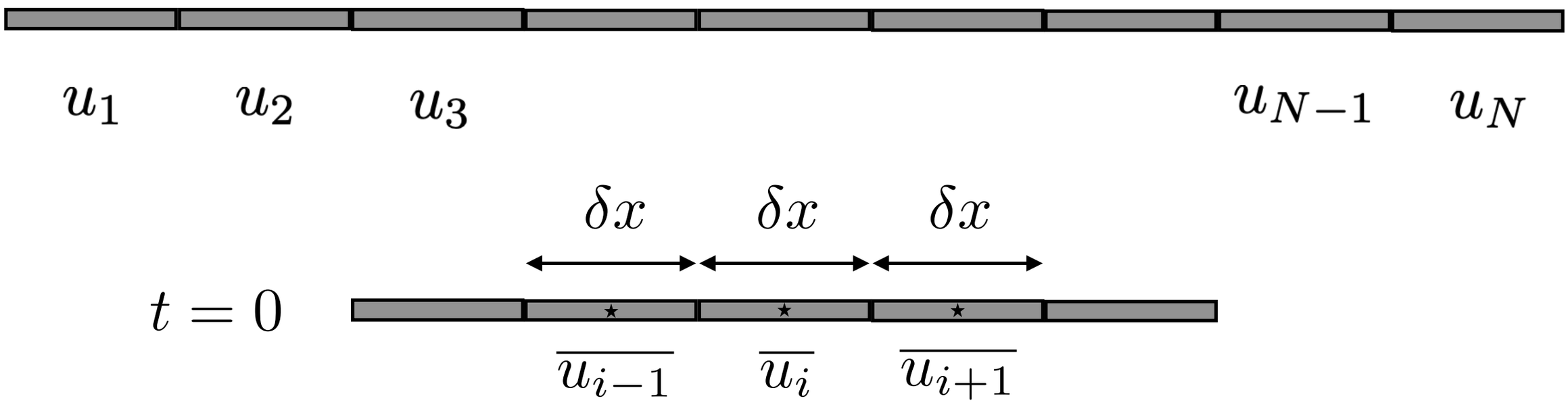




だんだん．．

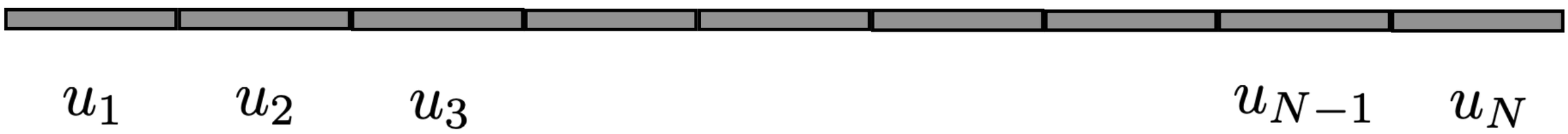
最終的には．．

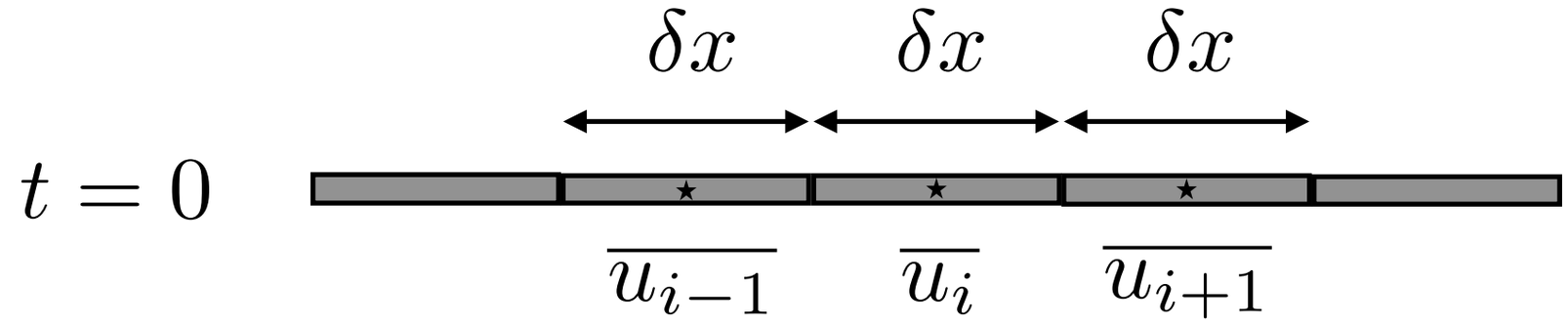
となりそう．

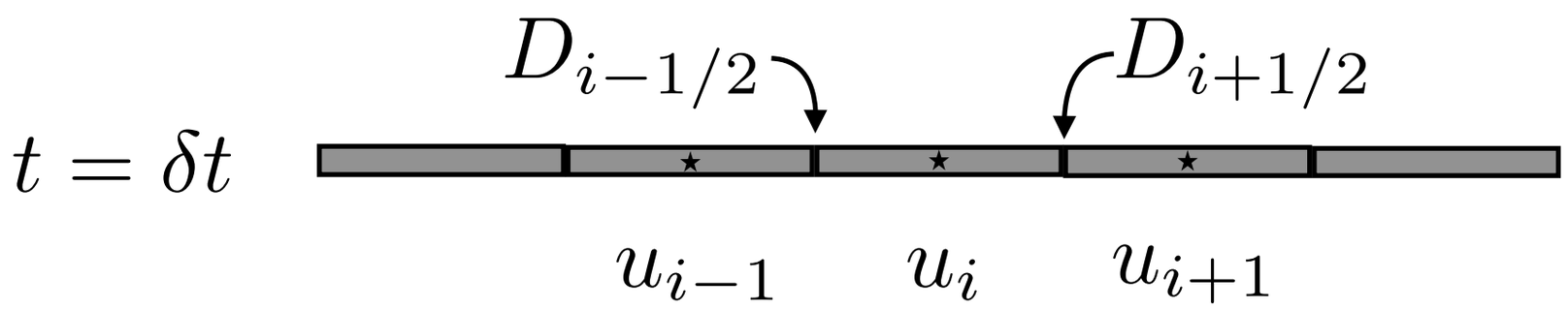


Fickの法則：温度の勾配に比例して物理量が伝播する.

$$u_i * \delta x - \overline{u_i} * \delta x = \delta t \left(D_{i+1/2}(\frac{\overline{u_{i+1}} - \overline{u_i}}{\delta x}) - D_{i-1/2}(\frac{\overline{u_i} - \overline{u_{i-1}}}{\delta x}) \right)$$







?度　　４度　　10度　４度　　４度　　6度

$$u_i * \delta x - \overline{u_i} * \delta x = \delta t \left(D_{i+1/2}(\frac{\overline{u_{i+1}} - \overline{u_i}}{\delta x}) - D_{i-1/2}(\frac{\overline{u_i} - \overline{u_{i-1}}}{\delta x}) \right)$$

正の符号　　負の符号

$$
\begin{aligned}
u_i * \delta x - \overline{u_i} * \delta x &= \delta t \left(D_{i+1/2}(\frac{\overline{u_{i+1}} - \overline{u_i}}{\delta x}) - D_{i-1/2}(\frac{\overline{u_i} - \overline{u_{i-1}}}{\delta x}) \right) \\
&= \delta t \left(D(\frac{\overline{u_{i+1}} - \overline{u_i}}{\delta x}) - D(\frac{\overline{u_i} - \overline{u_{i-1}}}{\delta x}) \right) \\
&= \delta t \frac{D}{\delta x} \left((\overline{u_{i+1}} - \overline{u_i}) - (\overline{u_i} - \overline{u_{i-1}}) \right) \\
&= \delta t \frac{D}{\delta x} \left(\overline{u_{i+1}} - 2\overline{u_i} - \overline{u_{i-1}} \right)
\end{aligned}
$$

$$
\frac{u_i - \overline{u_i}}{\delta t} = \frac{D}{\delta x^2} \left(\overline{u_{i+1}} - 2\overline{u_i} - \overline{u_{i-1}} \right)
$$

$$
\dot{u}_i = D \nabla^2 u
$$

※Fickの法則だけを使った厳密な導出方法もあります.

拡散方程式

を数値計算しよう.

$$\dot{u}_i = D\nabla^2 u$$

$$\frac{u_i - \overline{u_i}}{\delta t} = \frac{D}{\delta x^2}\left(\overline{u_{i+1}} - 2\overline{u_i} - \overline{u_{i-1}}\right)$$

$$U(x+\delta x) = U(x) + \frac{U'(x)}{1!}\delta x + \frac{U''(x)}{2!}\delta x^2 + \frac{U'''(x)}{3!}\delta x^3 + O(\delta x^4)$$

$$U(x-\delta x) = U(x) - \frac{U'(x)}{1!}\delta x + \frac{U''(x)}{2!}\delta x^2 - \frac{U'''(x)}{3!}\delta x^3 + O(\delta x^4)$$

辺どうし足す

$$U(x+\delta x) + U(x-\delta x) = 2U(x) + U''(x)\delta x^2 + O(\delta x^4)$$

$$U''(x) = \frac{U(x-\delta x) - 2U(x) + U(x+\delta x)}{\delta x^2} + O(\delta x^2)$$

拡散方程式の差分化

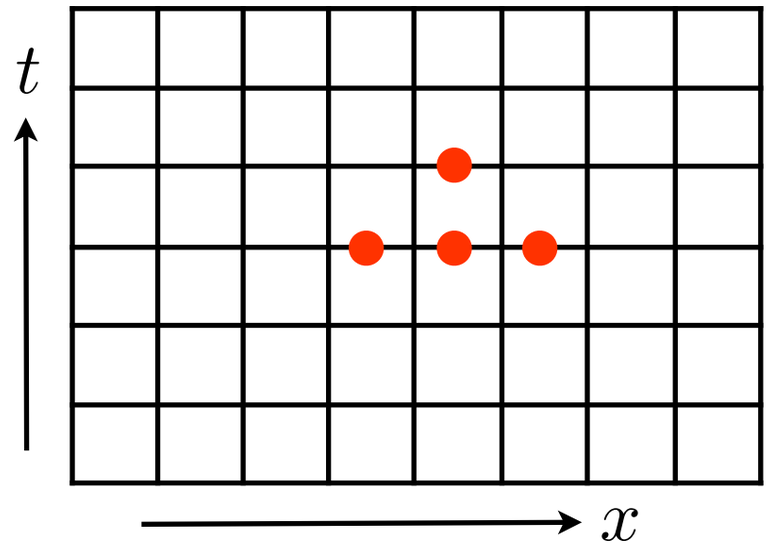
$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2}$$

時間微分は前進差分で,
空間微分は中心２階差分で

$$\frac{u(x,t+\delta t)-u(x,t)}{\delta t} \simeq D\frac{u(x-\delta x,t)-2u(x,t)+u(x+\delta x,t)}{\delta x^2}$$

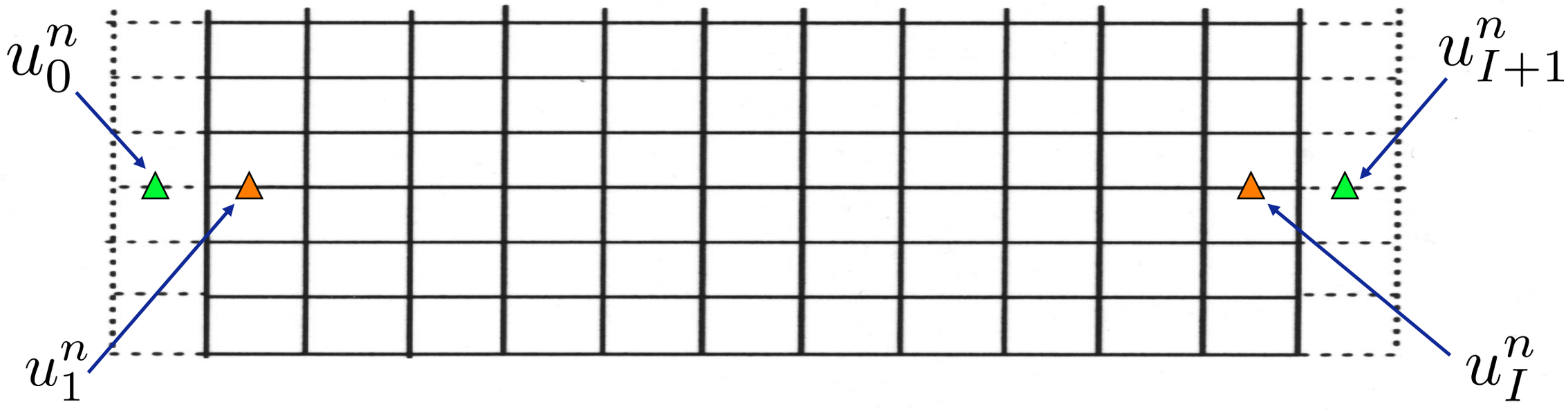
$$\frac{u_i^{n+1}-u_i^n}{\delta t} = D\frac{u_{i-1}^n-2u_i^n+u_{i+1}^n}{\delta x^2}$$

$$(i=1,2,\cdots,I;\quad n=0,1,\cdots)$$



境界条件の処理

差分のcellを両側に1つずつ広げて，そこでの値を u_0^n, u_{I+1}^n とおく．



- ディリクレ条件

$$\frac{u_0^n + u_1^n}{2} \simeq u(0, t_n) = 0, \quad \frac{u_I^n + u_{I+1}^n}{2} \simeq u(\ell, t_n) = 0$$

$$\rightarrow \quad u_0^n = -u_1^n, \; u_{I+1}^n = -u_I^n$$

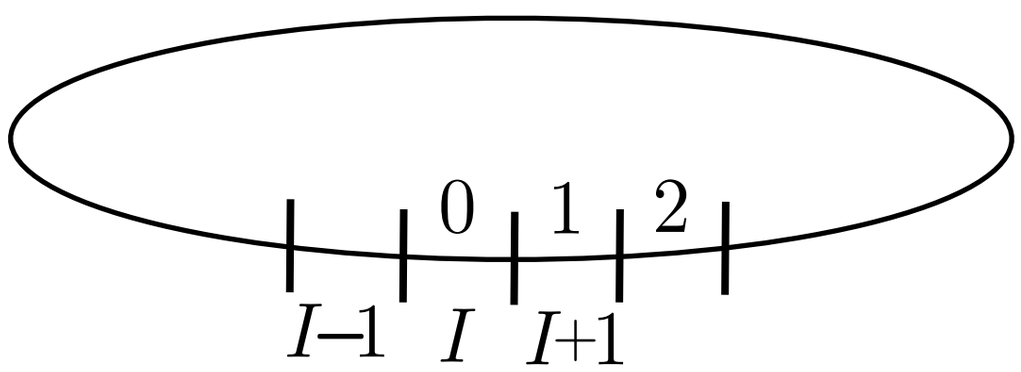
境界条件の処理

- ノイマン条件

$$\frac{u_1^n - u_0^n}{\delta x} \simeq \frac{\partial u}{\partial x}(0, t_n) = 0, \quad \frac{u_{I+1}^n - u_I^n}{\delta x} \simeq \frac{\partial u}{\partial x}(\ell, t_n) = 0$$

$$\rightarrow \quad u_0^n = u_1^n, \; u_{I+1}^n = u_I^n$$

- 周期境界条件



$$\rightarrow \quad u_0^n = u_I^n, \; u_{I+1}^n = u_1^n$$

初期値境界値問題の計算スキーム

拡散方程式

$$\frac{u_i^{n+1} - u_i^n}{\delta t} = D \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\delta x^2}$$

$$(i = 1, 2, \cdots, I; \quad n = 0, 1, \cdots)$$

境界条件

ディリクレ条件　$u_0^n = -u_1^n,\ u_{I+1}^n = -u_I^n$

ノイマン条件　$u_0^n = u_1^n,\ u_{I+1}^n = u_I^n$　$(n = 0, 1, \cdots)$

周期境界条件　$u_0^n = u_I^n,\ u_{I+1}^n = u_1^n$

初期条件

$$u_i^0 = f(x_i) \qquad (i = 1, 2, \cdots, I)$$

初期値境界値問題の計算スキーム

拡散方程式

$$\frac{u_i^{n+1} - u_i^n}{\delta t} = D \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\delta x^2}$$

$$(i = 1, 2, \cdots, I; \quad n = 0, 1, \cdots)$$

境界条件

ディリクレ条件　$u_0^n = -u_1^n,\ u_{I+1}^n = -u_I^n$

ノイマン条件　$u_0^n = u_1^n,\ u_{I+1}^n = u_I^n$　$(n = 0, 1, \cdots)$

周期境界条件　$u_0^n = u_I^n,\ u_{I+1}^n = u_1^n$

初期条件

$$u_i^0 = f(x_i) \qquad (i = 1, 2, \cdots, I)$$

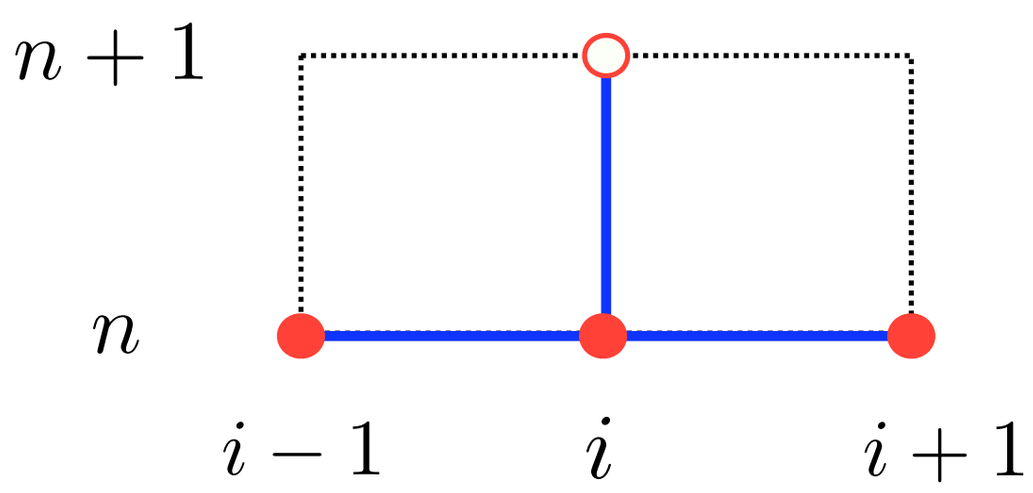
新しいステップの計算

$$\frac{u_i^{n+1} - u_i^n}{\delta t} = D \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\delta x^2}$$

$\lambda = \dfrac{D\delta t}{\delta x^2}$ とおくと

$$u_i^{n+1} = u_i^n + \lambda(u_{i-1}^n - 2u_i^n + u_{i+1}^n)$$

→ 陽解法(Explicit scheme)



初期値境界値問題の数値計算

6_Diffuse1Dim.cをDLして開こう.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define     Imax        (100)
#define     PI          (3.14159265358979)
#define     L           (2.0 * PI)
#define     INTV        (10)

double f(double x)
{
    double ans;
    ans = cos(3 * x);
    //ans = rand() / ((double)RAND_MAX);
    return ans;
}
int main(void)
{
    int i,i_time;
    FILE *gp;
    gp = popen("gnuplot -persist","w");
    //fprintf(gp,"set terminal x11\n");
    fprintf(gp,"set terminal aqua\n");
    fprintf(gp,"set xrange[0:%f]\n",L);
    fprintf(gp,"set yrange[-2:2]\n");
    double  u[Imax + 2];
    double  u_New[Imax + 2];
    double x,y;
    double dx = L / Imax;
    double dt = 0.0001;
    double D = 0.01;
    double lambda = D * dt / (dx * dx);

    for(i = 1;i <= Imax;i ++)
    {
        x = (i - 0.5) * dx;
        u[i] = f(x);
    }
```

```c
for(i_time = 0;;i_time ++)
    {
        if(1)
        {
            double sum = 0.0;
            for(i = 1;i <= Imax;i ++)
            {
                sum += u[i] * dx;
            }
            printf("%15.15f\n",sum);
        }
         //B.C.
         u[0] = u[1];
         u[Imax + 1] = u[Imax];

         //plot
        if(i_time % INTV == 0)
        {
            fprintf(gp,"plot '-' with lines\n");
            for(i = 1;i <= Imax;i ++)
            {
                x = (i - 0.5) * dx;
                y = u[i];
                fprintf(gp,"%f %f\n",x,y);
            }
            fprintf(gp,"e\n");
            fflush(gp);
        }
         //Calc Eq
         for(i = 1;i <= Imax;i ++)
         {
             u_New[i] = u[i] + lambda * (u[i - 1] - 2 *
u[i] + u[i + 1]);
         }

         //Subs
         for(i = 1;i <= Imax;i ++)
         {
             u[i] = u_New[i];
         }
```

初期値境界値問題の数値計算

6_Diffuse1Dim.cをDLして開こう.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define     Imax        (100)
#define     PI          (3.14159265358979)
#define     L           (2.0 * PI)
#define     INTV        (10)

double f(double x)
{
    double ans;
    ans = cos(3 * x);
    //ans = rand() / ((double)RAND_MAX);
    return ans;
}
int main(void)
{
    int i,i_time;
    FILE *gp;
    gp = popen("gnuplot -persist","w");
    //fprintf(gp,"set terminal x11\n");
    fprintf(gp,"set terminal aqua\n");
    fprintf(gp,"set xrange[0:%f]\n",L);
    fprintf(gp,"set yrange[-2:2]\n");
    double  u[Imax + 2];
    double  u_New[Imax + 2];
    double x,y;
    double dx = L / Imax;
    double dt = 0.0001;
    double D = 0.01;
    double lambda = D * dt / (dx * dx);

    for(i = 1;i <= Imax;i ++)
    {
        x = (i - 0.5) * dx;
        u[i] = f(x);
    }
```

```c
for(i_time = 0;;i_time ++)
    {
        if(1)
        {
            fprintf(gp,"plot '-' with lines\n");
            for(i = 1;i <= Imax;i ++)
            {
                x = (i - 0.5) * dx;
                y = u[i];
                fprintf(gp,"%f %f\n",x,y);
            }
            fprintf(gp,"e\n");
            fflush(gp);
        }
        //Calc Eq
        for(i = 1;i <= Imax;i ++)
        {
            u_New[i] = u[i] + lambda * (u[i - 1] - 2 *
u[i] + u[i + 1]);
        }

        //Subs
        for(i = 1;i <= Imax;i ++)
        {
            u[i] = u_New[i];
        }
```

Imax　箱の数

L 領域の長さ

INTV 画面表示の間引き間隔

初期値境界値問題の数値計算

6_Diffuse1Dim.cをDLして開こう.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define     Imax        (100)
#define     PI          (3.14159265358979)
#define     L           (2.0 * PI)
#define     INTV        (10)

double f(double x)
{
    double ans;
    ans = cos(3 * x);
    //ans = rand() / ((double)RAND_MAX);
    return ans;
}
int main(void)
{
    int i,i_time;
    FILE *gp;
    gp = popen("gnuplot -persist","w");
    //fprintf(gp,"set terminal x11\n");
    fprintf(gp,"set terminal aqua\n");
    fprintf(gp,"set xrange[0:%f]\n",L);
    fprintf(gp,"set yrange[-2:2]\n");
    double  u[Imax + 2];
    double  u_New[Imax + 2];
    double x,y;
    double dx = L / Imax;
    double dt = 0.0001;
    double D = 0.01;
    double lambda = D * dt / (dx * dx);

    for(i = 1;i <= Imax;i ++)
    {
        x = (i - 0.5) * dx;
        u[i] = f(x);
    }
```

uの初期の形を決める関数

```c
for(i_time = 0;;i_time ++)
    {
        if(1)
        {
            fprintf(gp,"plot '-' with lines\n");
            for(i = 1;i <= Imax;i ++)
            {
                x = (i - 0.5) * dx;
                y = u[i];
                fprintf(gp,"%f %f\n",x,y);
            }
            fprintf(gp,"e\n");
            fflush(gp);
        }
        //Calc Eq
        for(i = 1;i <= Imax;i ++)
        {
            u_New[i] = u[i] + lambda * (u[i - 1] - 2 *
u[i] + u[i + 1]);
        }

        //Subs
        for(i = 1;i <= Imax;i ++)
        {
            u[i] = u_New[i];
        }
```

初期値境界値問題の数値計算

6_Diffuse1Dim.cをDLして開こう.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define     Imax        (100)
#define     PI          (3.14159265358979)
#define     L           (2.0 * PI)
#define     INTV        (10)

double f(double x)
{
    double ans;
    ans = cos(3 * x);
    //ans = rand() / ((double)RAND_MAX);
    return ans;
}
int main(void)
{
    int i,i_time;
    FILE *gp;
    gp = popen("gnuplot -persist","w");
    //fprintf(gp,"set terminal x11\n");
    fprintf(gp,"set terminal aqua\n");
    fprintf(gp,"set xrange[0:%f]\n",L);
    fprintf(gp,"set yrange[-2:2]\n");
    double  u[Imax + 2];
    double  u_New[Imax + 2];
    double x,y;
    double dx = L / Imax;
    double dt = 0.0001;
    double D = 0.01;
    double lambda = D * dt / (dx * dx);

    for(i = 1;i <= Imax;i ++)
    {
        x = (i - 0.5) * dx;
        u[i] = f(x);
    }
```

Gnuplotのいつものおまじない

```c
for(i_time = 0;;i_time ++)
    {
        if(1)
        {
            fprintf(gp,"plot '-' with lines\n");
            for(i = 1;i <= Imax;i ++)
            {
                x = (i - 0.5) * dx;
                y = u[i];
                fprintf(gp,"%f %f\n",x,y);
            }
            fprintf(gp,"e\n");
            fflush(gp);
        }
        //Calc Eq
        for(i = 1;i <= Imax;i ++)
        {
            u_New[i] = u[i] + lambda * (u[i - 1] - 2 *
u[i] + u[i + 1]);
        }

        //Subs
        for(i = 1;i <= Imax;i ++)
        {
            u[i] = u_New[i];
        }
```

初期値境界値問題の数値計算

6_Diffuse1Dim.cをDLして開こう.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define     Imax        (100)
#define     PI          (3.14159265358979)
#define     L           (2.0 * PI)
#define     INTV        (10)

double f(double x)
{
    double ans;
    ans = cos(3 * x);
    //ans = rand() / ((double)RAND_MAX);
    return ans;
}
int main(void)
{
    int i,i_time;
    FILE *gp;
    gp = popen("gnuplot -persist","w");
    //fprintf(gp,"set terminal x11\n");
    fprintf(gp,"set terminal aqua\n");
    fprintf(gp,"set xrange[0:%f]\n",L);
    fprintf(gp,"set yrange[-2:2]\n");
    double  u[Imax + 2];
    double  u_New[Imax + 2];
    double x,y;
    double dx = L / Imax;
    double dt = 0.0001;
    double D = 0.01;
    double lambda = D * dt / (dx * dx);

    for(i = 1;i <= Imax;i ++)
    {
        x = (i - 0.5) * dx;
        u[i] = f(x);
    }
```

uは今の値を格納する.

u_Newは次の時刻の値を格納する.

サイズが+2多いのは境界条件を処理するため.

```c
for(i_time = 0;;i_time ++)
    {
        if(1)
        {
            fprintf(gp,"plot '-' with lines\n");
            for(i = 1;i <= Imax;i ++)
            {
                x = (i - 0.5) * dx;
                y = u[i];
                fprintf(gp,"%f %f\n",x,y);
            }
            fprintf(gp,"e\n");
            fflush(gp);
        }
        //Calc Eq
        for(i = 1;i <= Imax;i ++)
        {
            u_New[i] = u[i] + lambda * (u[i - 1] - 2 *
u[i] + u[i + 1]);
        }

        //Subs
        for(i = 1;i <= Imax;i ++)
        {
            u[i] = u_New[i];
        }
```

初期値境界値問題の数値計算

6_Diffuse1Dim.cをDLして開こう.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define     Imax        (100)
#define     PI          (3.14159265358979)
#define     L           (2.0 * PI)
#define     INTV        (10)

double f(double x)
{
    double ans;
    ans = cos(3 * x);
    //ans = rand() / ((double)RAND_MAX);
    return ans;
}
int main(void)
{
    int i,i_time;
    FILE *gp;
    gp = popen("gnuplot -persist","w");
    //fprintf(gp,"set terminal x11\n");
    fprintf(gp,"set terminal aqua\n");
    fprintf(gp,"set xrange[0:%f]\n",L);
    fprintf(gp,"set yrange[-2:2]\n");
    double  u[Imax + 2];
    double  u_New[Imax + 2];
    double x,y;
    double dx = L / Imax;
    double dt = 0.0001;
    double D = 0.01;
    double lambda = D * dt / (dx * dx);

    for(i = 1;i <= Imax;i ++)
    {
        x = (i - 0.5) * dx;
        u[i] = f(x);
    }
```

空間刻みdxはLをImaxで割った数.

dtは時間刻みの値.

Dは拡散定数

```c
for(i_time = 0;;i_time ++)
    {
        if(1)
        {
            fprintf(gp,"plot '-' with lines\n");
            for(i = 1;i <= Imax;i ++)
            {
                x = (i - 0.5) * dx;
                y = u[i];
                fprintf(gp,"%f %f\n",x,y);
            }
            fprintf(gp,"e\n");
            fflush(gp);
        }
        //Calc Eq
        for(i = 1;i <= Imax;i ++)
        {
            u_New[i] = u[i] + lambda * (u[i - 1] - 2 *
u[i] + u[i + 1]);
        }

        //Subs
        for(i = 1;i <= Imax;i ++)
        {
            u[i] = u_New[i];
        }
```

初期値境界値問題の数値計算

6_Diffuse1Dim.cをDLして開こう.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define     Imax        (100)
#define     PI          (3.14159265358979)
#define     L           (2.0 * PI)
#define     INTV        (10)

double f(double x)
{
    double ans;
    ans = cos(3 * x);
    //ans = rand() / ((double)RAND_MAX);
    return ans;
}
int main(void)
{
    int i,i_time;
    FILE *gp;
    gp = popen("gnuplot -persist","w");
    //fprintf(gp,"set terminal x11\n");
    fprintf(gp,"set terminal aqua\n");
    fprintf(gp,"set xrange[0:%f]\n",L);
    fprintf(gp,"set yrange[-2:2]\n");
    double  u[Imax + 2];
    double  u_New[Imax + 2];
    double x,y;
    double dx = L / Imax;
    double dt = 0.0001;
    double D = 0.01;
    double lambda = D * dt / (dx * dx);

    for(i = 1;i <= Imax;i ++)
    {
        x = (i - 0.5) * dx;
        u[i] = f(x);
    }
```

時刻0のときの，uの形状を関数fから決めている.

```c
for(i_time = 0;;i_time ++)
    {
        if(1)
        {
            fprintf(gp,"plot '-' with lines\n");
            for(i = 1;i <= Imax;i ++)
            {
                x = (i - 0.5) * dx;
                y = u[i];
                fprintf(gp,"%f %f\n",x,y);
            }
            fprintf(gp,"e\n");
            fflush(gp);
        }
        //Calc Eq
        for(i = 1;i <= Imax;i ++)
        {
            u_New[i] = u[i] + lambda * (u[i - 1] - 2 *
u[i] + u[i + 1]);
        }

        //Subs
        for(i = 1;i <= Imax;i ++)
        {
            u[i] = u_New[i];
        }
```

初期値境界値問題の数値計算

6_Diffuse1Dim.cをDLして開こう.

今は気にしない

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i,i_time;
    FILE *gp;
    gp = popen("gnuplot -persist","w");
    //fprintf(gp,"set terminal x11\n");
    fprintf(gp,"set terminal aqua\n");
    fprintf(gp,"set xrange[0:%f]\n",L);
    fprintf(gp,"set yrange[-2:2]\n");
    double  u[Imax + 2];
    double  u_New[Imax + 2];
    double x,y;
    double dx = L / Imax;
    double dt = 0.0001;
    double D = 0.01;
    double lambda = D * dt / (dx * dx);

    for(i = 1;i <= Imax;i ++)
    {
        x = (i - 0.5) * dx;
        u[i] = f(x);
    }
```

```c
for(i_time = 0;;i_time ++)
    {
        if(1)
        {
            double sum = 0.0;
            for(i = 1;i <= Imax;i ++)
            {
                sum += u[i] * dx;
            }
            printf("%15.15f\n",sum);
        }
        //B.C.
        u[0] = u[1];
        u[Imax + 1] = u[Imax];

        //plot
        if(i_time % INTV == 0)
        {
            fprintf(gp,"plot '-' with lines\n");
            for(i = 1;i <= Imax;i ++)
            {
                x = (i - 0.5) * dx;
                y = u[i];
                fprintf(gp,"%f %f\n",x,y);
            }
            fprintf(gp,"e\n");
            fflush(gp);
        }
        //Calc Eq
        for(i = 1;i <= Imax;i ++)
        {
            u_New[i] = u[i] + lambda * (u[i - 1] - 2 *
u[i] + u[i + 1]);
        }

        //Subs
        for(i = 1;i <= Imax;i ++)
        {
            u[i] = u_New[i];
        }
```

初期値境界値問題の数値計算

6_Diffuse1Dim.cをDLして開こう.

境界条件を処理するための記述.
ここでは，ノイマン０の境界条件を仮定している.

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i,i_time;
    FILE *gp;
    gp = popen("gnuplot -persist","w");
    //fprintf(gp,"set terminal x11\n");
    fprintf(gp,"set terminal aqua\n");
    fprintf(gp,"set xrange[0:%f]\n",L);
    fprintf(gp,"set yrange[-2:2]\n");
    double  u[Imax + 2];
    double  u_New[Imax + 2];
    double x,y;
    double dx = L / Imax;
    double dt = 0.0001;
    double D = 0.01;
    double lambda = D * dt / (dx * dx);

    for(i = 1;i <= Imax;i ++)
    {
        x = (i - 0.5) * dx;
        u[i] = f(x);
    }
```

```c
for(i_time = 0;;i_time ++)
    {
        if(1)
        {
            double sum = 0.0;
            for(i = 1;i <= Imax;i ++)
            {
                sum += u[i] * dx;
            }
            printf("%15.15f\n",sum);
        }
        //B.C.
        u[0] = u[1];
        u[Imax + 1] = u[Imax];

        //plot
        if(i_time % INTV == 0)
        {
            fprintf(gp,"plot '-' with lines\n");
            for(i = 1;i <= Imax;i ++)
            {
                x = (i - 0.5) * dx;
                y = u[i];
                fprintf(gp,"%f %f\n",x,y);
            }
            fprintf(gp,"e\n");
            fflush(gp);
        }
        //Calc Eq
        for(i = 1;i <= Imax;i ++)
        {
            u_New[i] = u[i] + lambda * (u[i - 1] - 2 *
u[i] + u[i + 1]);
        }

        //Subs
        for(i = 1;i <= Imax;i ++)
        {
            u[i] = u_New[i];
        }
```

初期値境界値問題の数値計算

6_Diffuse1Dim.cをDLして開こう.

i_time をINTVで割った余りが０の時
（つまりi_timeがINTVの倍数の時）
関数uの形状をgnuplotで可視化している.

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i,i_time;
    FILE *gp;
    gp = popen("gnuplot -persist","w");
    //fprintf(gp,"set terminal x11\n");
    fprintf(gp,"set terminal aqua\n");
    fprintf(gp,"set xrange[0:%f]\n",L);
    fprintf(gp,"set yrange[-2:2]\n");
    double  u[Imax + 2];
    double  u_New[Imax + 2];
    double x,y;
    double dx = L / Imax;
    double dt = 0.0001;
    double D = 0.01;
    double lambda = D * dt / (dx * dx);

    for(i = 1;i <= Imax;i ++)
    {
        x = (i - 0.5) * dx;
        u[i] = f(x);
    }
```

```c
for(i_time = 0;;i_time ++)
    {
        if(1)
        {
            double sum = 0.0;
            for(i = 1;i <= Imax;i ++)
            {
                sum += u[i] * dx;
            }
            printf("%15.15f\n",sum);
        }
         //B.C.
         u[0] = u[1];
         u[Imax + 1] = u[Imax];

         //plot
        if(i_time % INTV == 0)
        {
            fprintf(gp,"plot '-' with lines\n")
            for(i = 1;i <= Imax;i ++)
            {
                x = (i - 0.5) * dx;
                y = u[i];
                fprintf(gp,"%f %f\n",x,y);
            }
            fprintf(gp,"e\n");
            fflush(gp);
        }
         //Calc Eq
         for(i = 1;i <= Imax;i ++)
         {
             u_New[i] = u[i] + lambda * (u[i - 1] - 2 *
u[i] + u[i + 1]);
         }

         //Subs
         for(i = 1;i <= Imax;i ++)
         {
             u[i] = u_New[i];
         }
```

初期値境界値問題の数値計算

6_Diffuse1Dim.cをDLして開こう.

スキームに従って，次の時刻の
u_Newの値を計算している.
ここがミソ.

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i,i_time;
    FILE *gp;
    gp = popen("gnuplot -persist","w");
    //fprintf(gp,"set terminal x11\n");
    fprintf(gp,"set terminal aqua\n");
    fprintf(gp,"set xrange[0:%f]\n",L);
    fprintf(gp,"set yrange[-2:2]\n");
    double  u[Imax + 2];
    double  u_New[Imax + 2];
    double x,y;
    double dx = L / Imax;
    double dt = 0.0001;
    double D = 0.01;
    double lambda = D * dt / (dx * dx);

    for(i = 1;i <= Imax;i ++)
    {
        x = (i - 0.5) * dx;
        u[i] = f(x);
    }
```

```c
for(i_time = 0;;i_time ++)
    {
        if(1)
        {
            double sum = 0.0;
            for(i = 1;i <= Imax;i ++)
            {
                sum += u[i] * dx;
            }
            printf("%15.15f\n",sum);
        }
         //B.C.
         u[0] = u[1];
         u[Imax + 1] = u[Imax];

         //plot
        if(i_time % INTV == 0)
        {
            fprintf(gp,"plot '-' with lines\n");
            for(i = 1;i <= Imax;i ++)
            {
                x = (i - 0.5) * dx;
                y = u[i];
                fprintf(gp,"%f %f\n",x,y);
            }
            fprintf(gp,"e\n");
            fflush(gp);
        }
         //Calc Eq
         for(i = 1;i <= Imax;i ++)
         {
             u_New[i] = u[i] + lambda * (u[i - 1] - 2 *
u[i] + u[i + 1]);
         }

         //Subs
         for(i = 1;i <= Imax;i ++)
         {
             u[i] = u_New[i];
         }
```

初期値境界値問題の数値計算

6_Diffuse1Dim.cをDLして開こう.

次の時間ループの計算に備えて,
u_Newの値をuにコピーしている.

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i,i_time;
    FILE *gp;
    gp = popen("gnuplot -persist","w");
    //fprintf(gp,"set terminal x11\n");
    fprintf(gp,"set terminal aqua\n");
    fprintf(gp,"set xrange[0:%f]\n",L);
    fprintf(gp,"set yrange[-2:2]\n");
    double  u[Imax + 2];
    double  u_New[Imax + 2];
    double x,y;
    double dx = L / Imax;
    double dt = 0.0001;
    double D = 0.01;
    double lambda = D * dt / (dx * dx);

    for(i = 1;i <= Imax;i ++)
    {
        x = (i - 0.5) * dx;
        u[i] = f(x);
    }
```

```c
for(i_time = 0;;i_time ++)
    {
        if(1)
        {
            double sum = 0.0;
            for(i = 1;i <= Imax;i ++)
            {
                sum += u[i] * dx;
            }
            printf("%15.15f\n",sum);
        }
         //B.C.
         u[0] = u[1];
         u[Imax + 1] = u[Imax];

         //plot
        if(i_time % INTV == 0)
        {
            fprintf(gp,"plot '-' with lines\n");
            for(i = 1;i <= Imax;i ++)
            {
                x = (i - 0.5) * dx;
                y = u[i];
                fprintf(gp,"%f %f\n",x,y);
            }
            fprintf(gp,"e\n");
            fflush(gp);
        }
         //Calc Eq
         for(i = 1;i <= Imax;i ++)
         {
             u_New[i] = u[i] + lambda * (u[i - 1] - 2 *
u[i] + u[i + 1]);
         }

         //Subs
         for(i = 1;i <= Imax;i ++)
         {
             u[i] = u_New[i];
         }
```

初期値境界値問題の数値計算

6_Diffuse1Dim.cをコンパイル＆実行しよう.

チャンレジ課題.

Lを変えてみる.

cosの3を5にしてみる.

dtを変えてみる.

Dを変えてみる.

if分の中身を変えてみる.

初期値境界値問題 (ノイマン条件)

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} \qquad \text{for } \ 0 < x < \ell,\ t > 0$$

$$\frac{\partial u}{\partial x}(0,t) = \frac{\partial u}{\partial x}(\ell,t) = 0 \quad \text{for } \ t > 0 \qquad \Rightarrow \text{B.C.}$$

$$u(x,0) = f(x) \qquad \text{for } \ 0 < x < \ell \qquad \Rightarrow \text{I.C.}$$

ノイマン境界条件は, 境界において物の出入りがないことを意味している.

$$u \ \text{の総量} \int_0^\ell u dx \ \text{は保存される.}$$

$$\frac{d}{dt}\int_0^\ell u dx = \int_0^\ell \frac{\partial u}{\partial t} dx = \int_0^\ell D\frac{\partial^2 u}{\partial x^2} dx = D\frac{\partial u}{\partial x}\Big|_{x=0}^{x=\ell} = 0$$

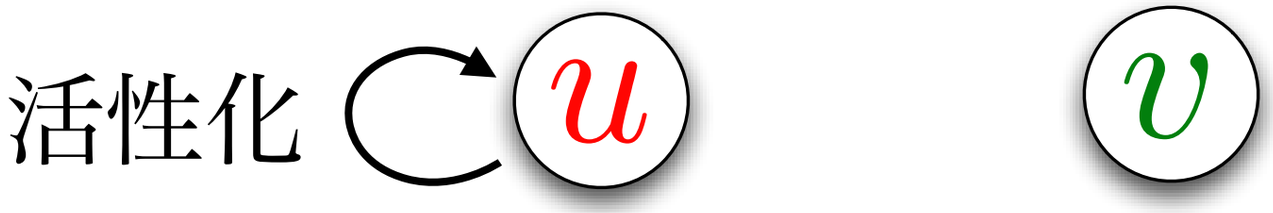
Turing Model

活性因子-抑制因子系

Activator-Inhibitor System

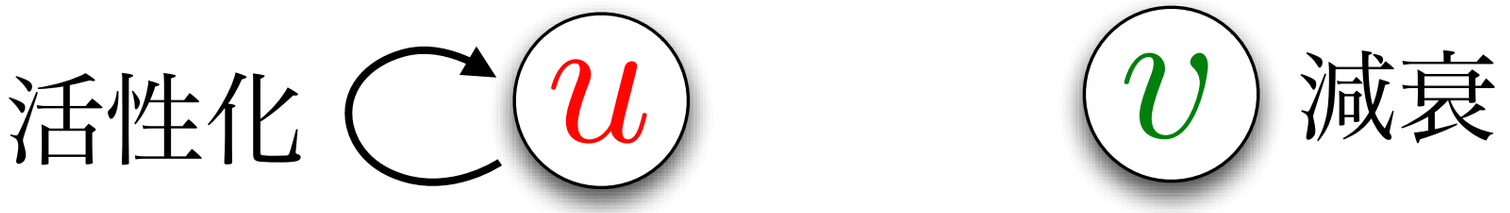
活性因子-抑制因子系

Activator-Inhibitor System



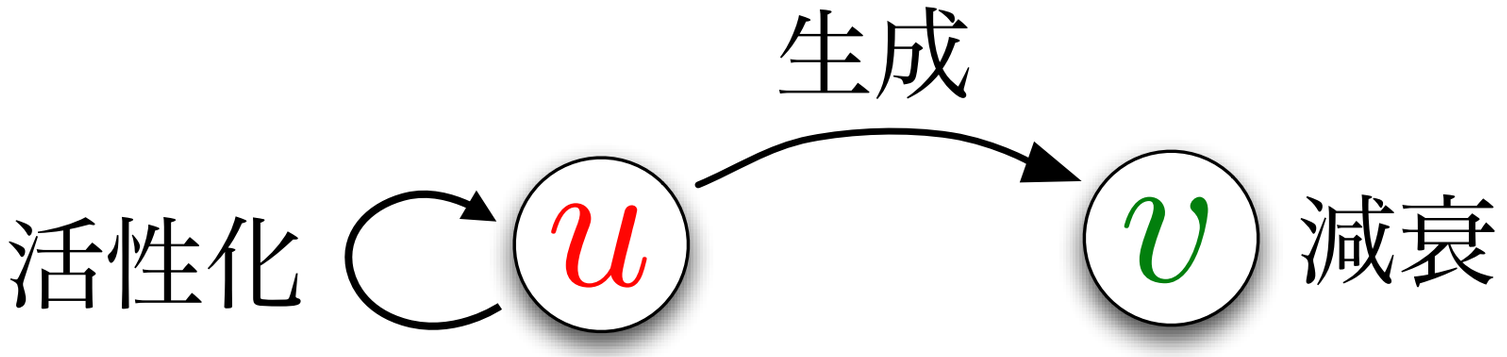
活性因子-抑制因子系

Activator-Inhibitor System



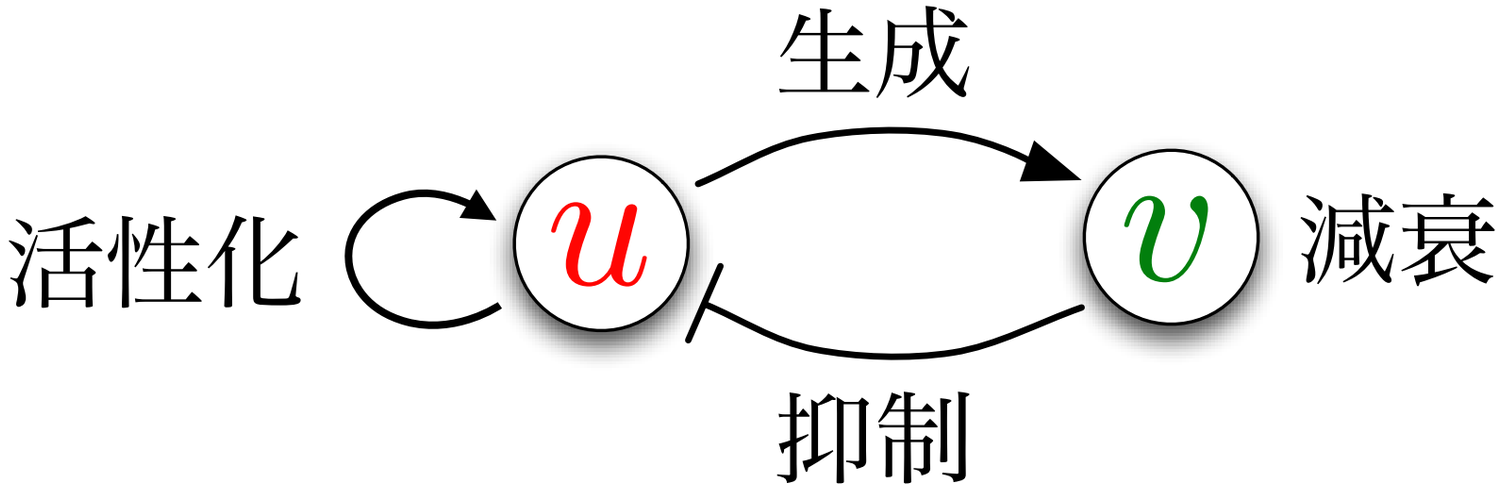
活性因子-抑制因子系

Activator-Inhibitor System



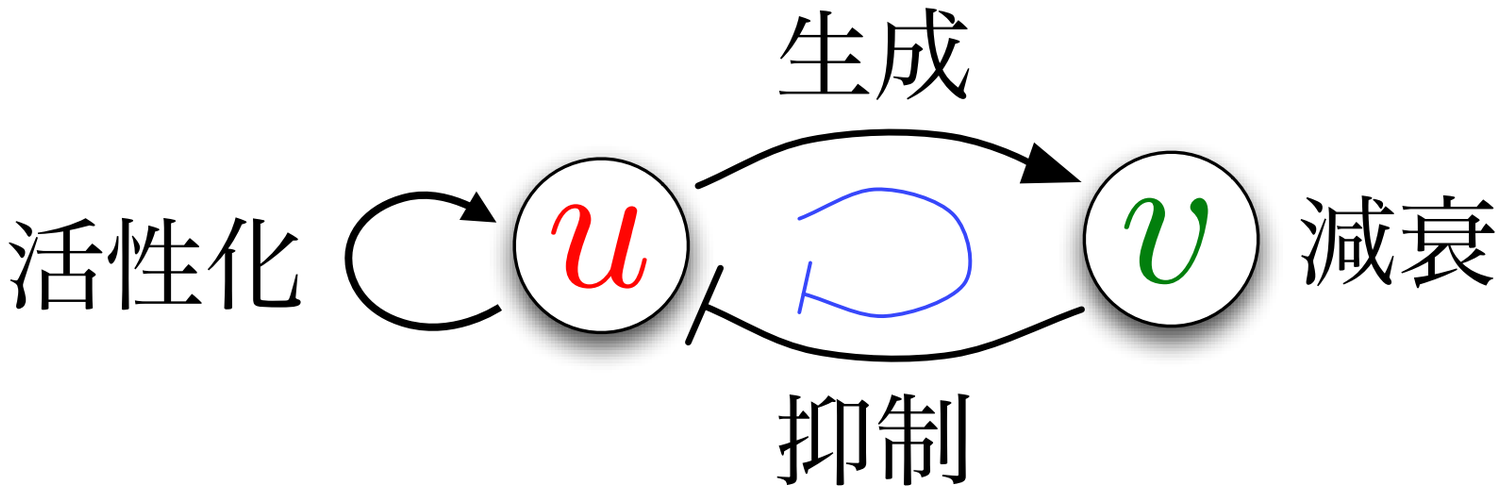
活性因子-抑制因子系

Activator-Inhibitor System



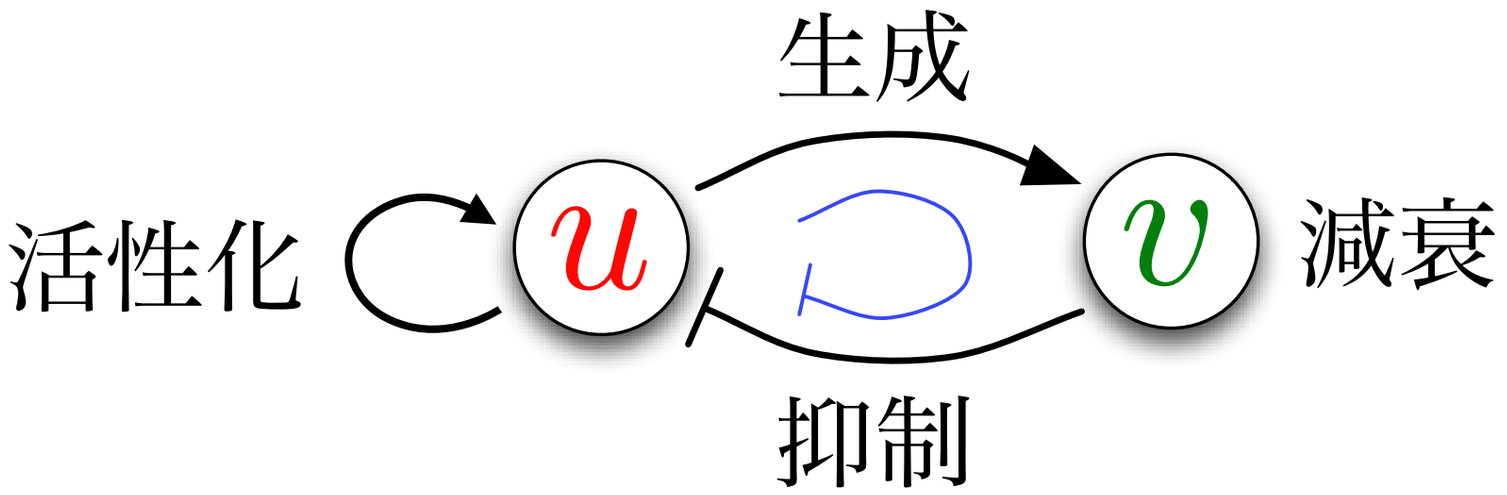
活性因子-抑制因子系

Activator-Inhibitor System



活性因子-抑制因子系

Activator-Inhibitor System

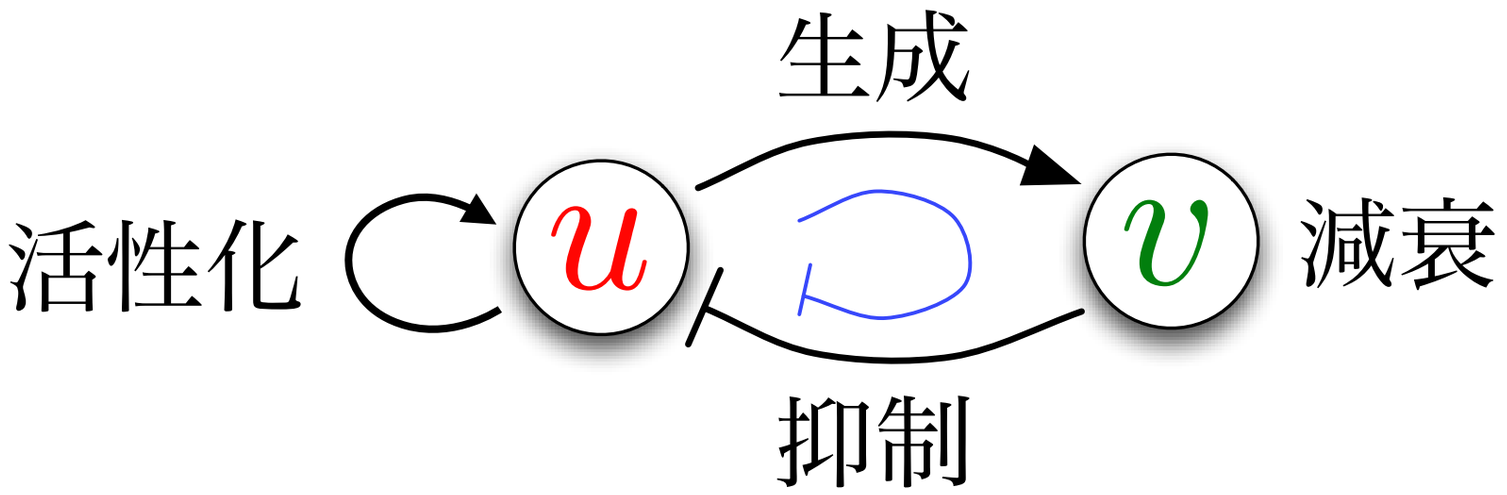


u: 活性因子

v: 抑制因子

活性因子-抑制因子系

Activator-Inhibitor System



u: 活性因子

v: 抑制因子

関係を表にすると

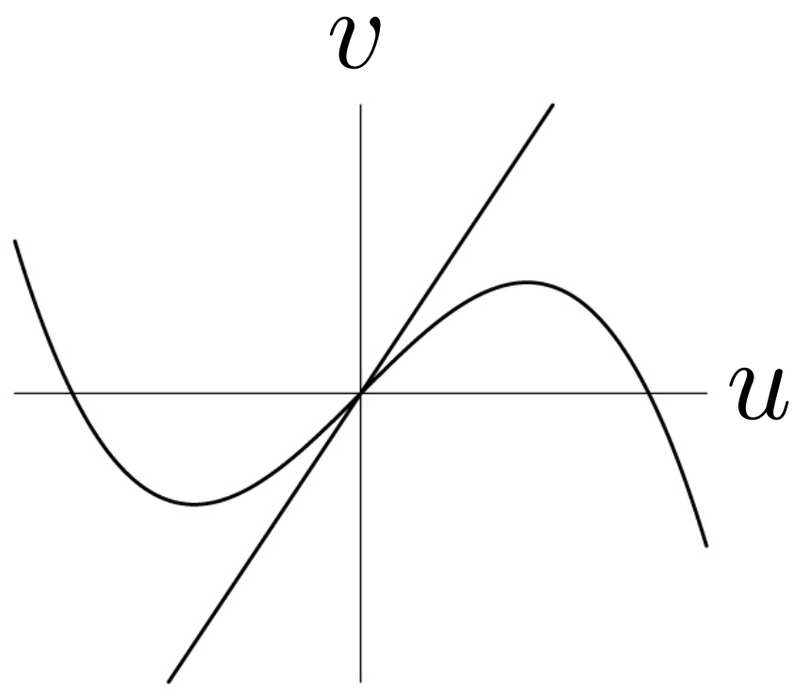
主語 / 目的語	u	v
u	＋	－
v	＋	－

$$\frac{du}{dt} = u(1 - u^2) - v$$

$$\frac{dv}{dt} = 3u - 2v$$

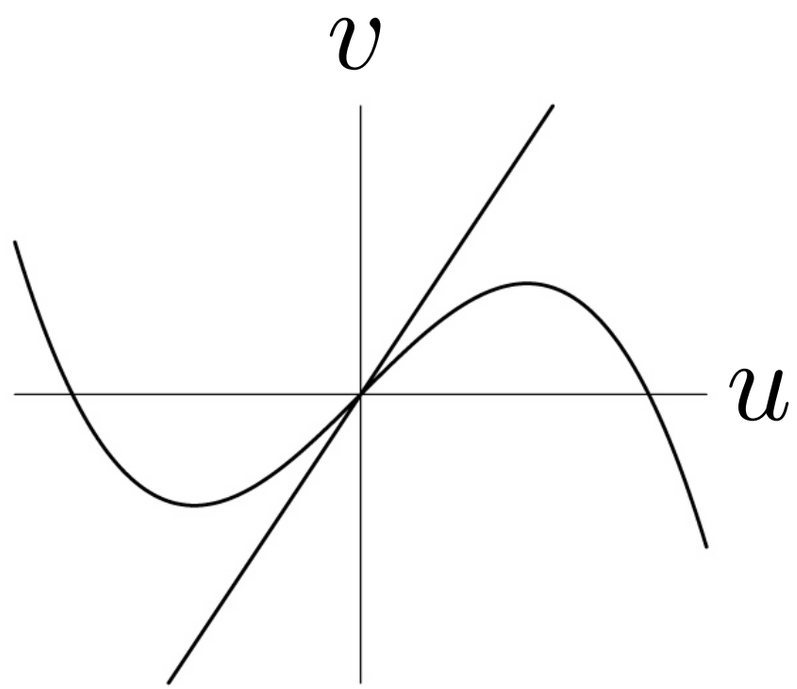
$$\frac{du}{dt} = u(1 - u^2) - v$$

$$\frac{dv}{dt} = 3u - 2v$$



$$\frac{du}{dt} = u(1-u^2) - v$$

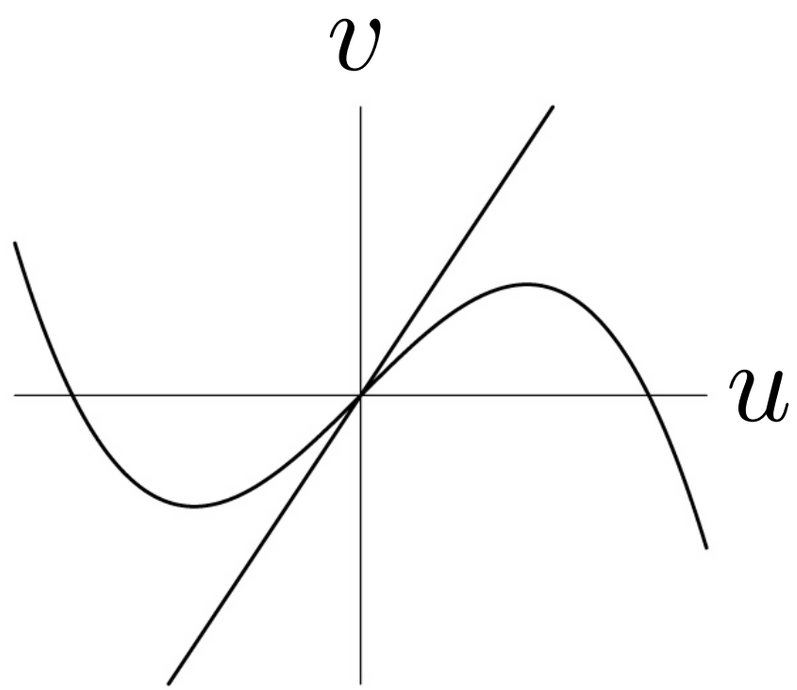
$$\frac{dv}{dt} = 3u - 2v$$



平衡点は $(0,0)$ のみ.
そこでのヤコビ行列を Aとすると,

$$\frac{du}{dt} = u(1-u^2) - v$$

$$\frac{dv}{dt} = 3u - 2v$$

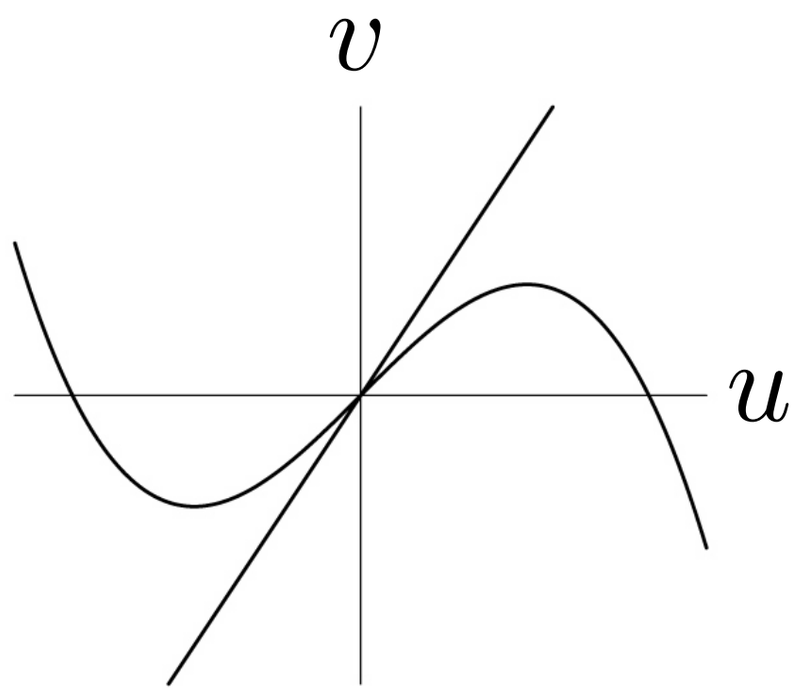


平衡点は $(0,0)$ のみ.
そこでのヤコビ行列を Aとすると,

$$A = \begin{bmatrix} 1 & -1 \\ 3 & -2 \end{bmatrix}$$

$$\frac{du}{dt} = u(1-u^2) - v$$

$$\frac{dv}{dt} = 3u - 2v$$



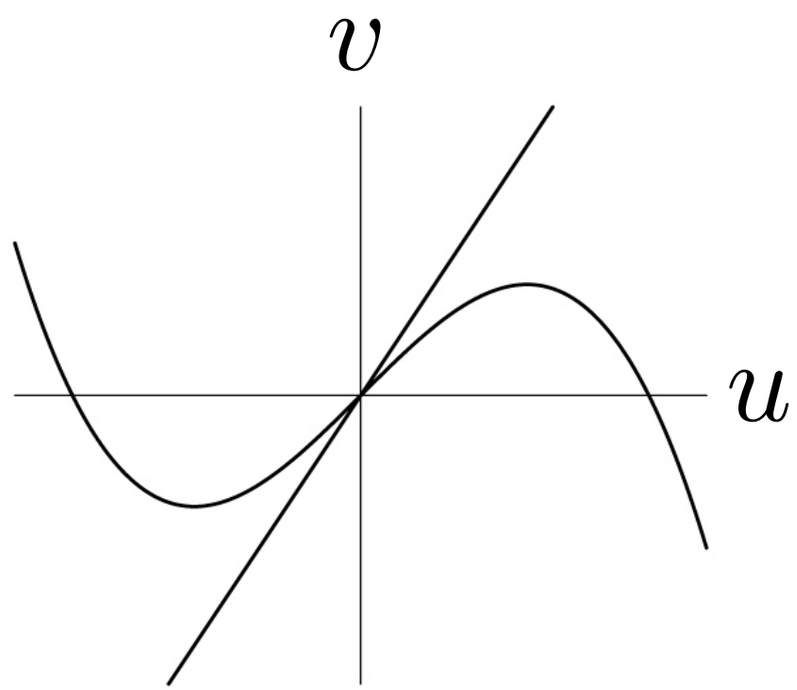
平衡点は $(0,0)$ のみ.
そこでのヤコビ行列を Aとすると,

$$A = \begin{bmatrix} 1 & -1 \\ 3 & -2 \end{bmatrix}$$

$$\mathrm{tr}A = -1 \qquad \det A = 1$$

$$\frac{du}{dt} = u(1-u^2) - v$$

$$\frac{dv}{dt} = 3u - 2v$$



平衡点は $(0,0)$ のみ.
そこでのヤコビ行列を Aとすると,

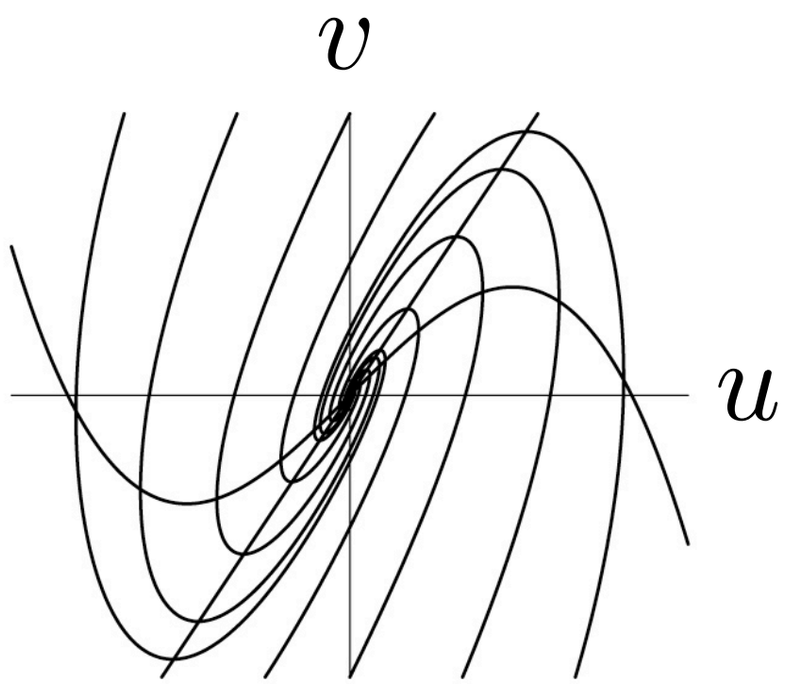
$$A = \begin{bmatrix} 1 & -1 \\ 3 & -2 \end{bmatrix}$$

$$\mathrm{tr}A = -1 \qquad \det A = 1$$

➡ $(0,0)$ は安定渦状点

$$\frac{du}{dt} = u(1-u^2) - v$$

$$\frac{dv}{dt} = 3u - 2v$$



平衡点は $(0,0)$ のみ.
そこでのヤコビ行列を Aとすると,

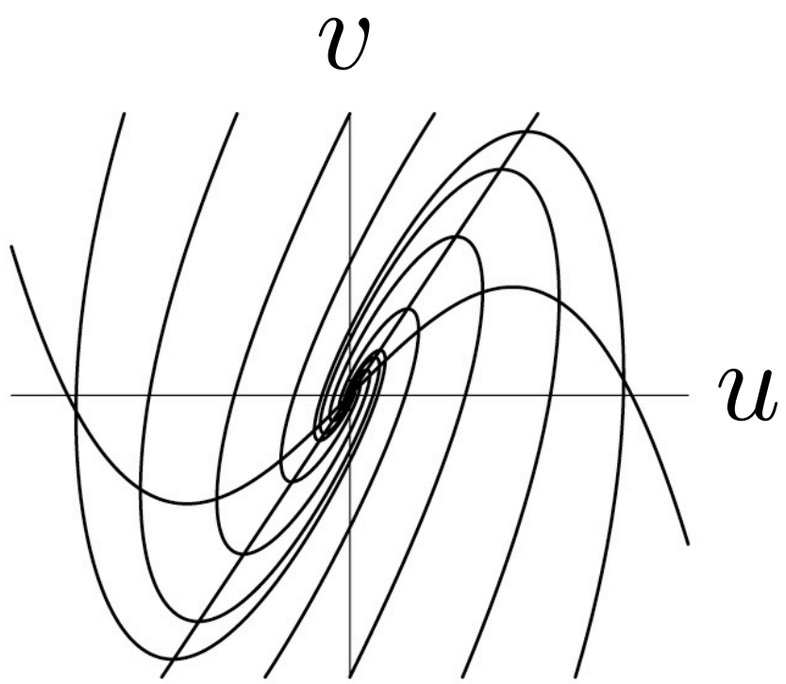
$$A = \begin{bmatrix} 1 & -1 \\ 3 & -2 \end{bmatrix}$$

$$\mathrm{tr}A = -1 \qquad \det A = 1$$

➡ $(0,0)$ は安定渦状点

$$\frac{du}{dt} = u(1-u^2) - v$$

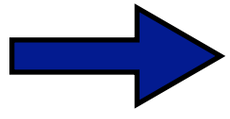
$$\frac{dv}{dt} = 3u - 2v$$



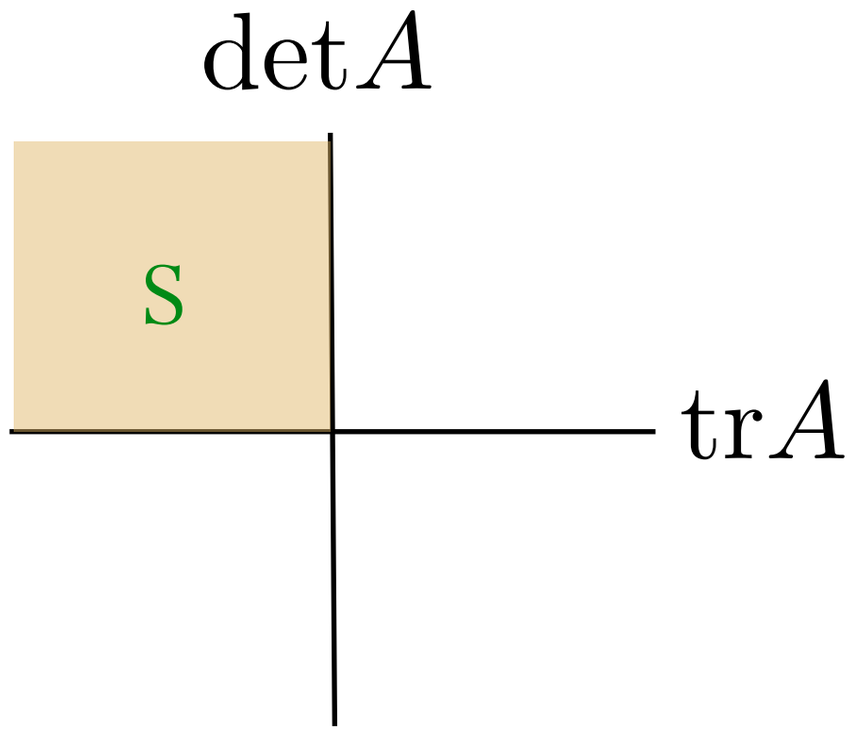
平衡点は $(0,0)$ のみ.
そこでのヤコビ行列を Aとすると,

$$A = \begin{bmatrix} 1 & -1 \\ 3 & -2 \end{bmatrix}$$

$\mathrm{tr}A = -1 \qquad \det A = 1$



$(0,0)$ は安定渦状点



反応拡散系へ

それぞれの因子が拡散すれば

$$\frac{\partial u}{\partial t} = D_u \frac{\partial^2 u}{\partial x^2} + u(1-u^2) - v$$

$$\frac{\partial v}{\partial t} = D_v \frac{\partial^2 v}{\partial x^2} + 3u - 2v \qquad \text{on } R$$

反応拡散系へ

それぞれの因子が拡散すれば

$$\frac{\partial u}{\partial t} = D_u \frac{\partial^2 u}{\partial x^2} + u(1-u^2) - v$$
$$\frac{\partial v}{\partial t} = D_v \frac{\partial^2 v}{\partial x^2} + 3u - 2v \qquad \text{on } R$$

問題： 定数定常解 $(u, v) \equiv (0, 0)$ は安定なのだろうか？

反応拡散系へ

それぞれの因子が拡散すれば

$$\frac{\partial u}{\partial t} = D_u \frac{\partial^2 u}{\partial x^2} + u(1-u^2) - v$$

$$\frac{\partial v}{\partial t} = D_v \frac{\partial^2 v}{\partial x^2} + 3u - 2v \qquad \text{on } R$$

問題： 定数定常解 $(u, v) \equiv (0, 0)$ は安定なのだろうか？

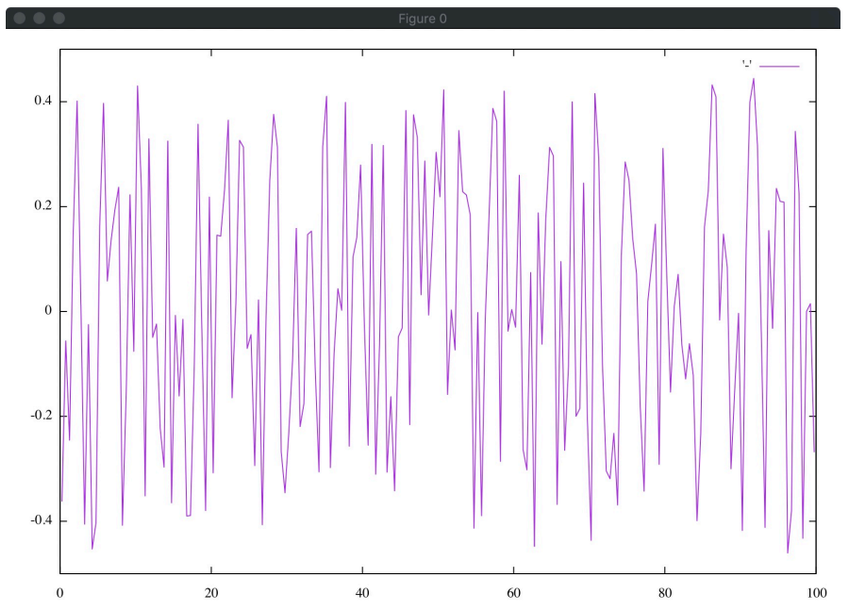
直感： $(0, 0)$ は非線形ダイナミクスのただ一つの安定解であるし、拡散は空間一様性を好むだろうから、安定に違いない！

数値計算で確かめて見よう！

$$\frac{\partial u}{\partial t} = D_u \frac{\partial^2 u}{\partial x^2} + u(1-u^2) - v$$

$$\frac{\partial v}{\partial t} = D_v \frac{\partial^2 v}{\partial x^2} + 3u - 2v$$

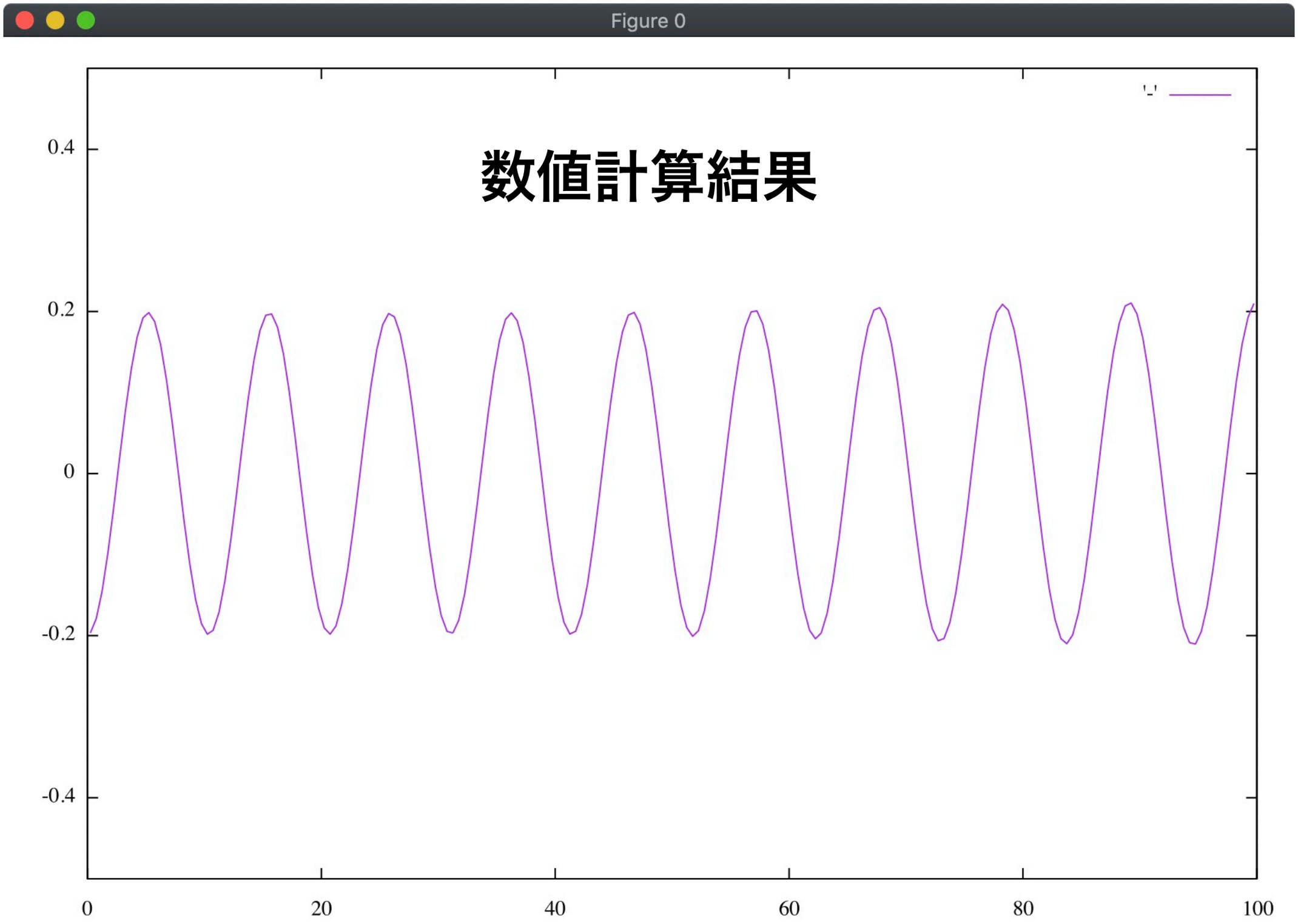
8_Turing1D.cは上記方程式の1次元のソルバー(境界条件はノイマン0)



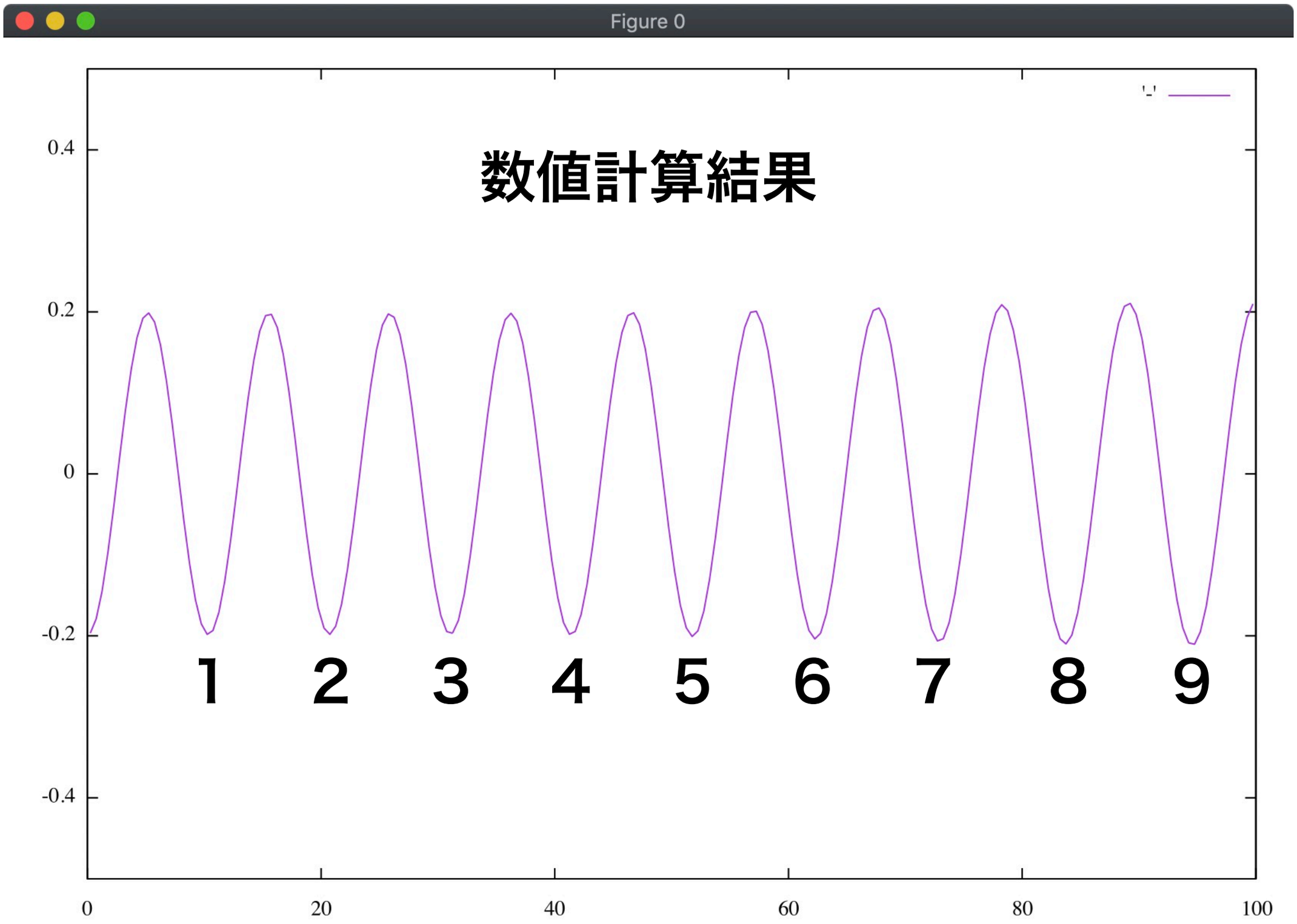
実験１：Du=Dv=1.0でパターンはどうなるかみよう！

実験２：Du=1.0, Dvを変化させてみよう！

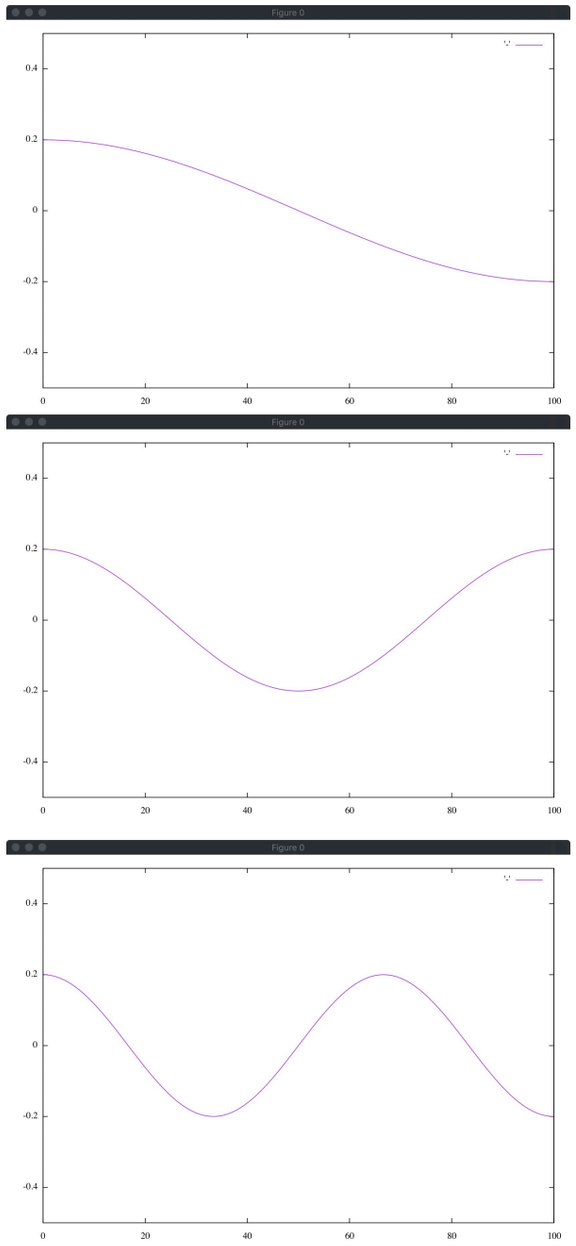
Du=1.0, Dv=10.0



Du=1.0, Dv=10.0



波の数は9.5くらい　なんでこうなるの？？



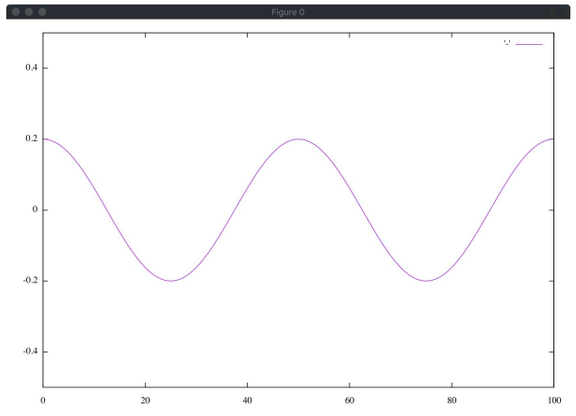
$$\cos(1 * \pi * x/L_x)$$

$$\cos(2 * \pi * x/L_x)$$

$$\cos(m * \pi * x/L_x)$$

mモード解

$$\cos(3 * \pi * x/L_x)$$



$$\cos(4 * \pi * x/L_x)$$

Lx

Lxを空間長とする.

線形化方程式導出

$$\begin{pmatrix} u(x,t) \\ v(x,t) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \delta \begin{pmatrix} \varphi(x,t) \\ \psi(x,t) \end{pmatrix} \quad |\delta| \ll 1$$

線形化方程式導出

$$\begin{pmatrix} u(x,t) \\ v(x,t) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \delta \begin{pmatrix} \varphi(x,t) \\ \psi(x,t) \end{pmatrix} \quad |\delta| \ll 1$$

$$\begin{cases} \dfrac{\partial \varphi}{\partial t} = D_u \dfrac{\partial^2 \varphi}{\partial x^2} + \varphi - \psi \\ \\ \dfrac{\partial \psi}{\partial t} = D_v \dfrac{\partial^2 \psi}{\partial x^2} + 3\varphi - 2\psi \end{cases}$$

$$\phi = e^{\lambda_k t}\cos(m * \pi * x/L_x)\phi_0$$
$$\psi = e^{\lambda_k t}\cos(m * \pi * x/L_x)\psi_0$$

mモード解を代入

(0,0)まわりの線形化方程式

行列Aの
固有値

行列A

$$k := m * \pi/L_x$$

行列の性質は固有値（と固有ベクトル）で決まる.

$$\phi = e^{\lambda_k t}\cos(m * \pi * x/L_x)\phi_0$$
$$\psi = e^{\lambda_k t}\cos(m * \pi * x/L_x)\psi_0$$

(0,0)まわりの線形化方程式

mモード解を代入

$$\lambda_k \begin{pmatrix} \varphi_0 \\ \psi_0 \end{pmatrix} = \begin{pmatrix} 1 - D_u k^2 & -1 \\ 3 & -2 - D_v k^2 \end{pmatrix} \begin{pmatrix} \varphi_0 \\ \psi_0 \end{pmatrix}$$

行列Aの固有値

行列A

$$k := m * \pi/L_x$$

行列の性質は固有値（と固有ベクトル）で決まる.

$$\begin{cases} \dfrac{\partial \varphi}{\partial t} = D_u \dfrac{\partial^2 \varphi}{\partial x^2} + \varphi - \psi \\ \dfrac{\partial \psi}{\partial t} = D_v \dfrac{\partial^2 \psi}{\partial x^2} + 3\varphi - 2\psi \end{cases}$$

(0,0)まわりの線形化方程式

$$\phi = e^{\lambda_k t} \cos(m * \pi * x / L_x)\phi_0$$
$$\psi = e^{\lambda_k t} \cos(m * \pi * x / L_x)\psi_0$$

mモード解を代入

$$\lambda_k \begin{pmatrix} \varphi_0 \\ \psi_0 \end{pmatrix} = \begin{pmatrix} 1 - D_u k^2 & -1 \\ 3 & -2 - D_v k^2 \end{pmatrix} \begin{pmatrix} \varphi_0 \\ \psi_0 \end{pmatrix}$$

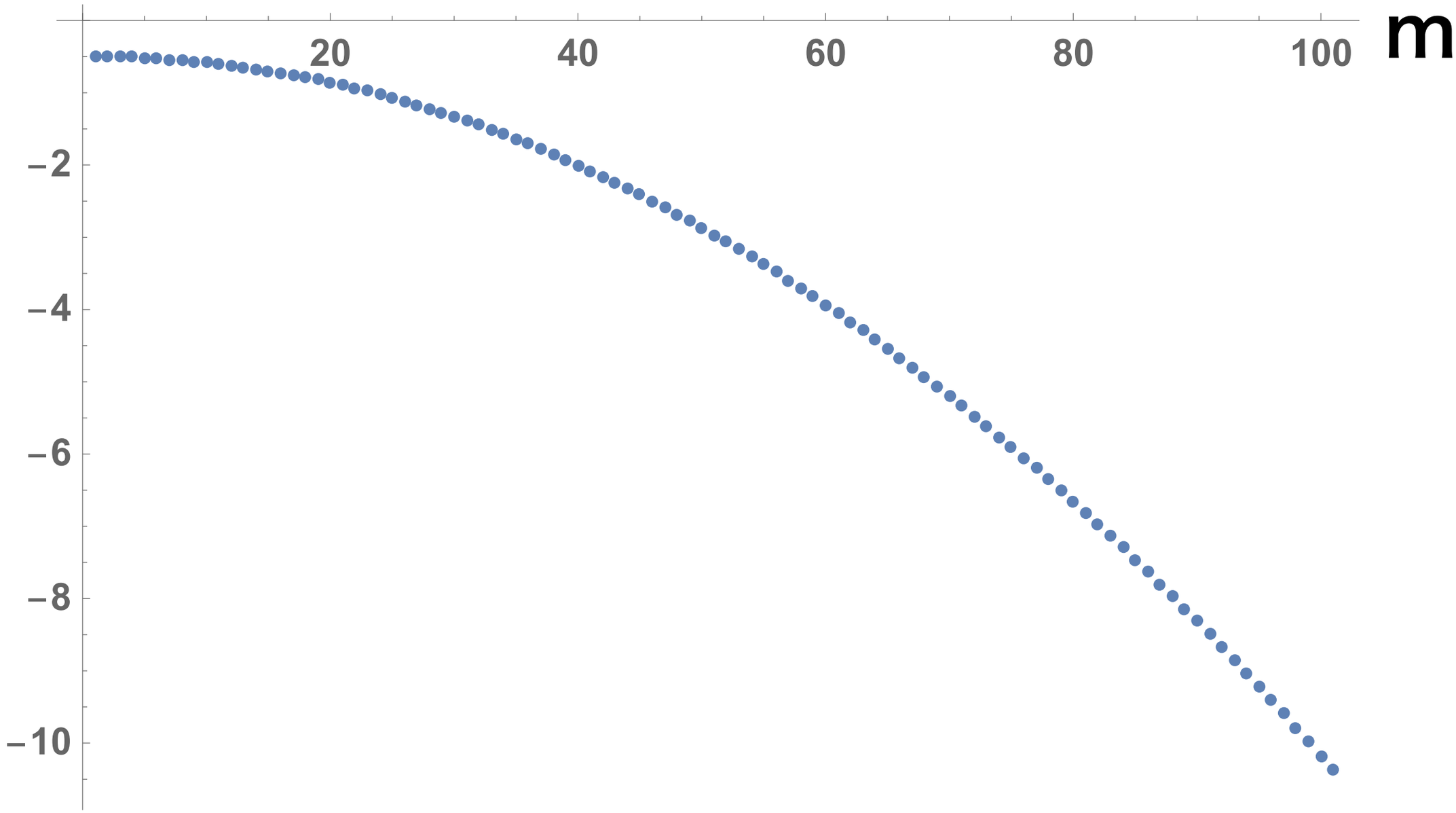
行列Aの固有値

行列A

$$k := m * \pi / L_x$$

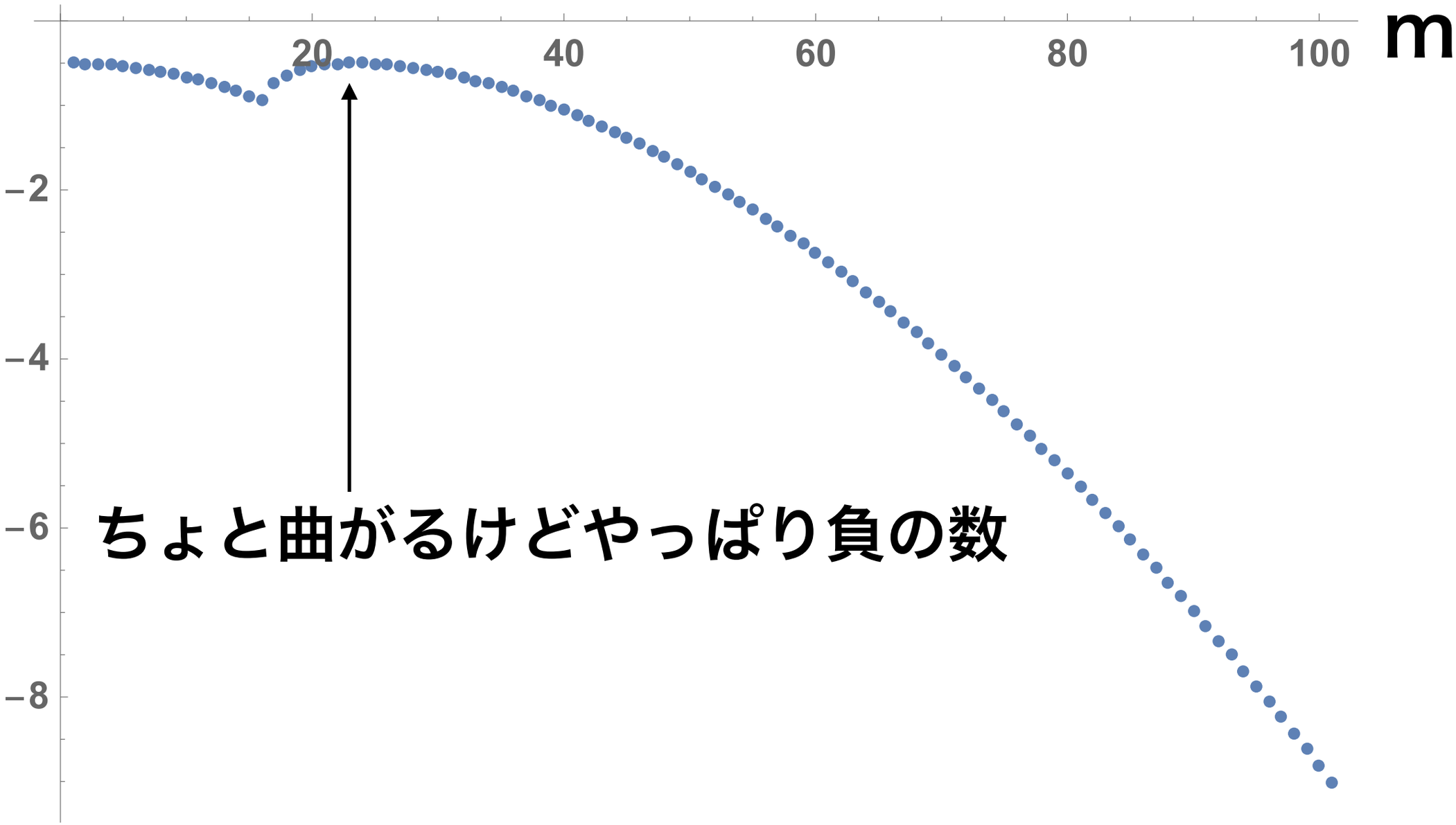
行列の性質は固有値（と固有ベクトル）で決まる.

固有値の実部が最大のもの



Du=1.0, Dv=1.0の時

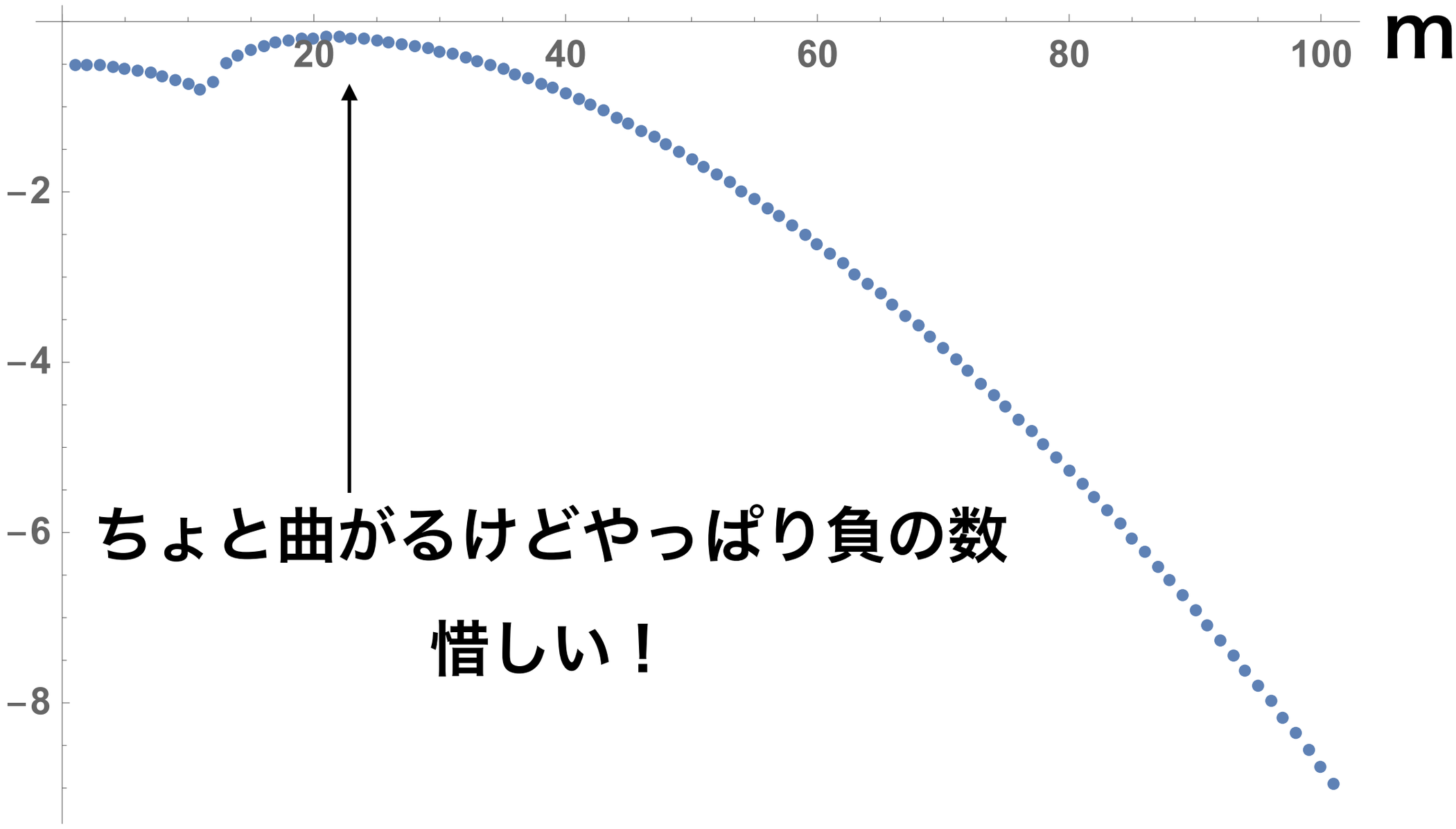
固有値の実部が最大のもの



ちょと曲がるけどやっぱり負の数

Du=1.0, Dv=3.0の時

固有値の実部が最大のもの

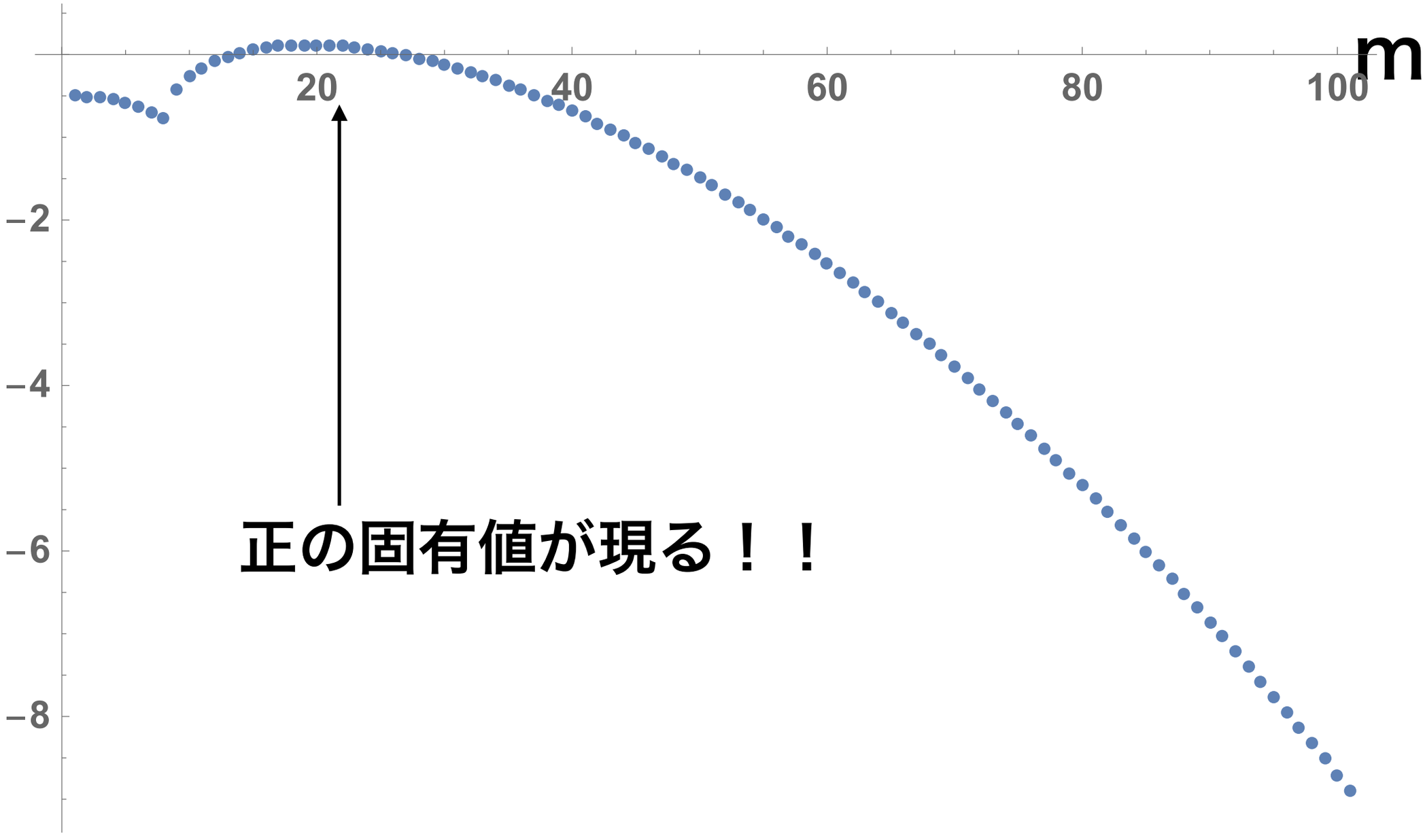


ちょと曲がるけどやっぱり負の数

惜しい！

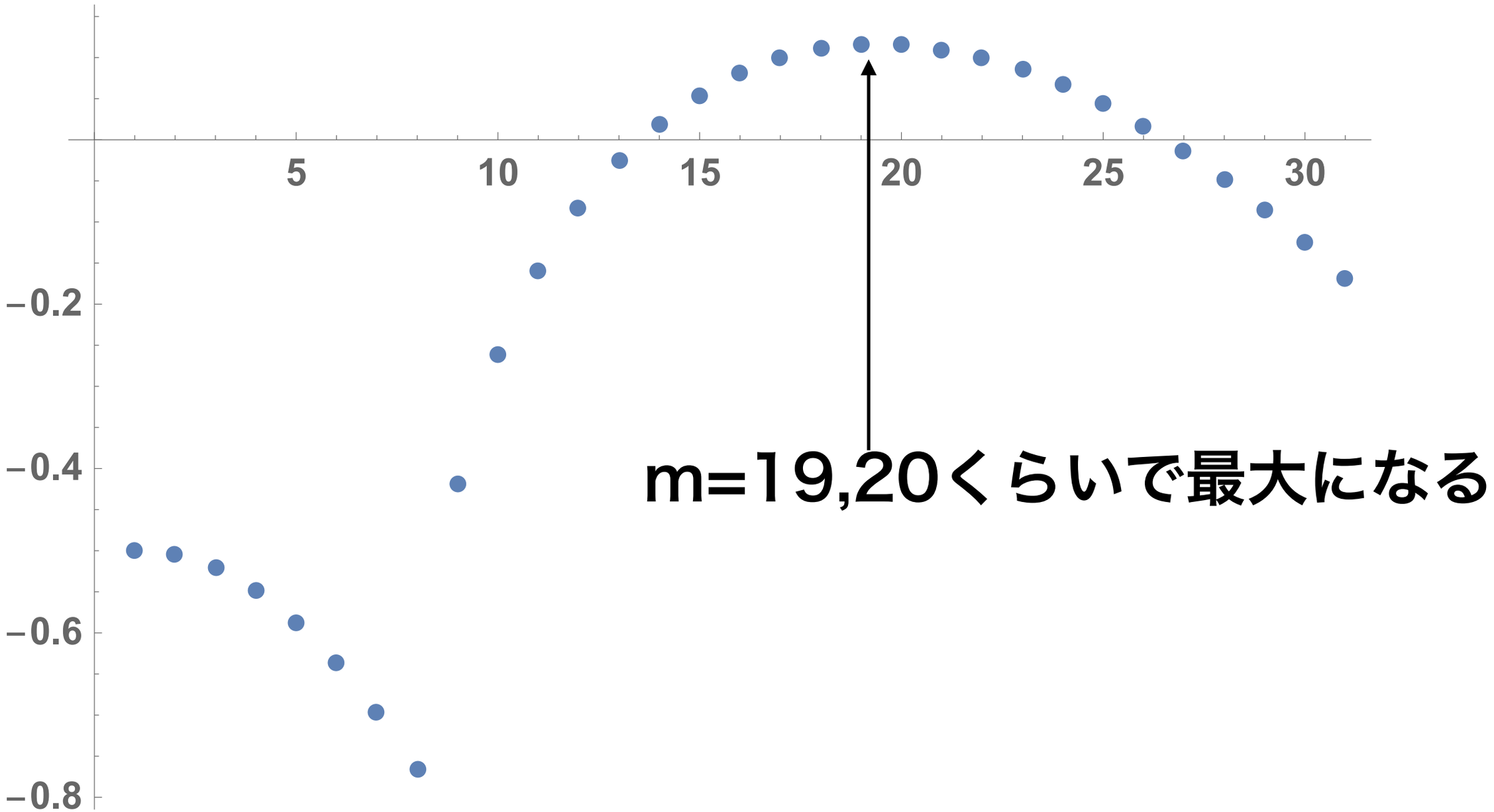
Du=1.0, Dv=5.0の時

固有値の実部が最大のもの

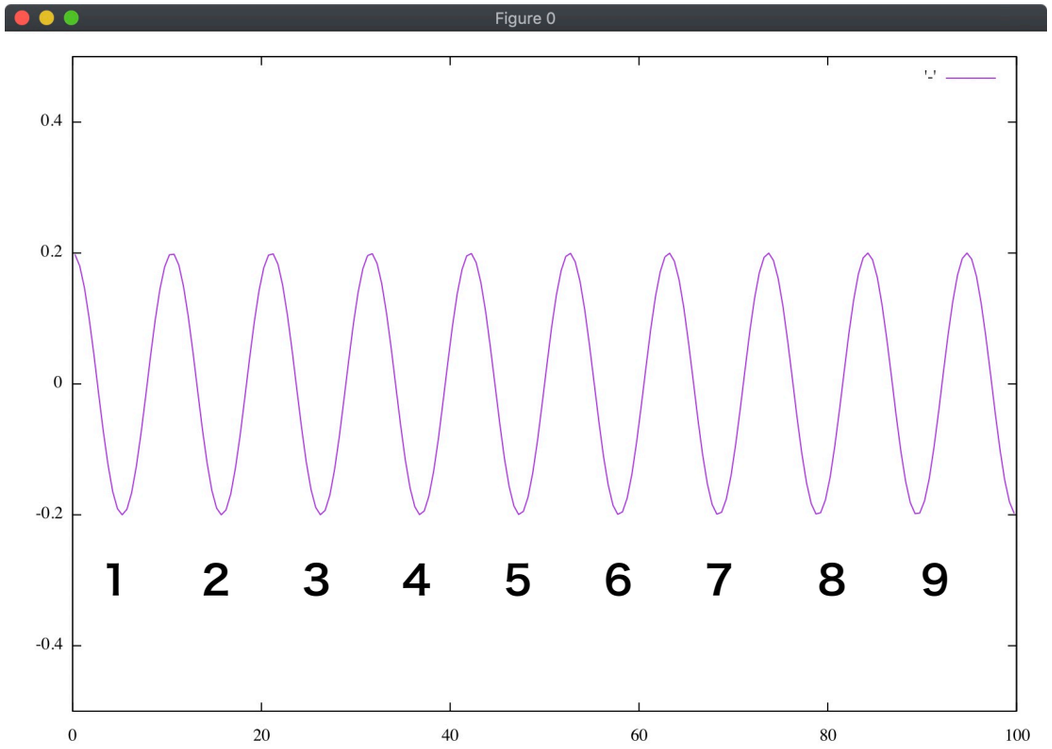


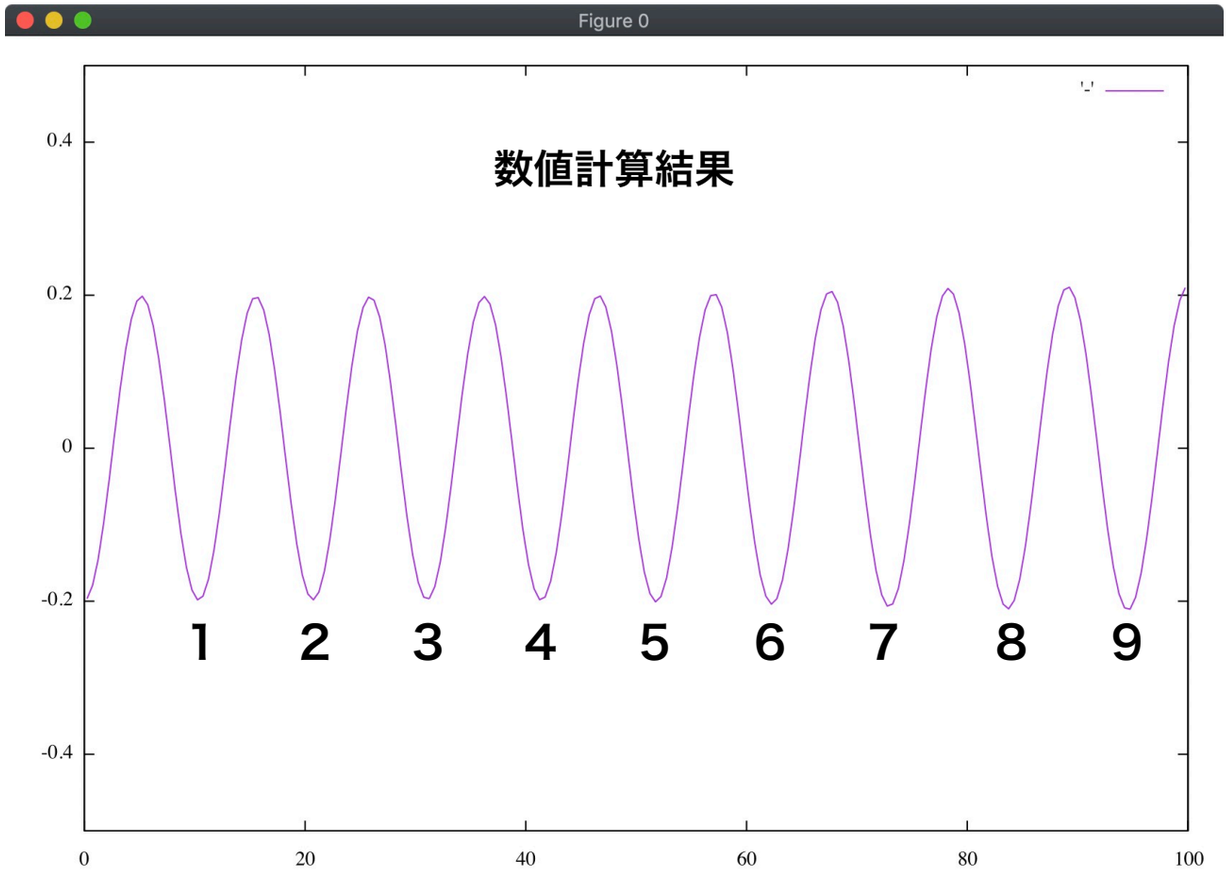
Du=1.0, Dv=10.0の時

拡大すると．．．



m=19,20くらいで最大になる





$$\cos(19 * \pi * x / L_x)$$

理論的に求めた不安定化するはずの解

以上から，Turing Instabilityの一連の解析の整合性は
数値計算からも確かめることができた.

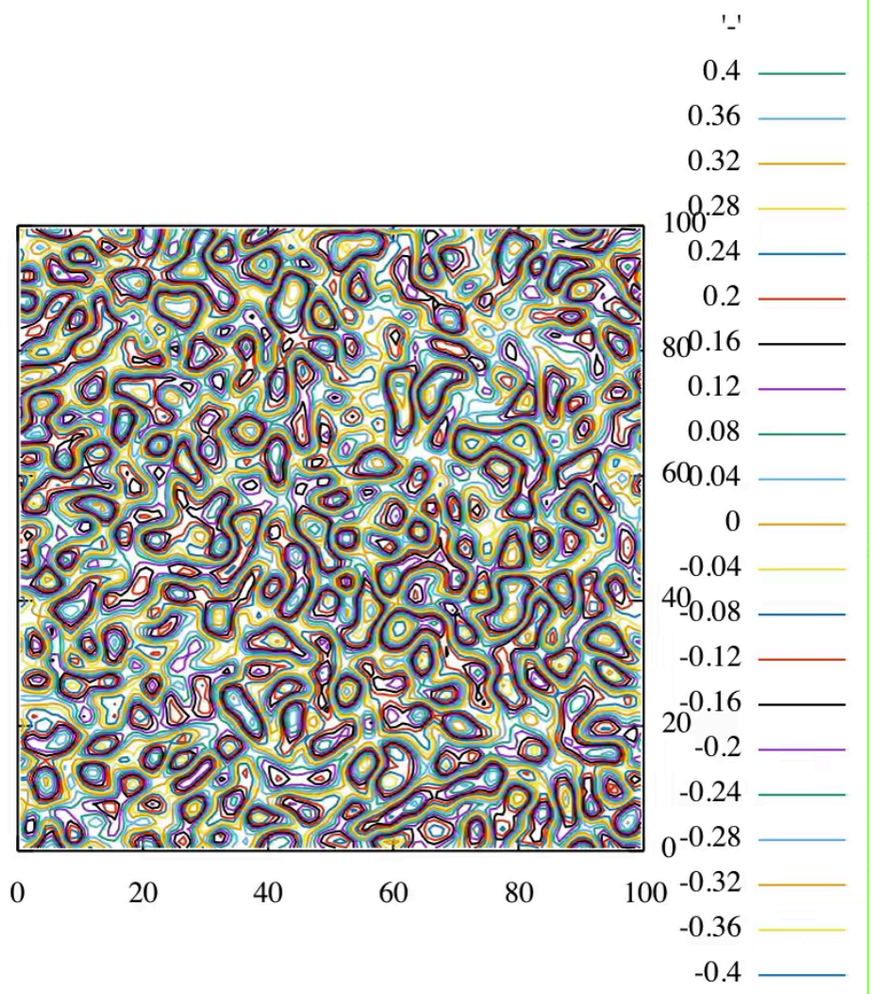
2Dシミュレーター

$$\frac{\partial u}{\partial t} = D_u \frac{\partial^2 u}{\partial x^2} + u(1 - u^2) - v$$

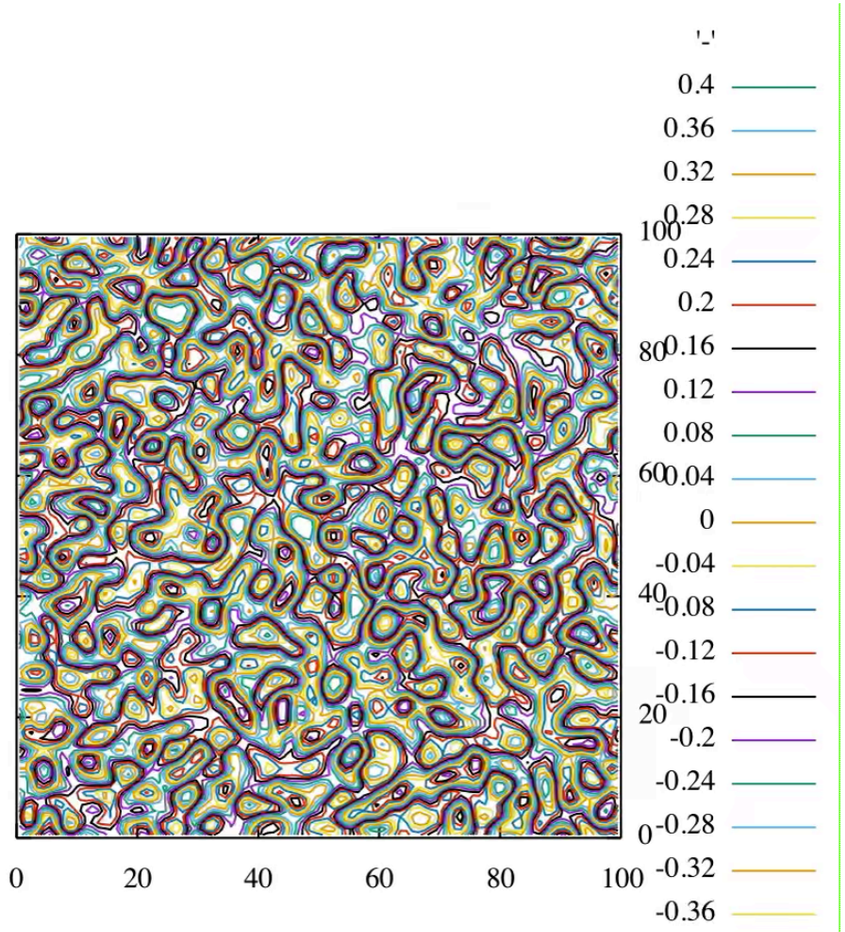
$$\frac{\partial v}{\partial t} = D_v \frac{\partial^2 v}{\partial x^2} + 3u - 2v$$

９_Turing2Dは上記方程式の２次元のソルバー(境界条件はノイマン0)

2Dシミュレーター

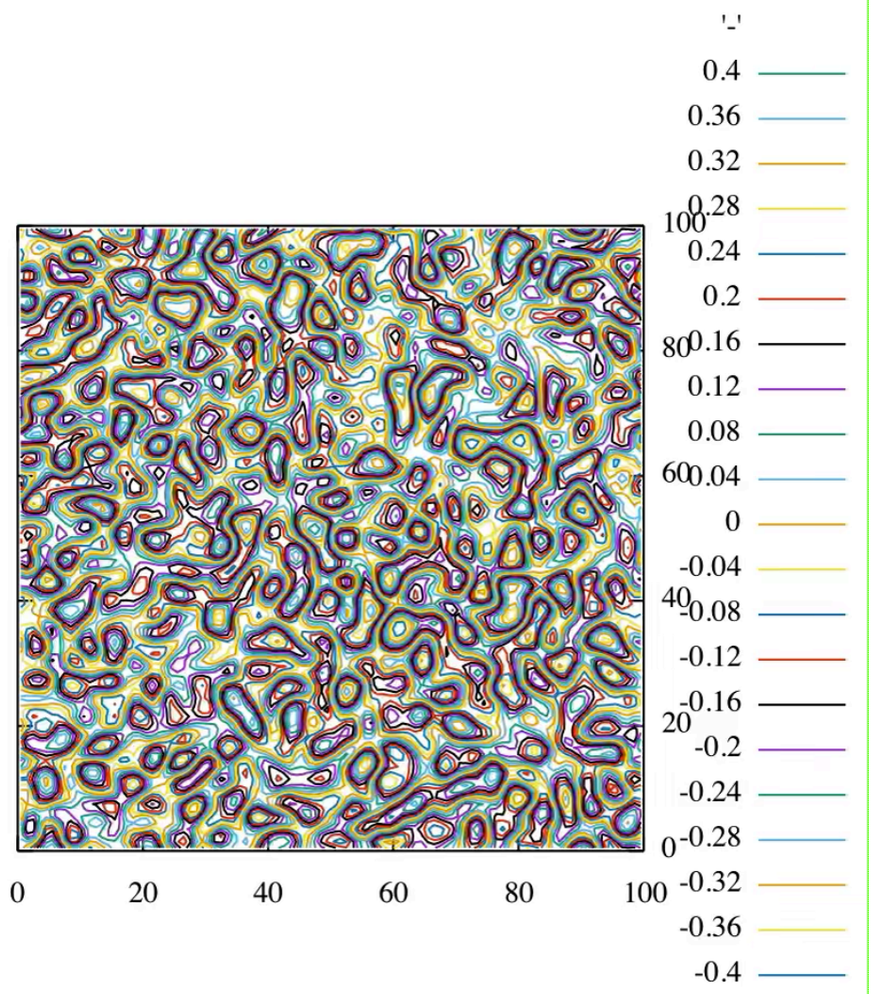


デフォルト値

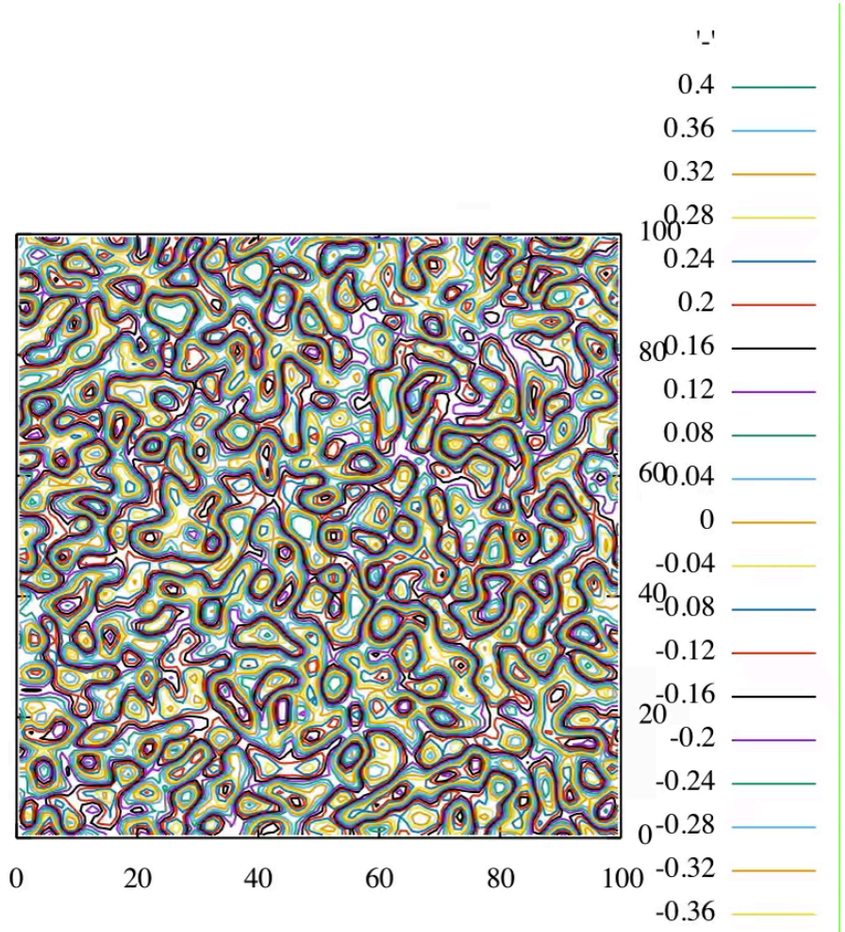


ある値で．．．．

2Dシミュレーター

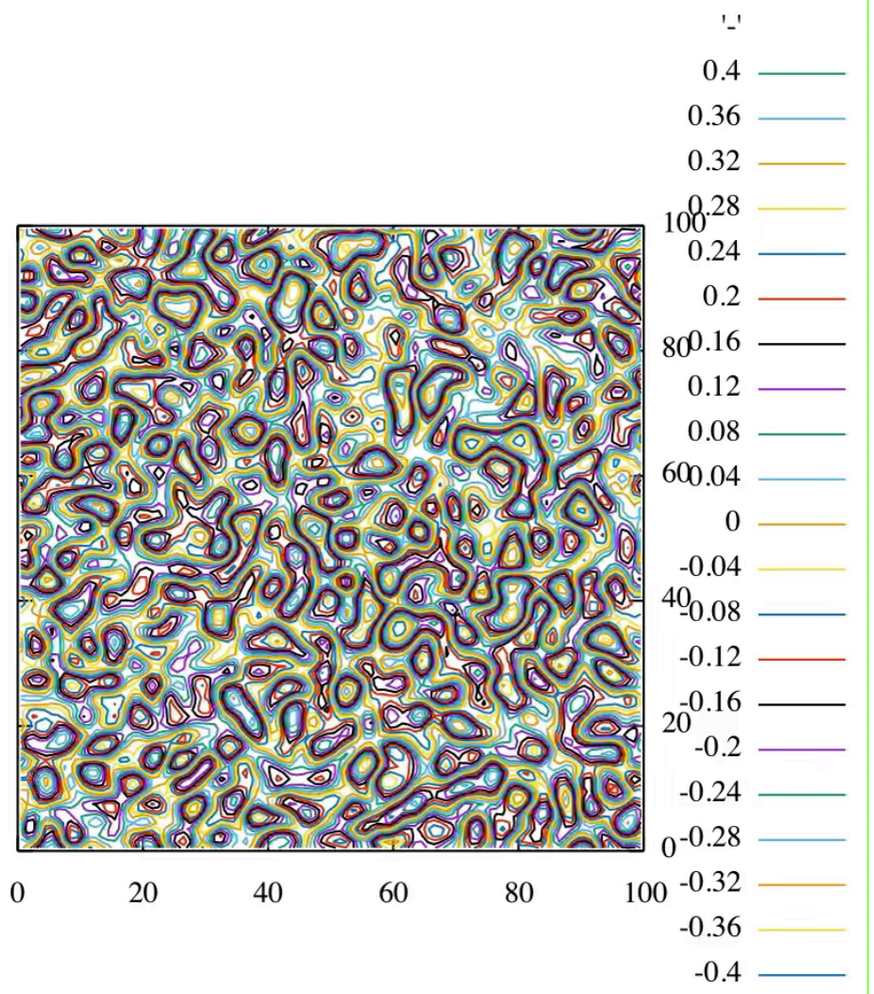


デフォルト値

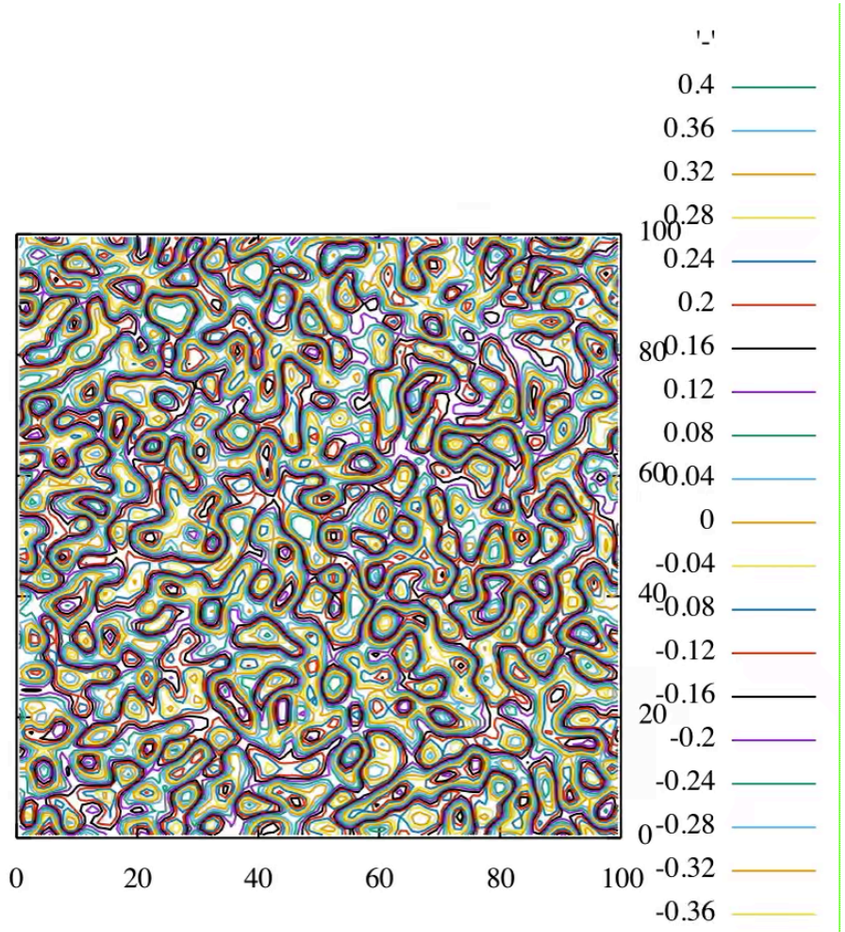


ある値で．．．．

2Dシミュレーター



デフォルト値

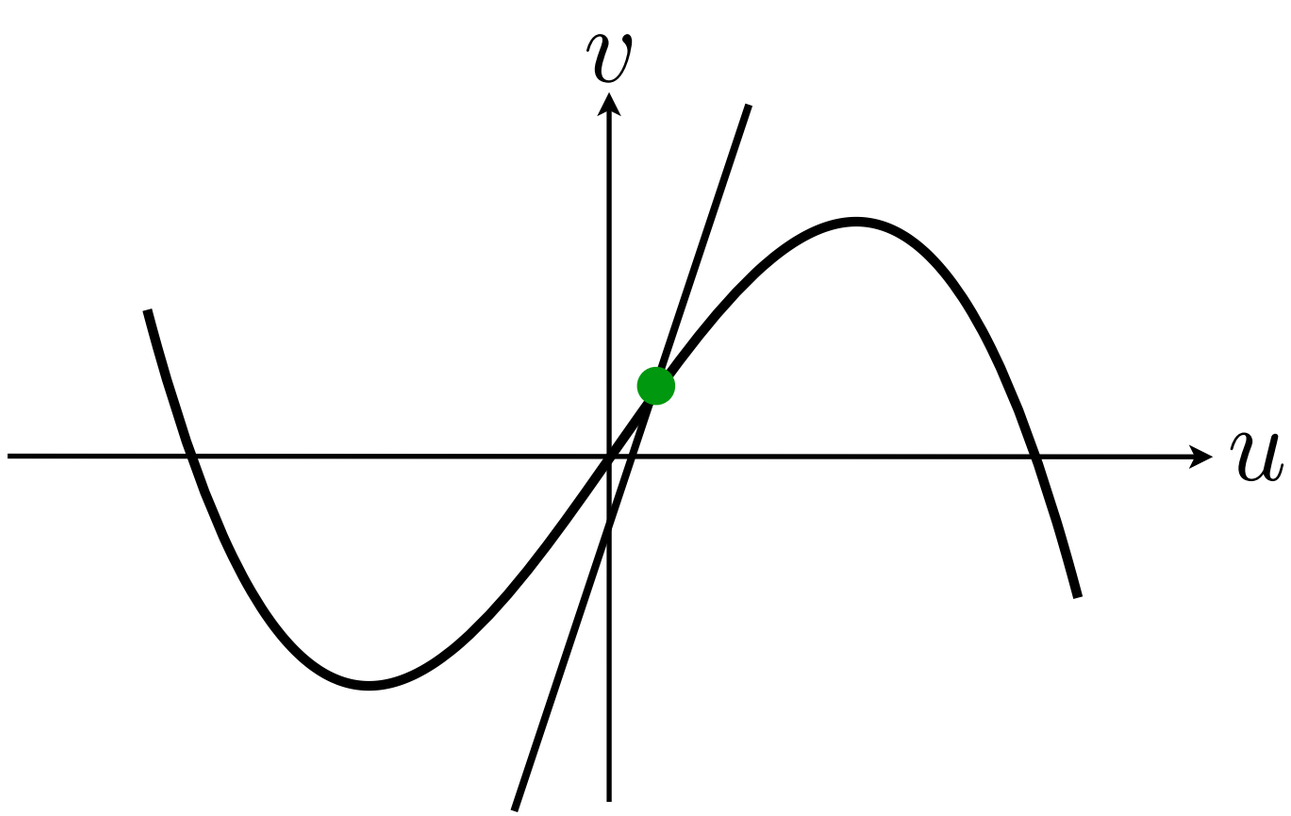


ある値で．．．．

反応項の対称性を崩す

$$\frac{\partial u}{\partial t} = D_u \frac{\partial^2 u}{\partial x^2} + u(1-u^2) - v$$

$$\frac{\partial v}{\partial t} = D_v \frac{\partial^2 v}{\partial x^2} + 3(u - \bar{u}) - 2v$$



秋山謹製 パラメタサーチプログラム

main1.c

```
50
51  #define     N           (2)
52  double omega_0 = 1.0;
53  double gam = 0.1;
54
55  void
56  kansu_f(int dim,double *x,double *x_dot,double time)
57  {
58      x_dot[0] = - 2 * gam * x[0] - omega_0 * omega_0 * x[1];
59      x_dot[1] = x[0];
60  }
61
62  int main(int argc, char *argv[])
```

動かしたいパラメタを決める.
グローバル変数にしておくことをおすすめする

mainの引数を「void」から
「int argc, char *argv[]」へと変更する.

main2.c

GetParam2関数を呼ぶ.

```
printf("omega_0 = %f,gam = %f\n",omega_0,gam);
double dummy;
GetParam2(argc, argv, 100,
          &(omega_0),                    //p0
          &(gam),                        //p1
          &(dummy),
          NULL);
printf("omega_0 = %f,gam = %f\n",omega_0,gam);
```

Point! GetParam2関数の前ではパラメタはデフォルト値だが,
Point! GetParam2関数の後ではパラメタの値を変更できる.

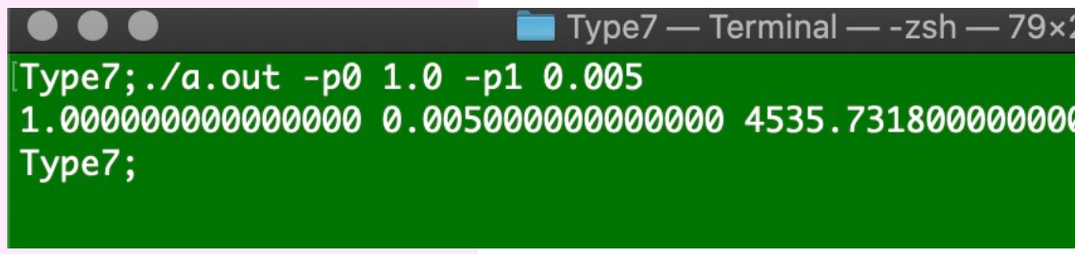
```
Type7;cc main2.c libbasic.a
Type7;./a.out
omega_0 = 1.000000,gam = 0.100000
omega_0 = 1.000000,gam = 0.100000
Type7;./a.out -p0 3.14 -p1 0.99
omega_0 = 1.000000,gam = 0.100000
omega_0 = 3.140000,gam = 0.990000
Type7;
```

main3.c

```
int stop_flag = 1;
double stop_time = t;
double eps = 1.0e-10;
for(i_time = 1;stop_flag ; i_time++)
{
    if(
       (fabs(x) < eps && fabs(p)  < eps)
       ||
       10000 < t)
    {
        stop_time = t;
        stop_flag = 0;
    }
    t = i_time * dt;
    Runge_n_dim(N,u,t,dt,work_u,kansu_f);
    p = u[0];
    x = u[1];
}
printf("%15.15f %15.15f %15.15f\n",omega_0,gam,stop_time);
return 0;
```

x座標が０に近くて，運動量も０に近い

時刻が10000を超えたら終わり



最後にパラメタと数値を出して終わり

main_Multi.c

```
#include <math.h>
#define     N                   (2)//パラメタの数
#define     TaiwaMode           (0)//これを1にすると対話的にパラメタを入力することができる.
//#define     NumberOfCPU         (2)//コメントアウトすると自動的にCPU数が設定される. 自動を推奨.
```

Nは問題によって変えること

```
        CallParameter(N,Param);
    }
    else
    {
        i = 0;//omega_0
        Param[i].Start = 0.0;
        Param[i].End = 1.0;
        Param[i].dx = 0.01;
        CreateData(&Param[i]);
        i = 1;//gamma
        Param[i].Start = 0.0001;
        Param[i].End = 0.01;
        Param[i].dx = 0.0001;
        CreateData(&Param[i]);
    }
    /* Get Now Time */
```

パラメタが増えたら適宜, このような表現を増やすこと

基本はこれだけ！

パラメタ並列探査プログラムの実行方法

１

main_Multi.c　を編集して，コンパイル＆実行　　ShellBoxができる.

２

cc main3.c libbasic.a　してa.outを作成

３

a.outをShellBoxにD&D

４

ShellBoxにcdして, ./myshell.shする

GLSC3D

GLSC3Dのインストールのための準備

MacOSでGLSC3Dを使用するために準備するもの

Xcode

MacOSでC言語などのプログラムがコンパイルできるようにするためには開発環境を整える必要があります.
とにかくXcodeをインストールして使えるようにしましょう.

あなたのMacは最新(macOS High Sierra, 10.13.1)である.

Yes⇒ OK

No⇒ 本日の講習会は最新型で行います. 極力Upgradeしましょう.

あなたのMacにはXcode Command Line Toolsの最新版が
インストールされている.

Yes⇒ OK

No⇒ 必須です. 絶対にインストールしてください.

あなたのMacにはXcodeの最新版がインストールされている.

Yes⇒ OK

No⇒ 本日の講習会は最新型で行います. 極力Upgradeしましょう.

自分のMacのOSの調べ方

自分のMacのOSの調べ方

Xcode Command Line Toolsがインストールされているか？

Xcode Command Line Toolsがインストールされているか？

Xcodeがインストールされているか調べる.

Xcodeがインストールされているか調べる.

Xcodeをインストールする.

Xcodeをインストールする.

(1) GLSC3Dを使用するにあたって

GLSC3Dについて

もともと数値計算結果を描画するグラフィックスライブラリとして、龍谷大学理工学部数理情報学科にて、小林亮氏、高橋大輔氏、中野浩氏、松木平淳太氏により

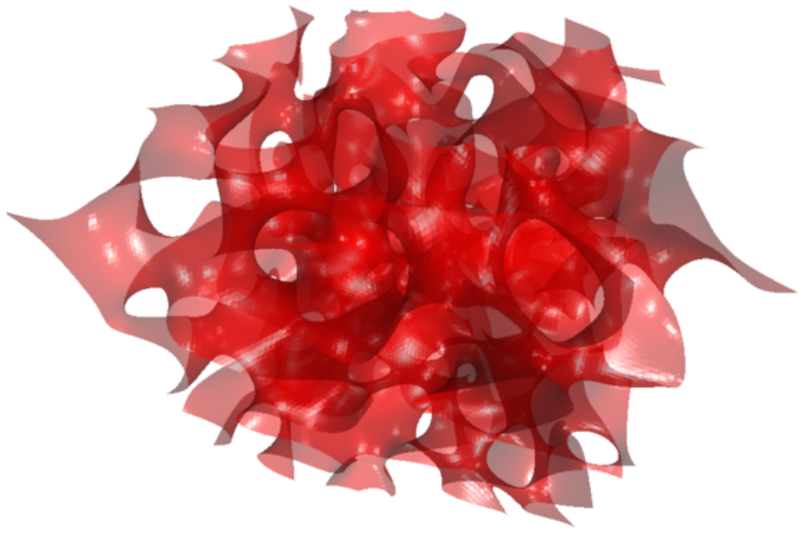
”GLSC (Graphics Library for Scientific Computing)”

が開発された.

GLSCは二次元平面上での描画を可能にするものである.

昨今の計算機性能の向上により，３次元の計算を行いつつ可視化することが求められるようになった.

⇒　三次元の計算結果を可視化させたい！



⇒　GLSC3Dの製作が始まった.

最新版は ”GLSC3DVer3.0.1” (2019/9/1) で、開発活動は現在も進行中である.

(1) GLSC3Dを使用するにあたって

メリットとデメリット

描画ツールを使用する場合:

メリット: Mathematica・AVXなどの描画はコマンドが非常に簡単でわかりやすい.

デメリット: 描画に時間がかかる場合がある. 内部を理解することは難しいだろう.
ソフトウェアは非常に高価(20万~数百万程度)なものである.

GLSC3Dを使用する場合:

メリット: GLSC3DはC言語などのプログラミングが必要であるが、描画はユーザーのプログラミング能力次第である. 関数を自作して、すぐに実装できる.

デメリット: C言語などの知識が必要なので、不慣れな方は序盤が大変かもしれない.
描画のすべてをユーザーが行うので、作業量は多くなるかもしれない.

GLSCとGLSC3Dを取り巻く環境

GLSC2D-3.5a+patch(小林亮，上山修正Ver)

描画：X Window System

言語：C,C++,Fortran

GLSC2D-3.8.6(小林亮，秋山修正Ver)

描画：X Window System

言語：C,C++,Fortran

2D描画までで限界（X Window Systemの制限）

GLSC3D-2.2.1(秋山)

描画：OpenGL 2.1

補助ライブラリ：freeGLUT

言語：C,C++

GLSC3D-3.0.0(秋山)

描画：OpenGL 3.3 + GLSL

補助ライブラリ：SDL

言語：C,C++，(Fortran)

最新規格（OpenGLの固定機能ではなく，シェーダーをつかって描画を行う．）

GLSC3D-3.0.0はGLSC2Dでできていたことをすべて包含することを目標にしている．

用語解説

OpenGL：1.1，1.5(2003~)，2.x(2004~)，3.x(2008~)，4.x(2010~)
がリリースされている.

GLSL：OpenGL Shading Language．OpenGL 2.0以降で使用可能になった.
グラフィックを効率的に行うための言語.

X Window System：GLSCまでは標準に使用されていた．3Dの描画機能はできない．Macではデフォルトでインストールされない．おそらく将来的に動作しなくなる可能性大.

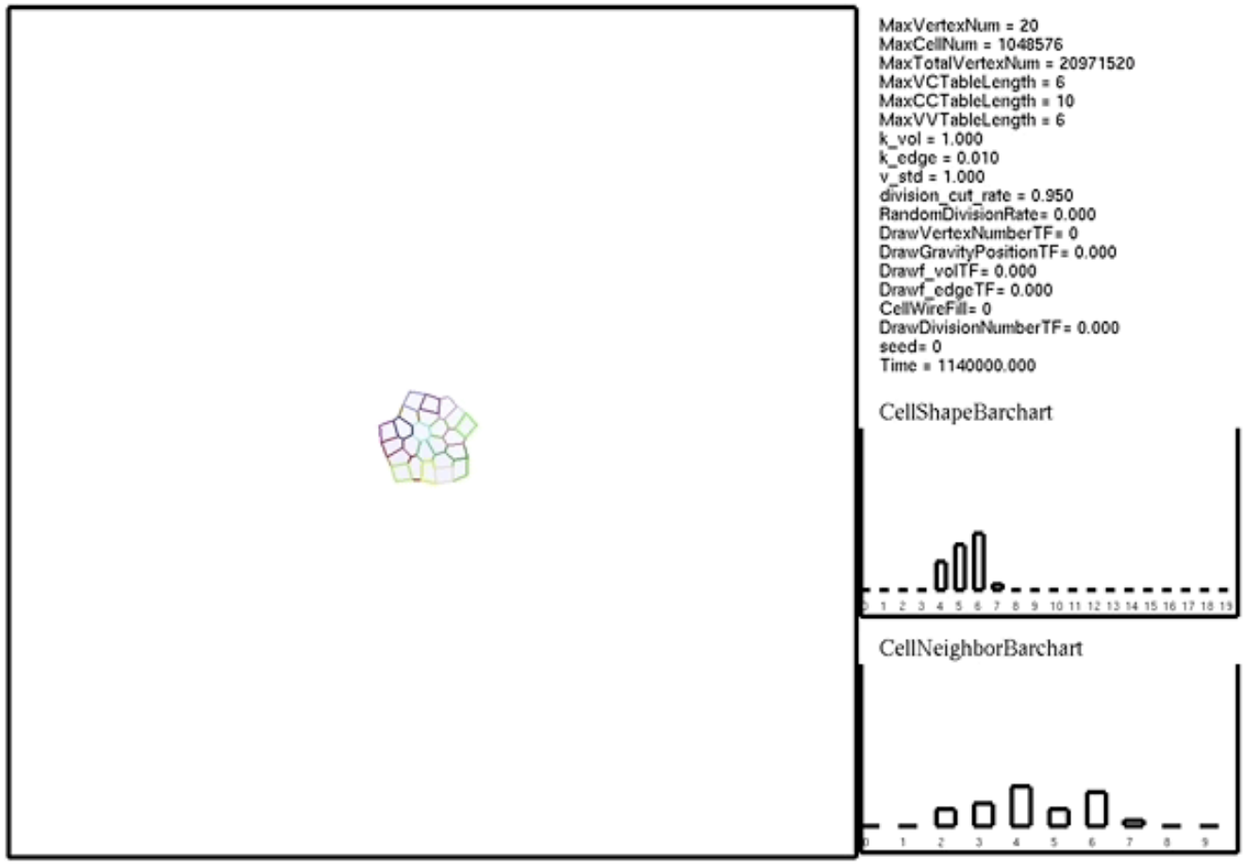
GLUT(OpenGL Utility Toolkit) ：1998 年に発表された．Version 3.7 以降整備が行われなった.

free GLUT：GLUTの後継だが，やはり最近は2015年から止まっている．．．

SDL：GLUTを置き換える新しいツール．Mac,Windows,Linux,iOSなどをカバーしている.

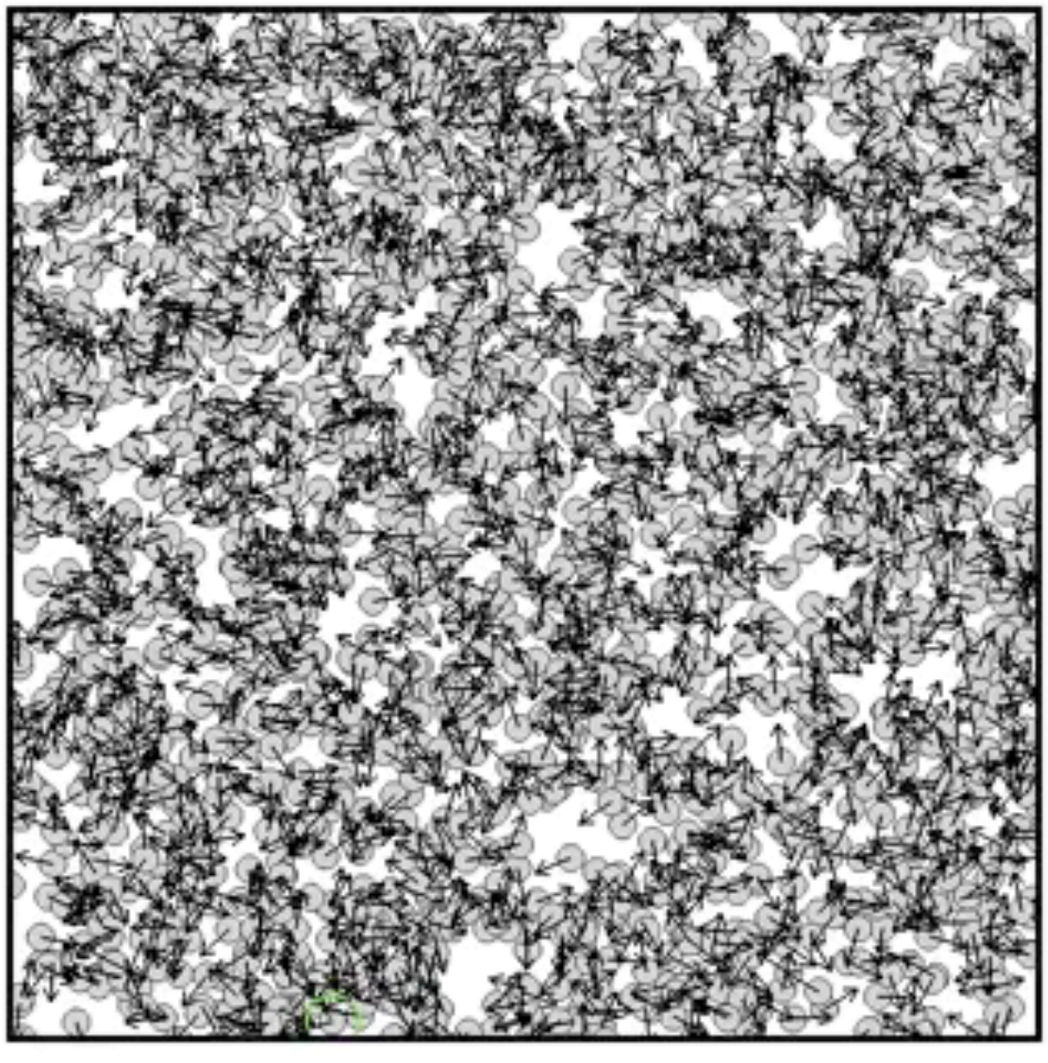
今日の講習会を受けるとこんなことができるようになる.

2D デモンストレーション

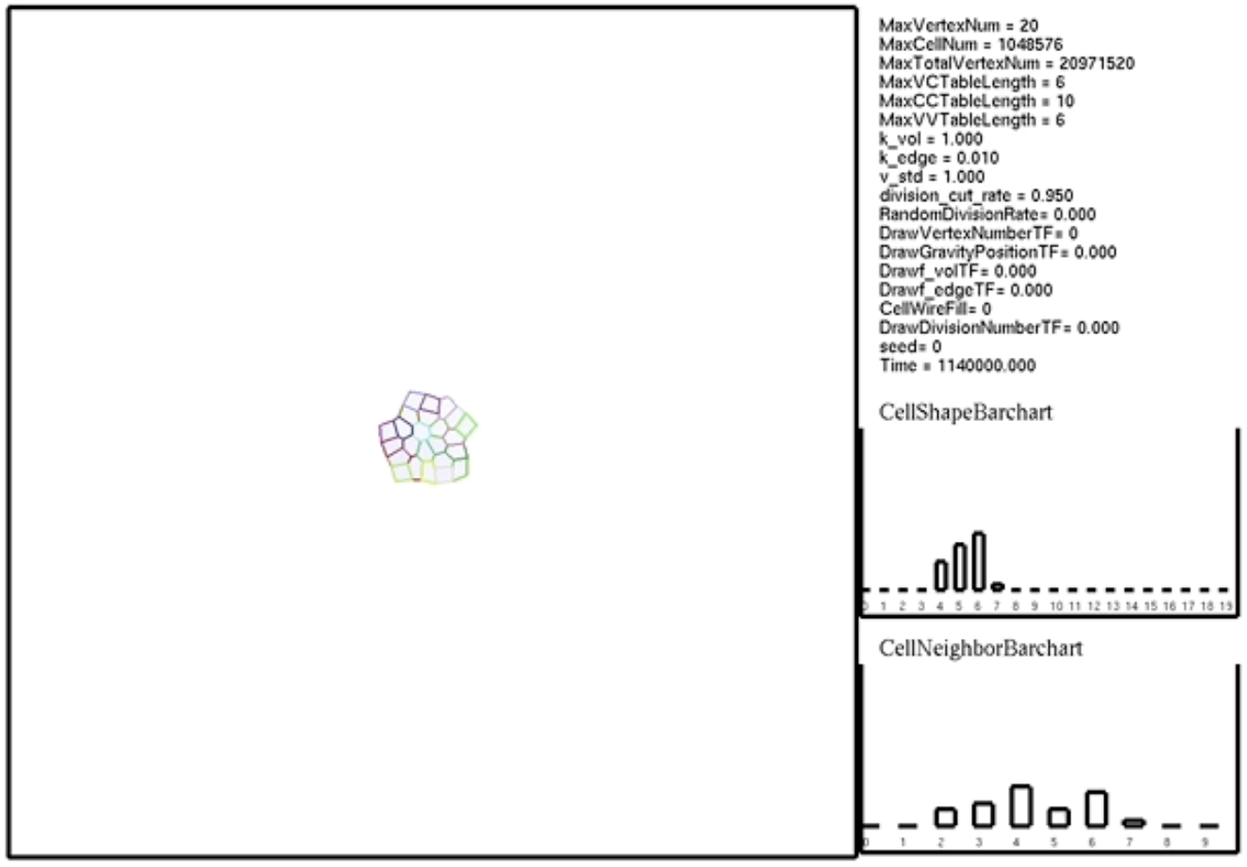


細胞分裂の様子を
シミュレーションする.

鳥が群れて飛ぶ
シミュレーション

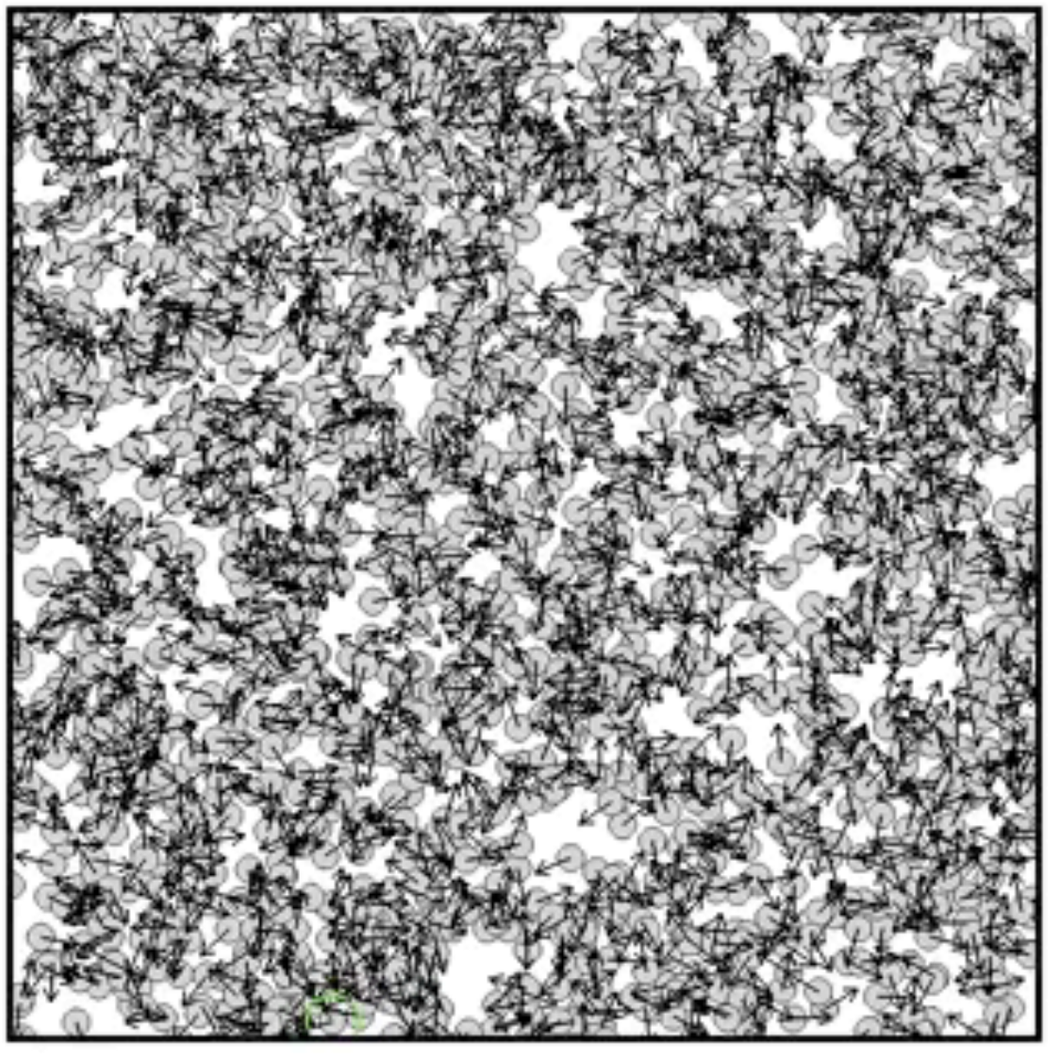


2D デモンストレーション

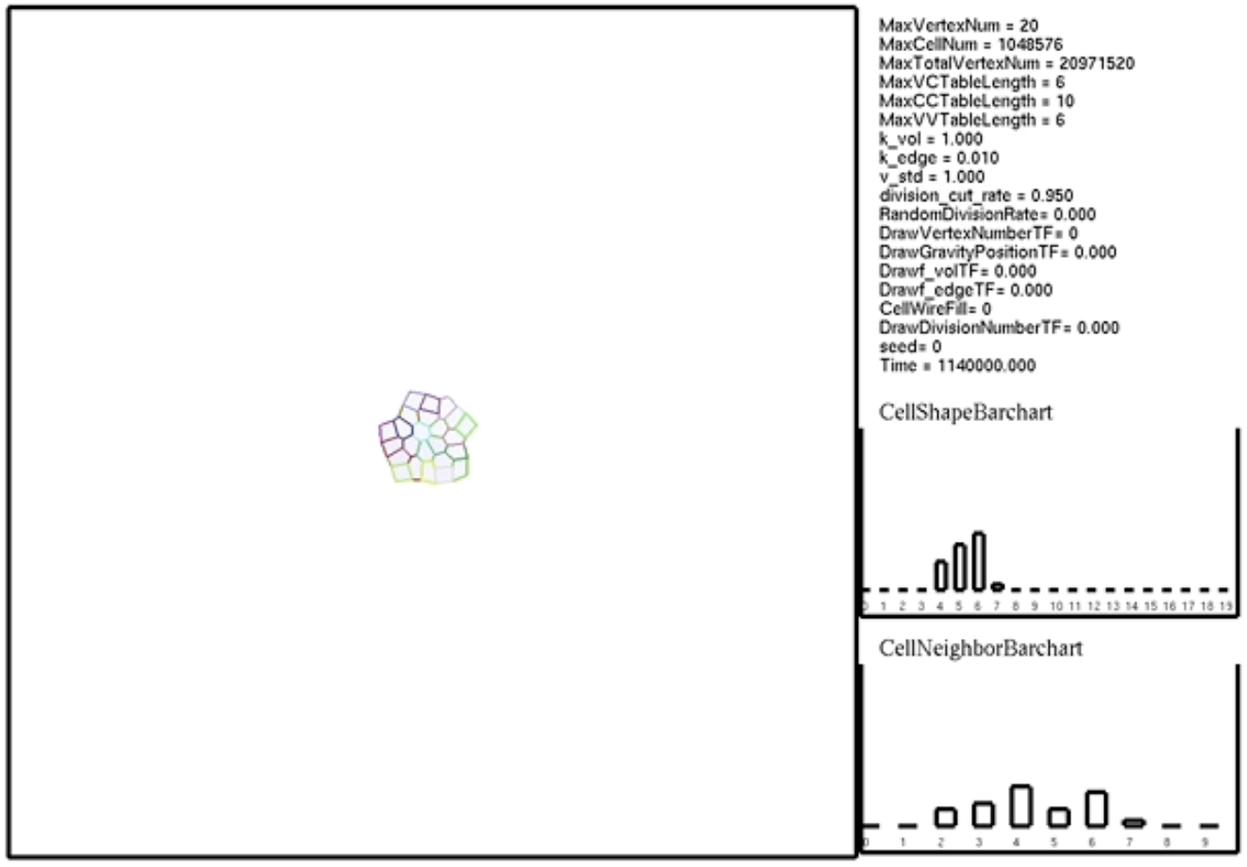


細胞分裂の様子を
シミュレーションする.

鳥が群れて飛ぶ
シミュレーション

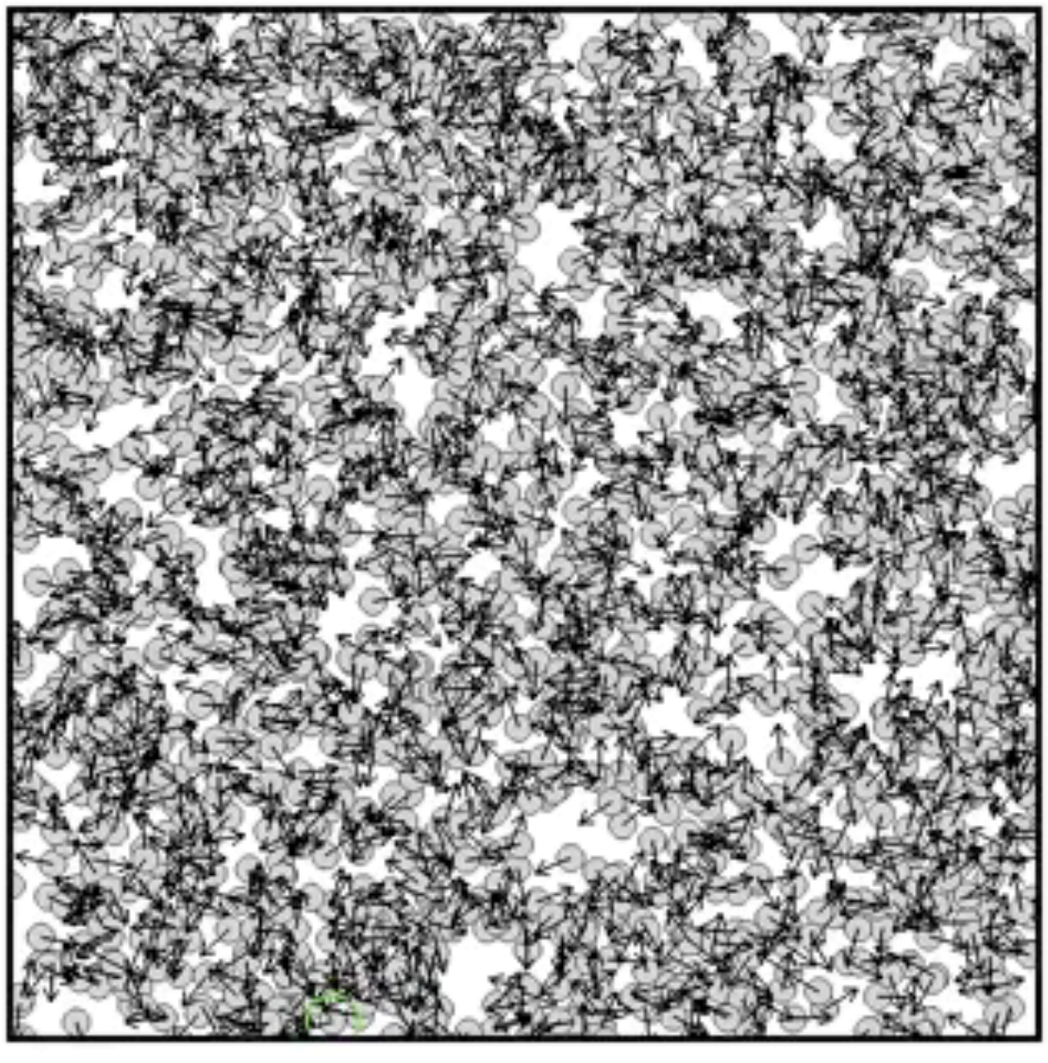


2D デモンストレーション

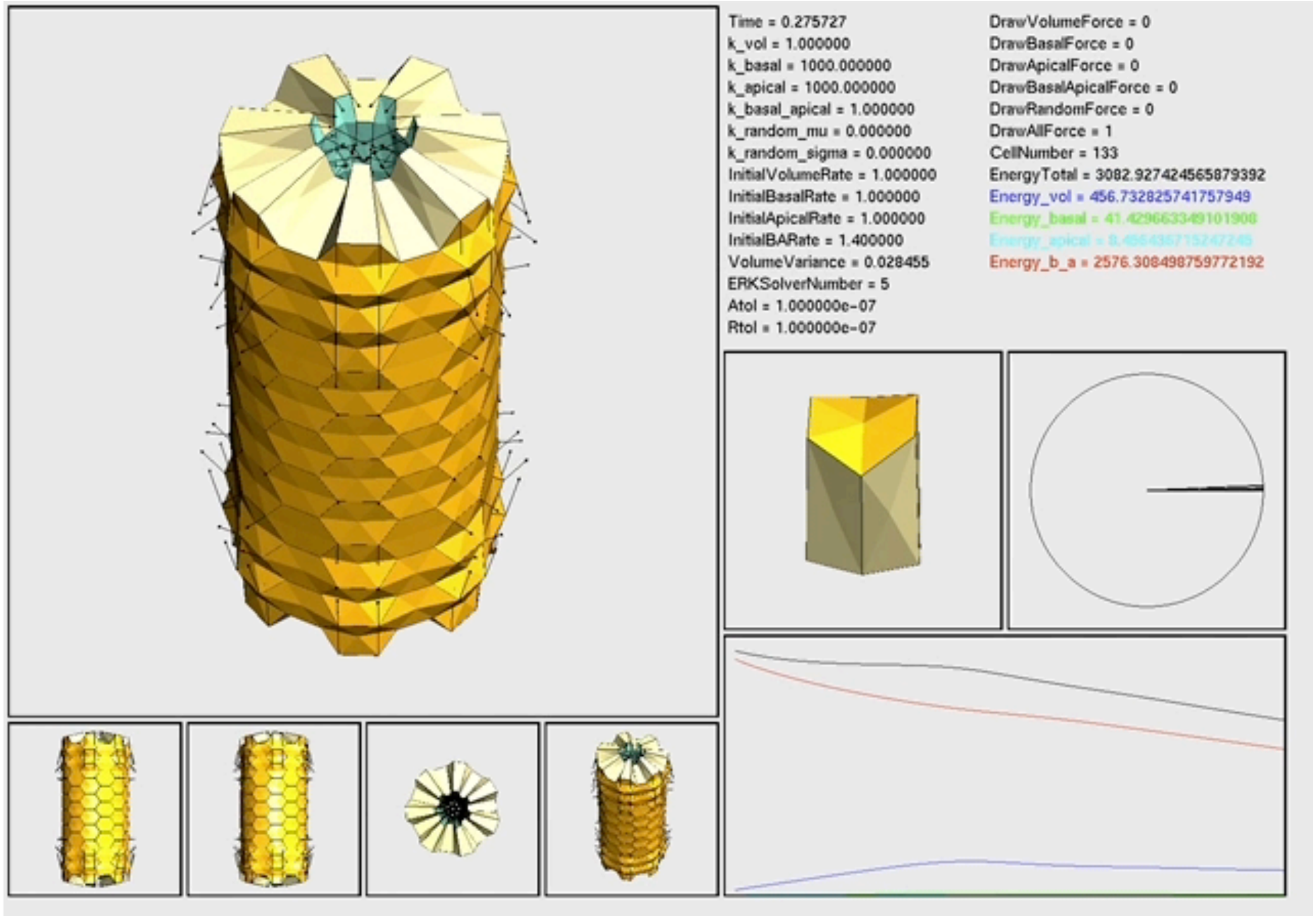


細胞分裂の様子を
シミュレーションする.

鳥が群れて飛ぶ
シミュレーション

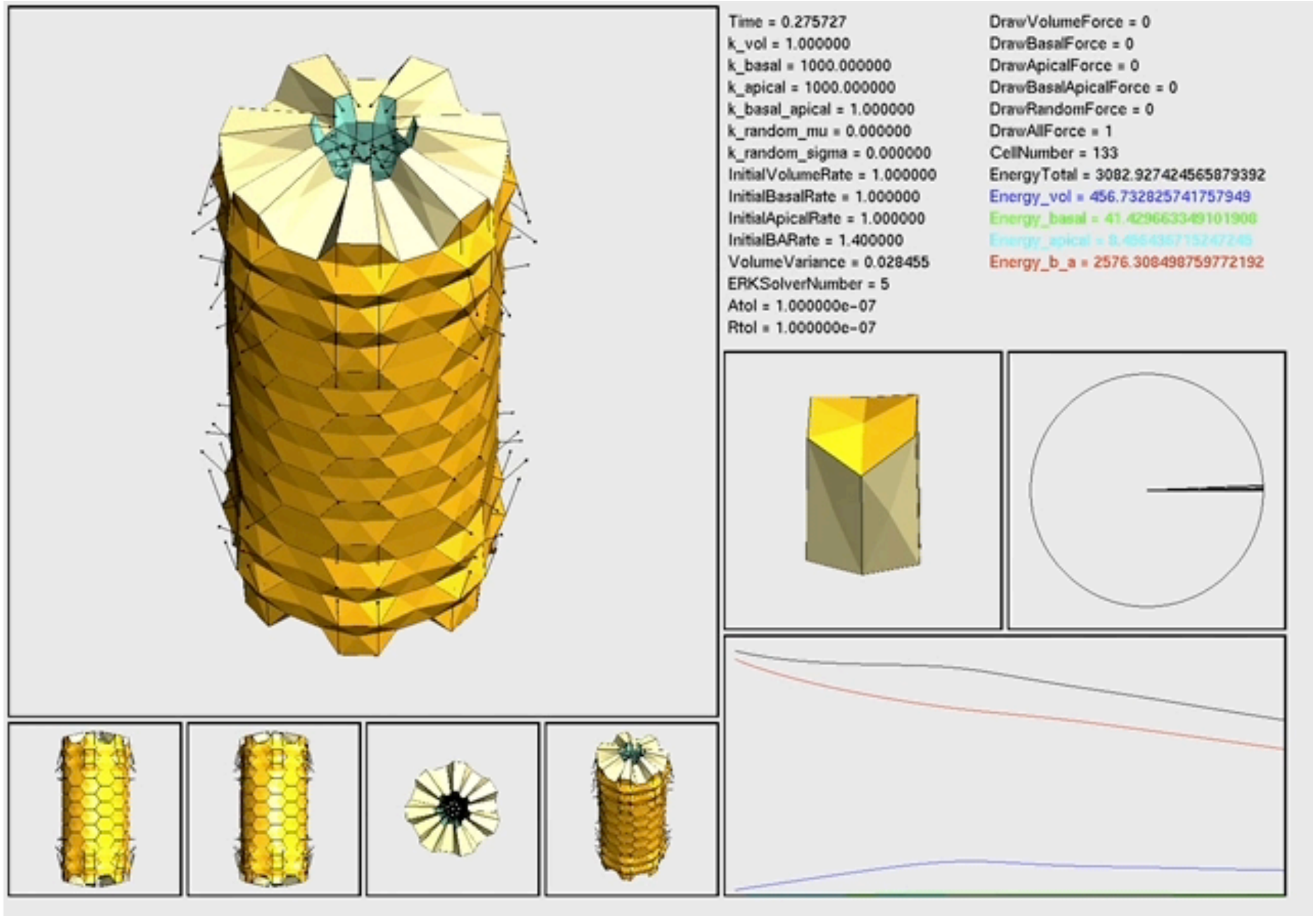


3D デモンストレーション



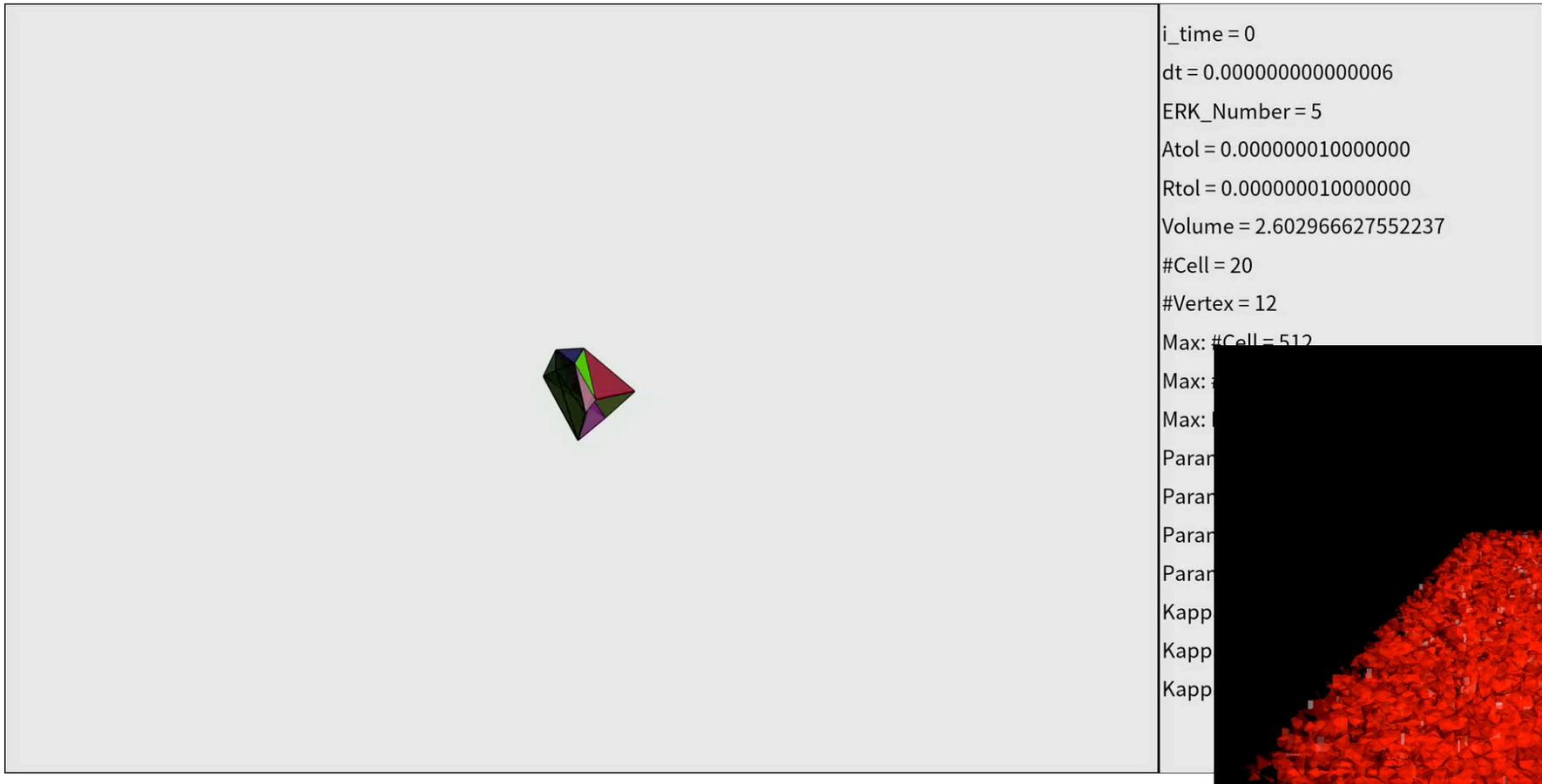
組織にどのような力が加われば，個々の細胞にどのような
変化が起きるかシミュレーションする

3D デモンストレーション

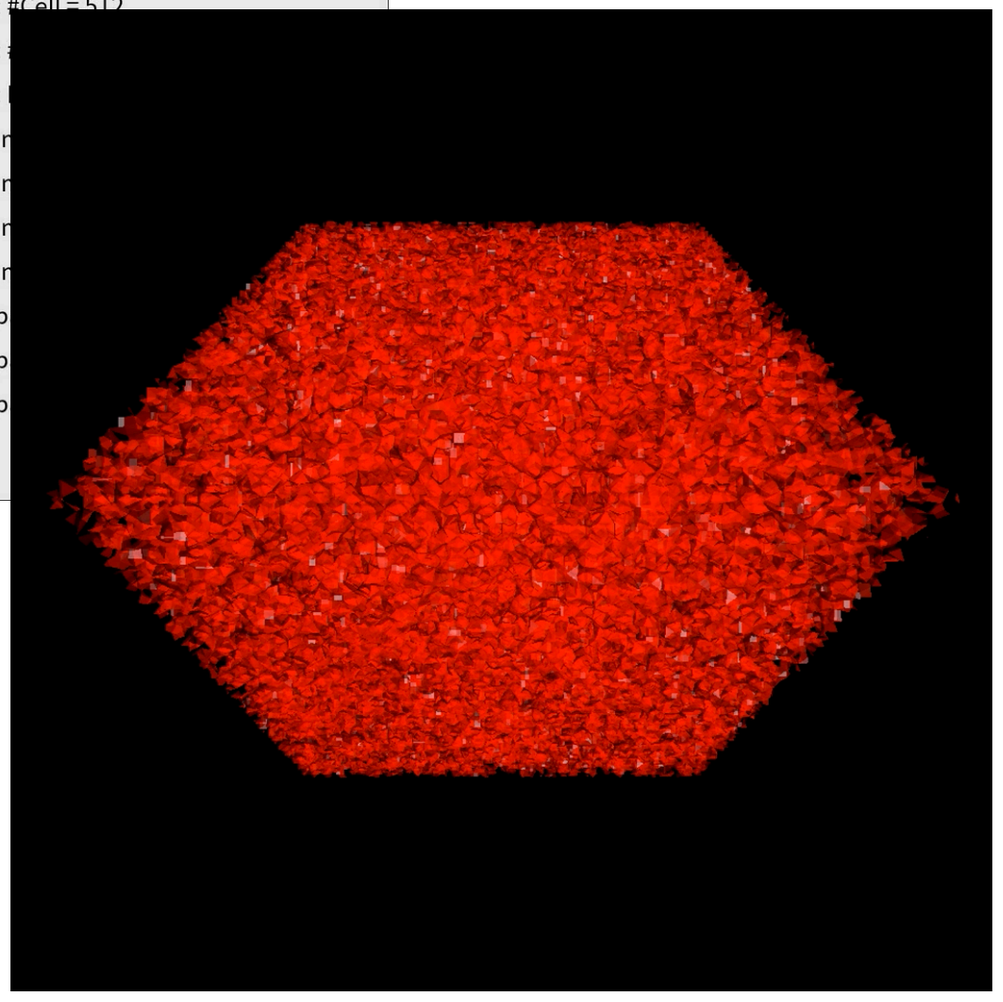


組織にどのような力が加われば，個々の細胞にどのような
変化が起きるかシミュレーションする

3D デモンストレーション

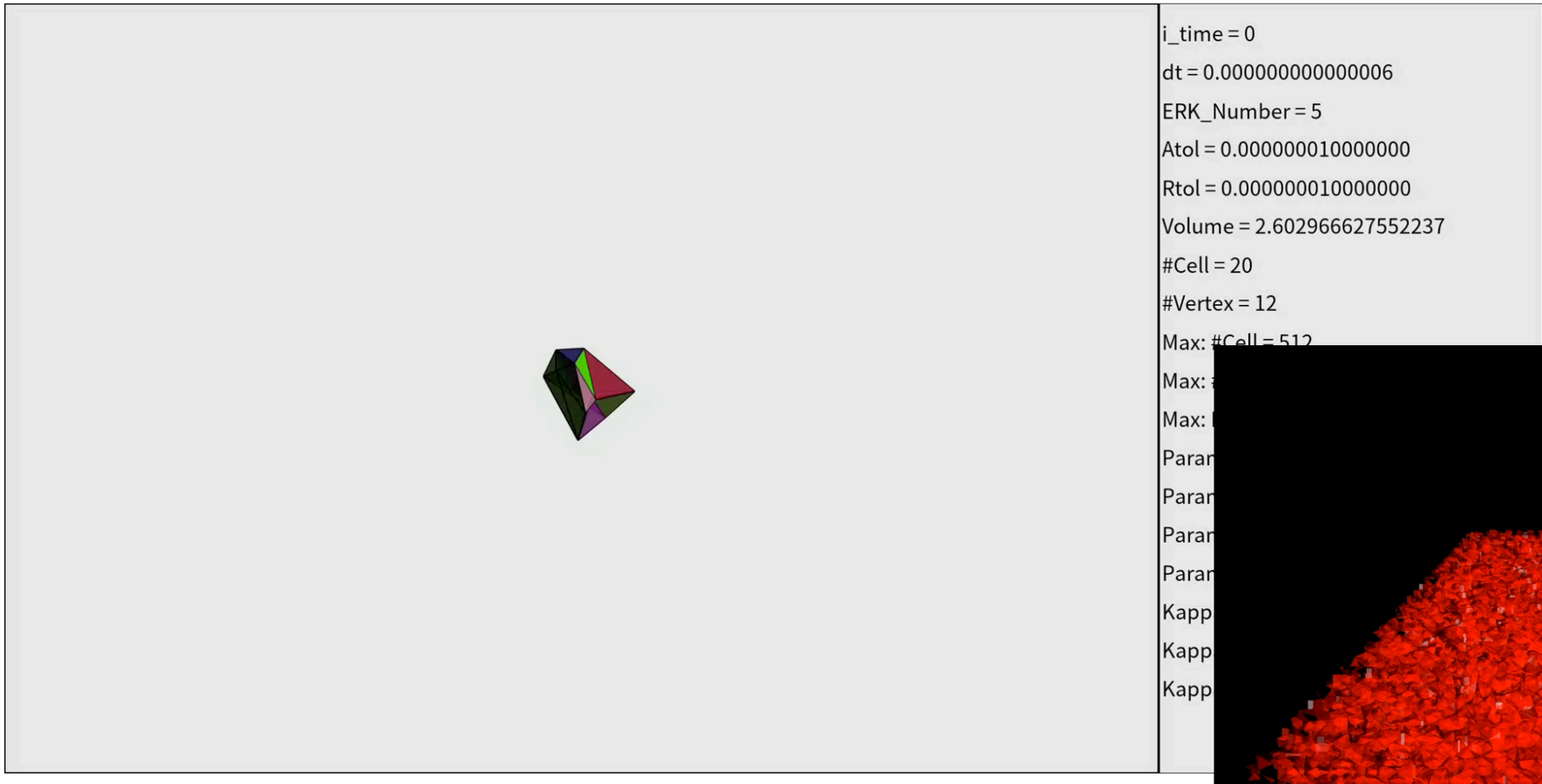


細胞分裂？の過程を3D
シミュレーションする

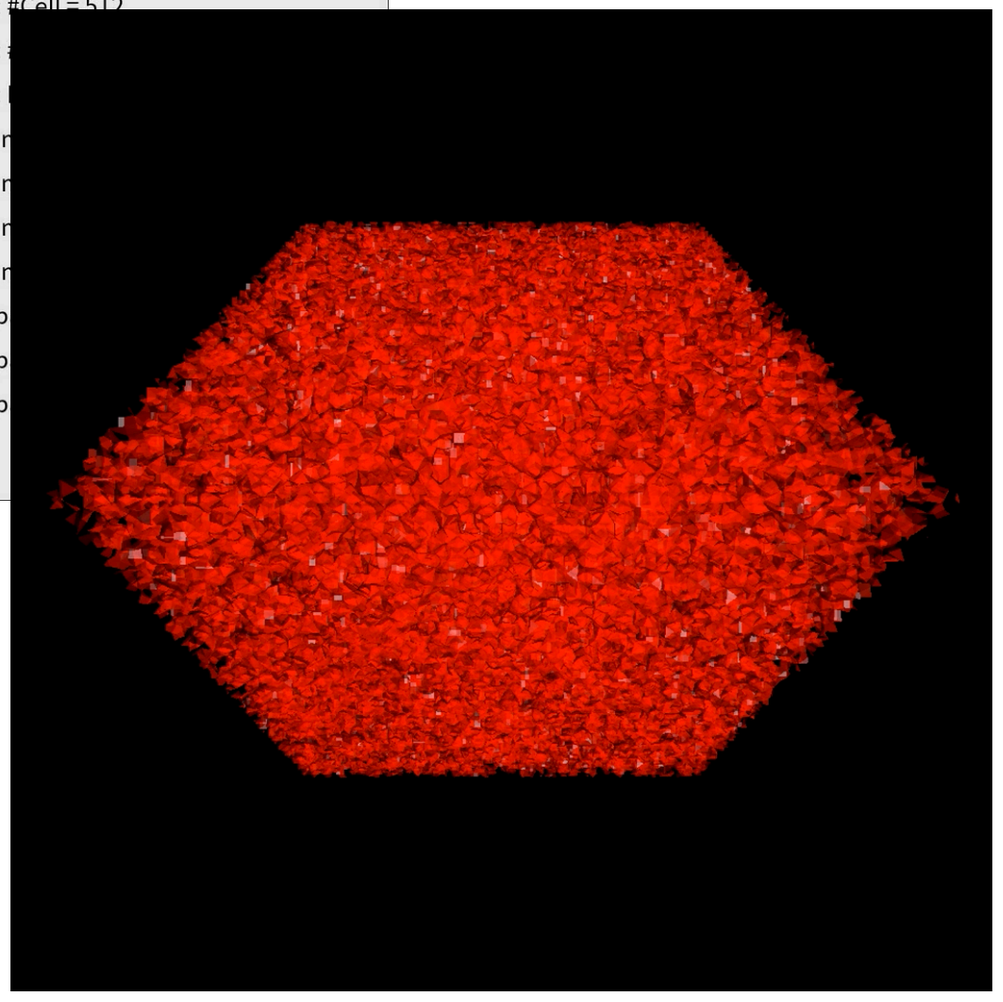


3D Turingモデル

3D デモンストレーション

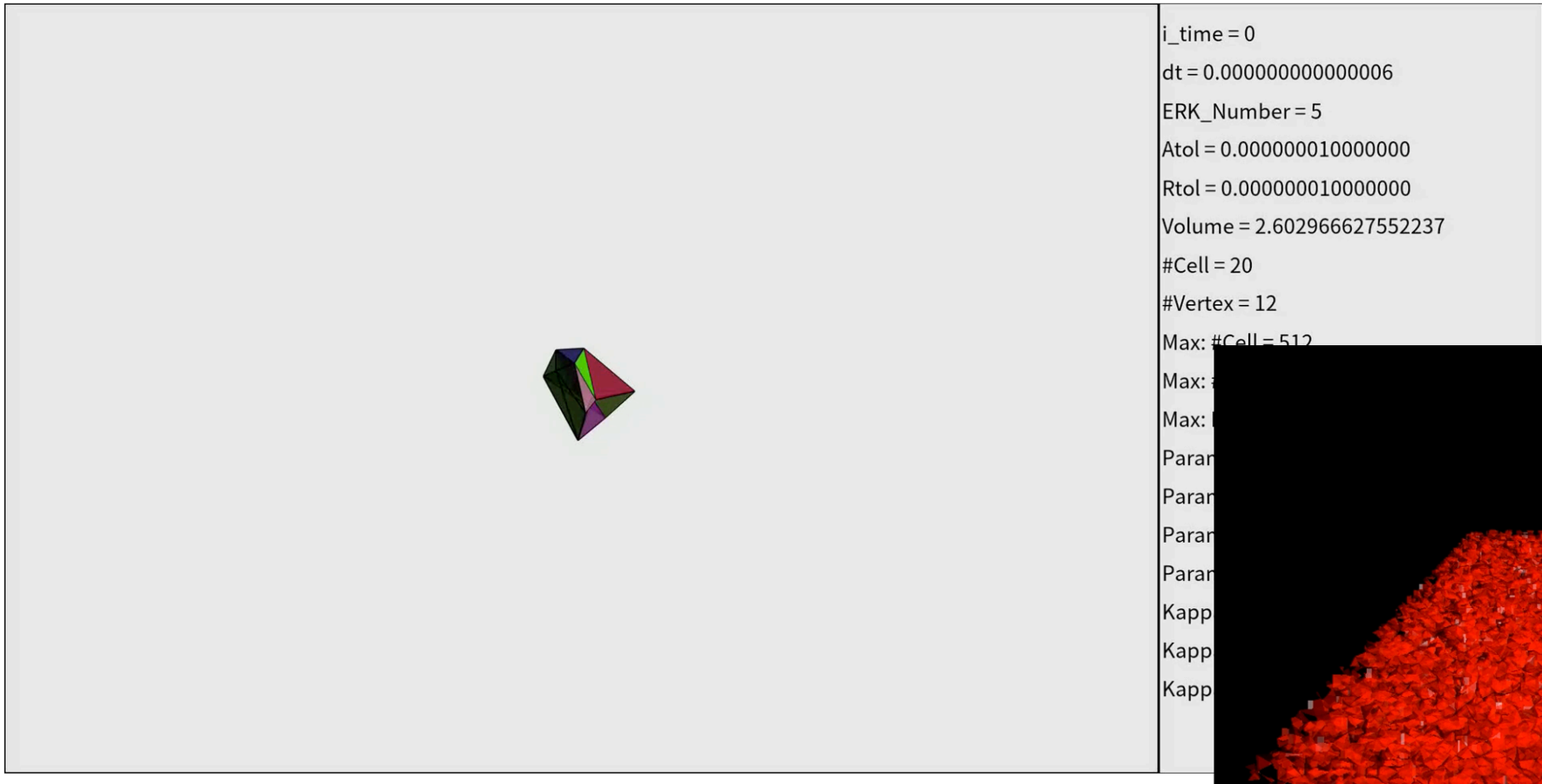


細胞分裂？の過程を3D
シミュレーションする

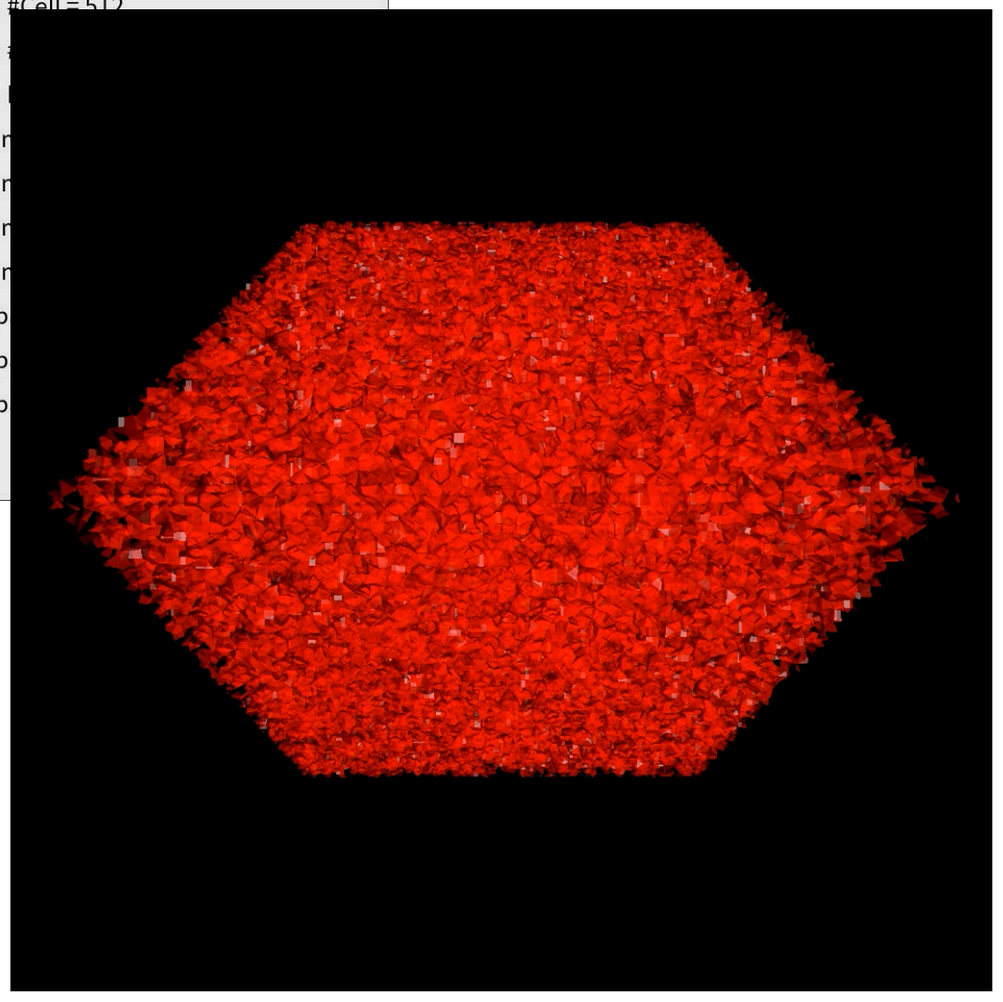


3D Turingモデル

3D デモンストレーション



細胞分裂？の過程を3D
シミュレーションする



3D Turingモデル

GLSC3Dのインストール

GLSC3Dのインストールは大きく分けて，２つある.

○前もって作製されたMac版のGLSC3Dのライブラリを適切な場所に設定する.

⇒ GLSC3D_mac_minimumを使用する.

手軽にGLSC3Dをインストールできるが，Mac版しかない．最新版のMacでコンパイルされている．実行ファイルのサイズが大きい.

○GLSC3Dのライブラリをコンパイルし，ライブラリを作製後，適切な場所に設定する.

⇒ Script_on_macを使用する.

どんな環境でも最新版のGLSC3Dをインストールできる．カスタムで作製できるので，必ず動作する.
一方で，コンパイルの設定を整えるのに時間がかかる.

1_GLSC3D講習会_Mac編へGo

GLSC3D_mac_minimumを使用したインストール

「秋山正和」と検索するか

http://www.isc.meiji.ac.jp/~akiyama_masakazu/

に行き，　GLSC3D_mac_minimum.zipをDLする.

⇒ GLSC3D_mac_minimumを使用する.
GLSC3D_mac_minimum.zipを展開し，Desktopに置く.

⇒ GLSC3D_mac_minimumを使用する.
GLSC3D_mac_minimum.zipを展開し，Desktopに置く.

⇒ GLSC3D_mac_minimumを使用する.

と表示されたら，OK!

次スライドの設定を行う.

画面のキャプチャーも
できているか確認

もし，表示されていない場合は，Xcodeなどの設定を疑う.
もしくはScript_on_macを使用する.

⇒ GLSC3D_mac_minimumを使用する.

GLSC3Dをインストールする.

⇒ GLSC3D_mac_minimumを使用する.

GLSC3Dをインストールする.

⇒ GLSC3D_mac_minimumを使用する.

GLSC3Dをアンインストールする.

⇒ GLSC3D_mac_minimumを使用する.

GLSC3Dをアンインストールする.

⇒ Script_on_macを使用する.

http://www-mmc.es.hokudai.ac.jp/~masakazu

に行き，Script_on_mac.zipをDLする.

公開ソフトウェア

G.L.S.C.3D

C,C++から呼び出して使うことができる，3次元グラフィックライブラリ

Manual(Ver3.0.0)

Ver3.0.0からは本体のダウンロードからインストールまでのすべてを自動的に行ってくれるインストーラを提供しました.

Ver3.0.0は一部の関数の引数がVer2.2.1と異なります．インストールを行う際はこの点にご注意ください.

Macユーザの方で最小限でインストールをする場合は

こちら

Macユーザの方でフルインストールをする場合は

こちら

←これ

をダウンロードしてください.

Ubuntuユーザの方は

こちら

をダウンロードしてください.

Centosユーザの方は将来的に使用できるまで，Ver2.2.1を使用してください.

Ver2.2.1はこちらです(マニュアル公開2014/12/8)

Manual(Ver2.2.1)　Download(zip)

MD5 (GLSC3DVer2.2.1.zip) = 7956875e257fec861c0c1ead2e82f899
SHA1(GLSC3DVer2.2.1.zip)= 46dd6d41e8d101b1eb724de6383143143d04ec18

⇒ Script_on_macを使用する.

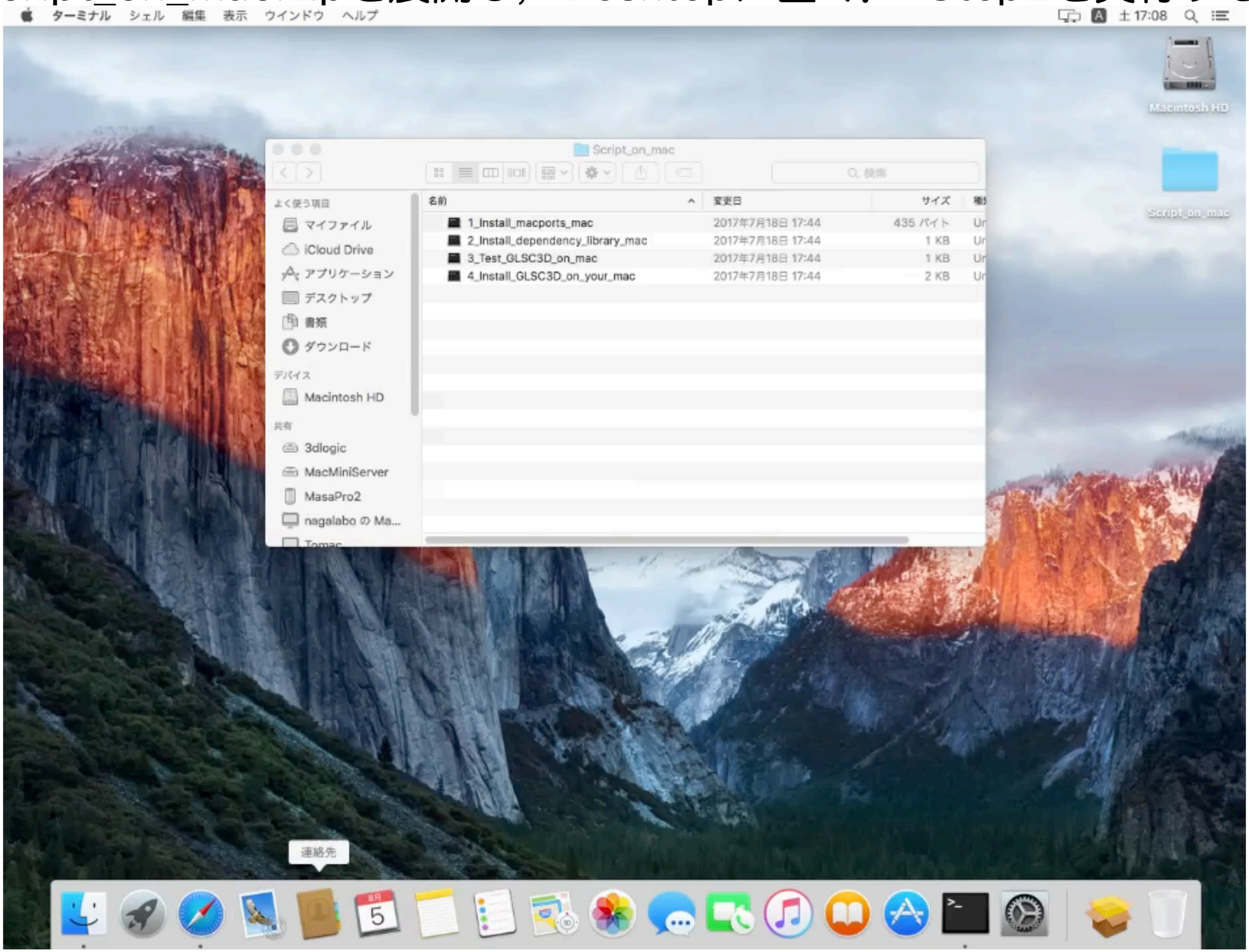
Script_on_mac.zipを展開し，Desktopに置く. ⇒Step1を実行する.

⇒ Script_on_macを使用する.

Script_on_mac.zipを展開し，Desktopに置く. ⇒Step1を実行する.

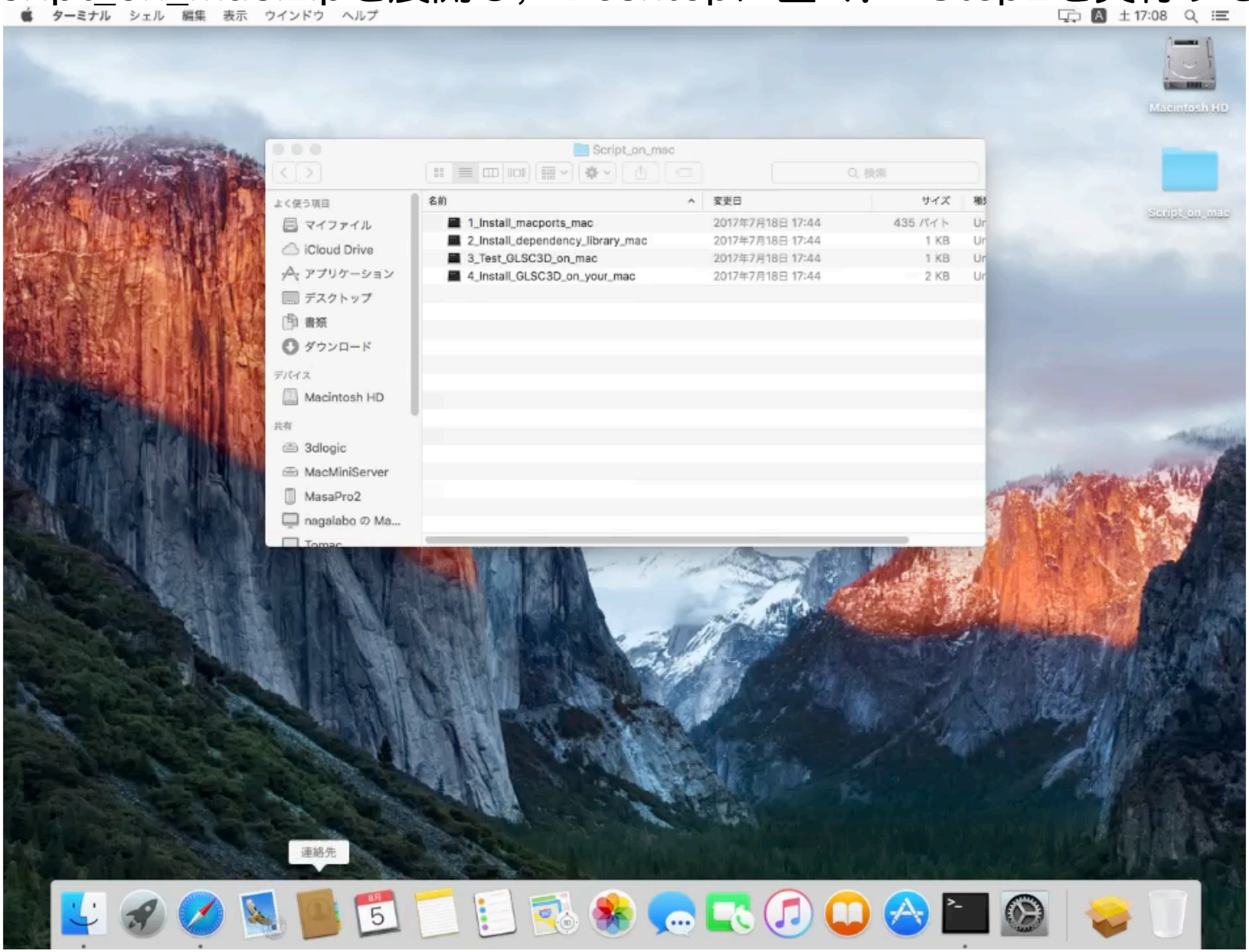
⇒ Script_on_macを使用する.

Script_on_mac.zipを展開し，Desktopに置く．⇒Step2を実行する.



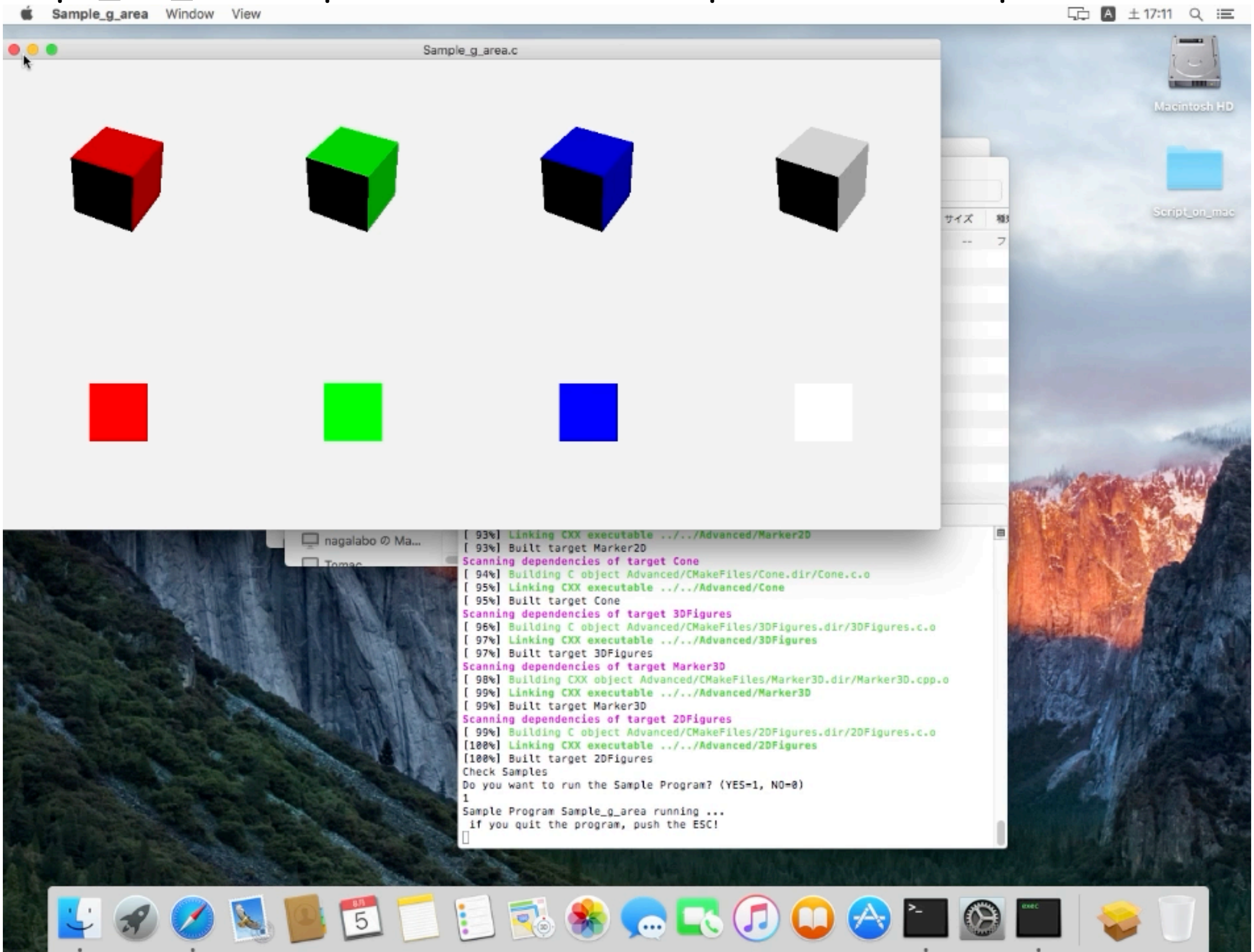
⇒ Script_on_macを使用する.

Script_on_mac.zipを展開し，Desktopに置く．⇒Step2を実行する.



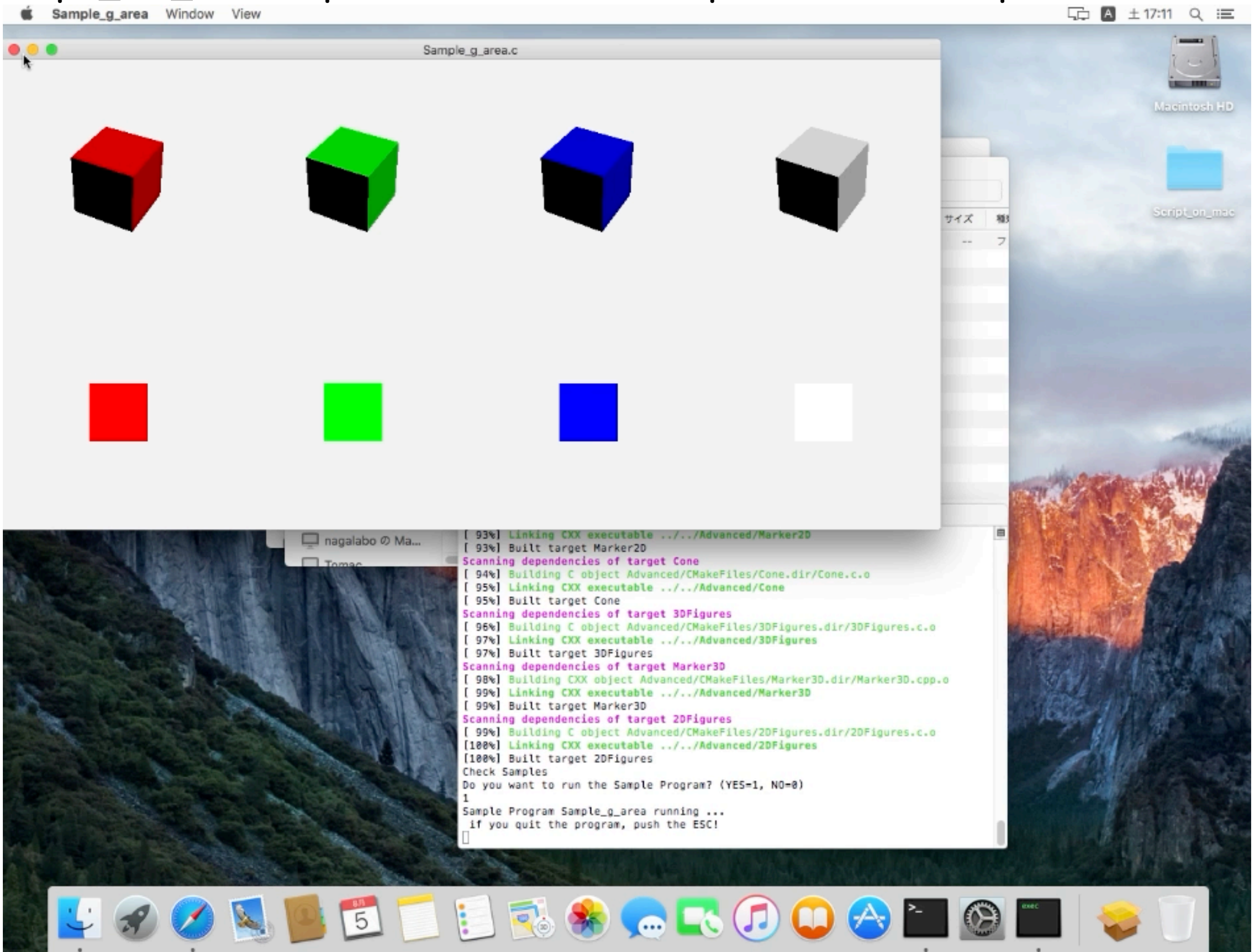
⇒ Script_on_macを使用する.

Script_on_mac.zipを展開し，Desktopに置く. ⇒Step3を実行する.



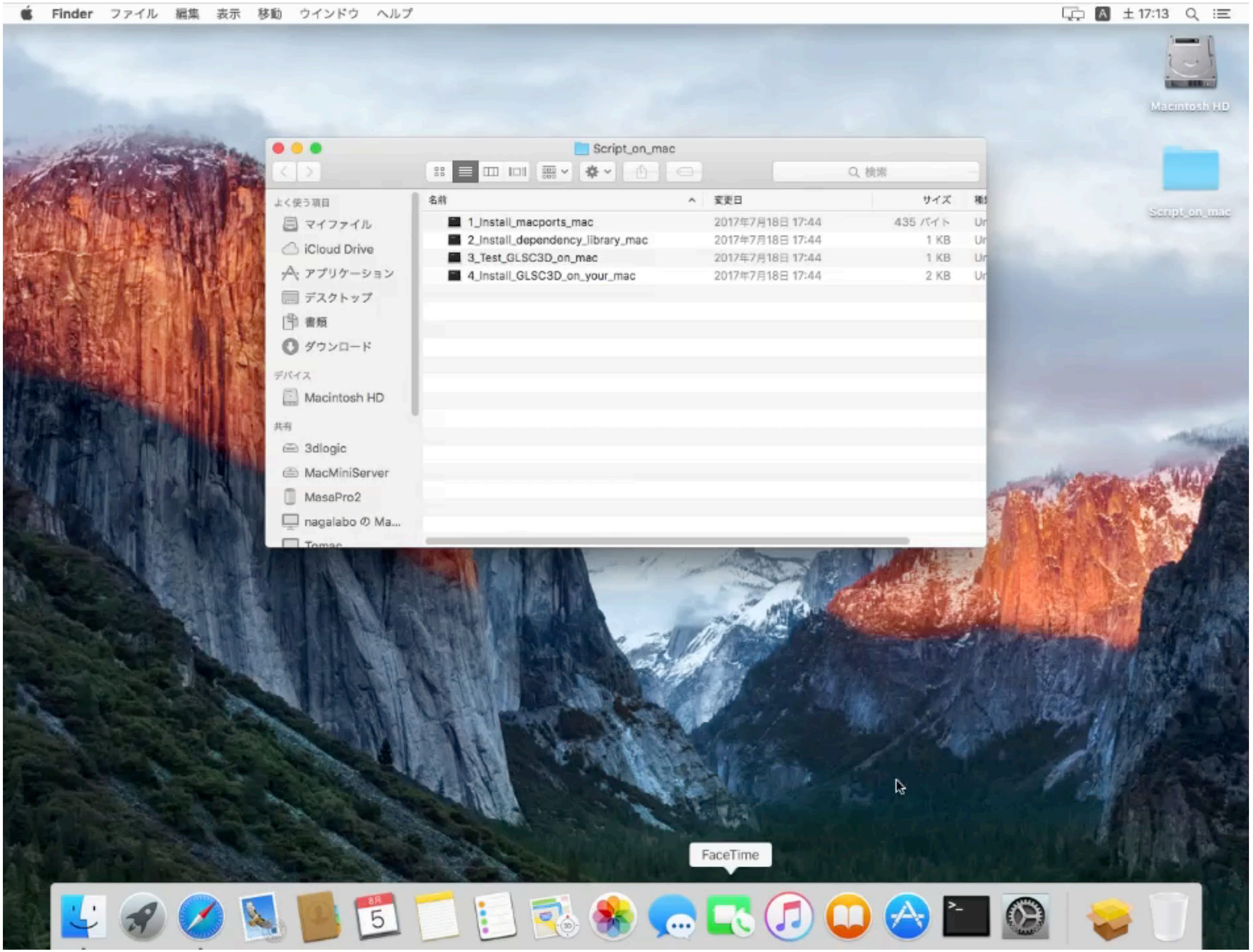
⇒ Script_on_macを使用する.

Script_on_mac.zipを展開し，Desktopに置く. ⇒Step3を実行する.



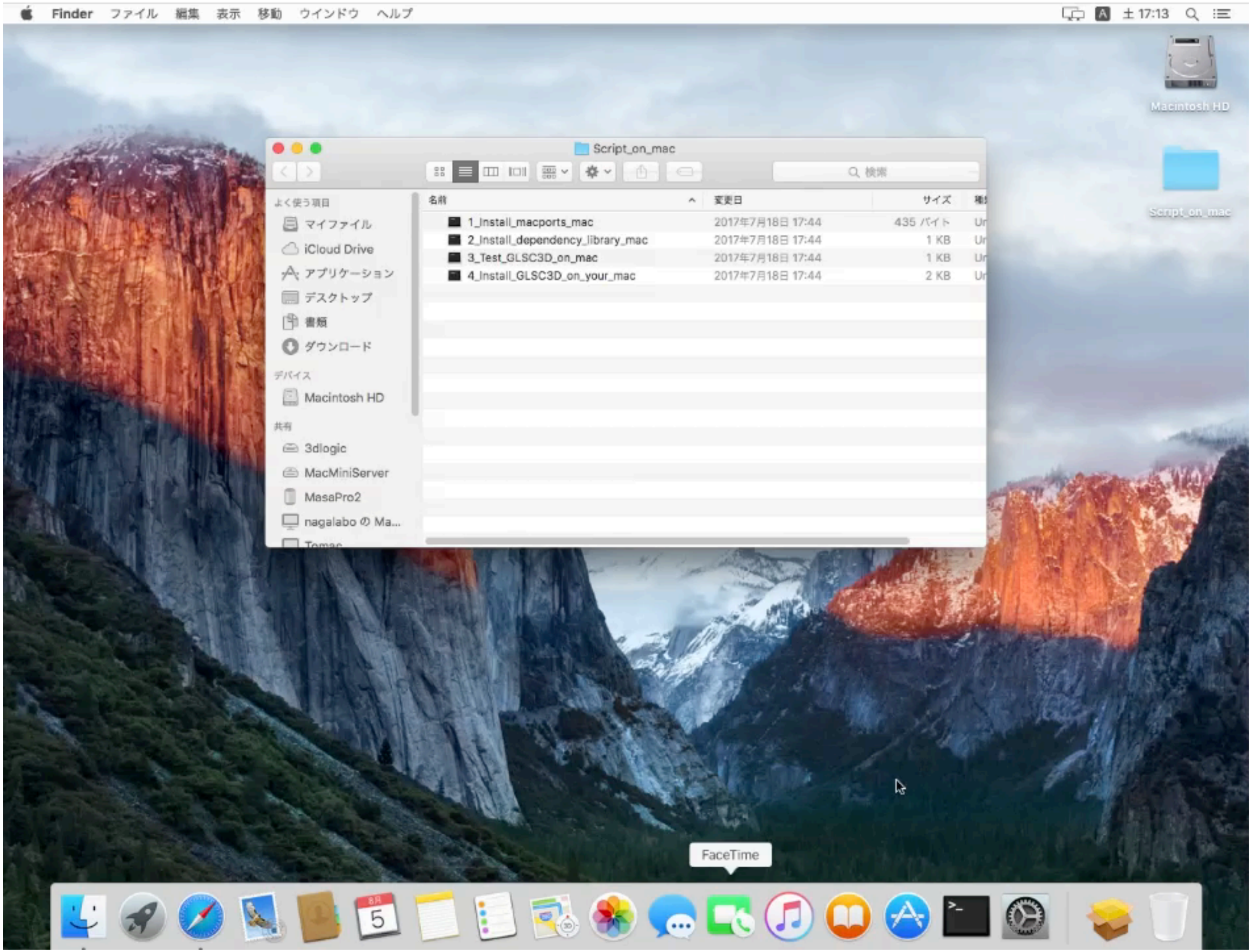
⇒ Script_on_macを使用する.

Script_on_mac.zipを展開し，Desktopに置く. ⇒Step4を実行する.



⇒ Script_on_macを使用する.

Script_on_mac.zipを展開し，Desktopに置く. ⇒Step4を実行する.



⇒ Script_on_macを使用する.

Step4までOKであれば，インストールは完了．次のようにしてインストールが上手くいったかを確認

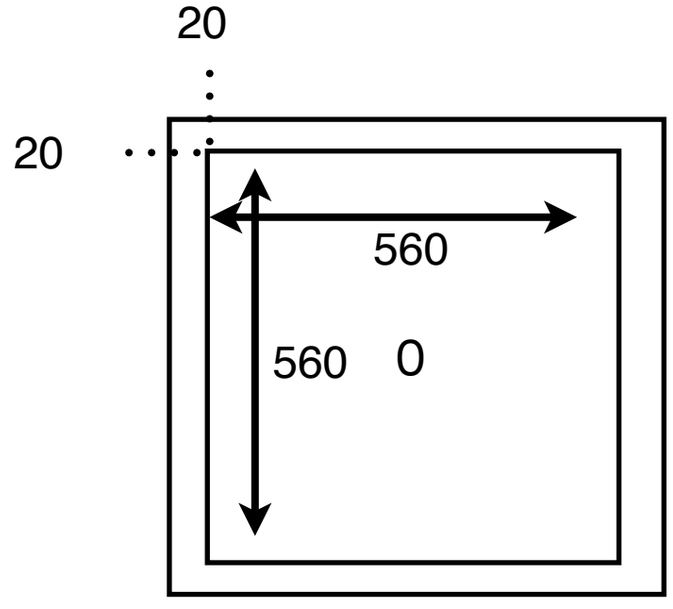
⇒ Script_on_macを使用する.

Step4までOKであれば，インストールは完了．次のようにしてインストールが上手くいったかを確認

GLSC3Dの2Dの世界へ

2DのGLSC3D

~1_CreateWindow.c~

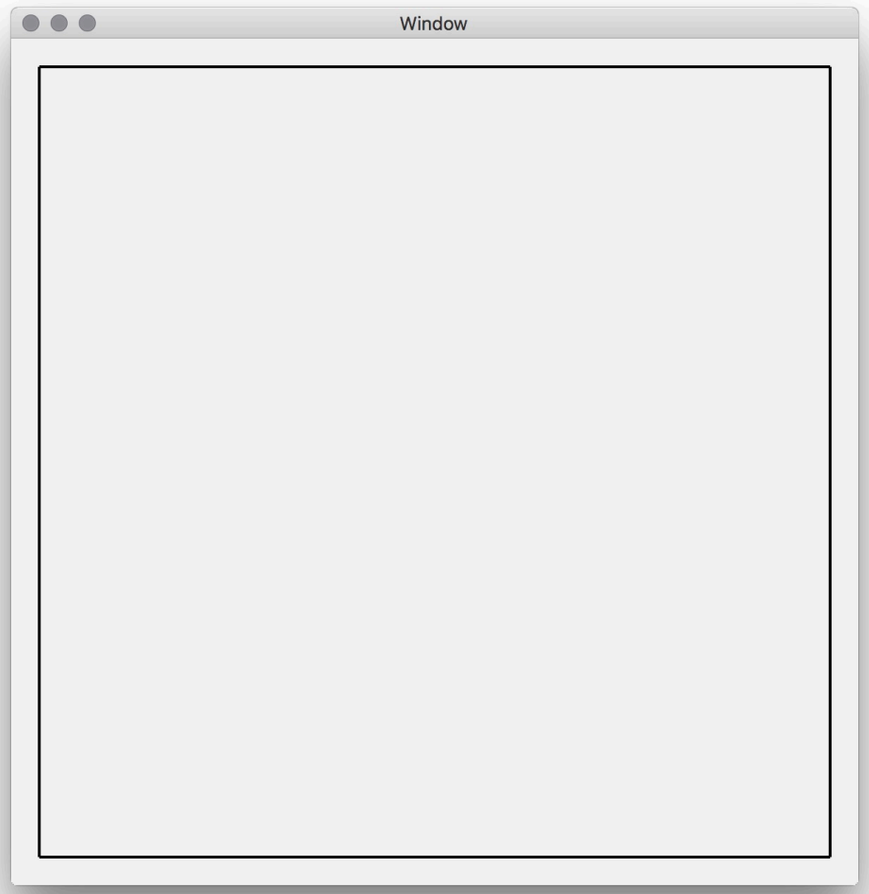



```c
#include<stdio.h>
#include<glsc3d_3.h>

int main()
{
    g_init("Window", 600, 600);  //Pixel Size
    g_def_scale_2D(0,            //ID
                   -1, 1,        //xmin,xmax
                   -1, 1,        //ymin,ymax
                   20.0, 20.0,   //Window (Left, Top) Position
                   560, 560);    //Window Size (x,y)
    g_cls();                     //Clear window
    g_sel_scale(0);              //Select Virtual scale
    g_boundary();                //Draw Boundary
    g_finish();                  //flush Draw buffer
    g_sleep(10.0);               //Sleep 10 sec

    return 0;
}
```

1_CreateWindow.c

2DのGLSC3D

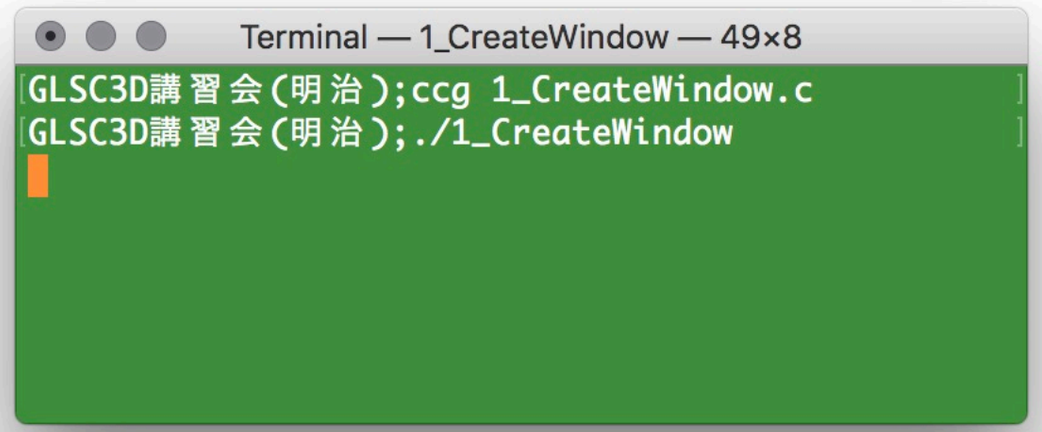
~1_CreateWindow.c~



実行結果

$ ccg 1_CreateWindow.cとうつ.
1_CreateWindowという名前の
実行ファイルができる.

$ ./1_CreateWindowとうつ.



1_CreateWindow.c

2DのGLSC3D

~1_CreateWindow.c~

解説

```c
#include<stdio.h>
#include<glsc3d_3.h>

int main()
{
    g_init("Window", 600, 600);  //Pixel Size
    g_def_scale_2D(0,            //ID
                   -1, 1,        //xmin,xmax
                   -1, 1,        //ymin,ymax
                   20.0, 20.0,   //Window (Left, Top) Position
                   560, 560);    //Window Size (x,y)
    g_cls();                     //Clear window
    g_sel_scale(0);              //Select Virtual scale
    g_boundary();                //Draw Boundary
    g_finish();                  //flush Draw buffer
    g_sleep(10.0);               //Sleep 10 sec

    return 0;
}
```

←GLSC2Dと同じ

←ーーーーーー画面を消す

←ーーー0番のスケールを選ぶ

←ーーーー0番のスケールの箱を描く

←ーーー描画命令を処理する

←ーーー10秒休む.

1_CreateWindow.c

2DのGLSC3D

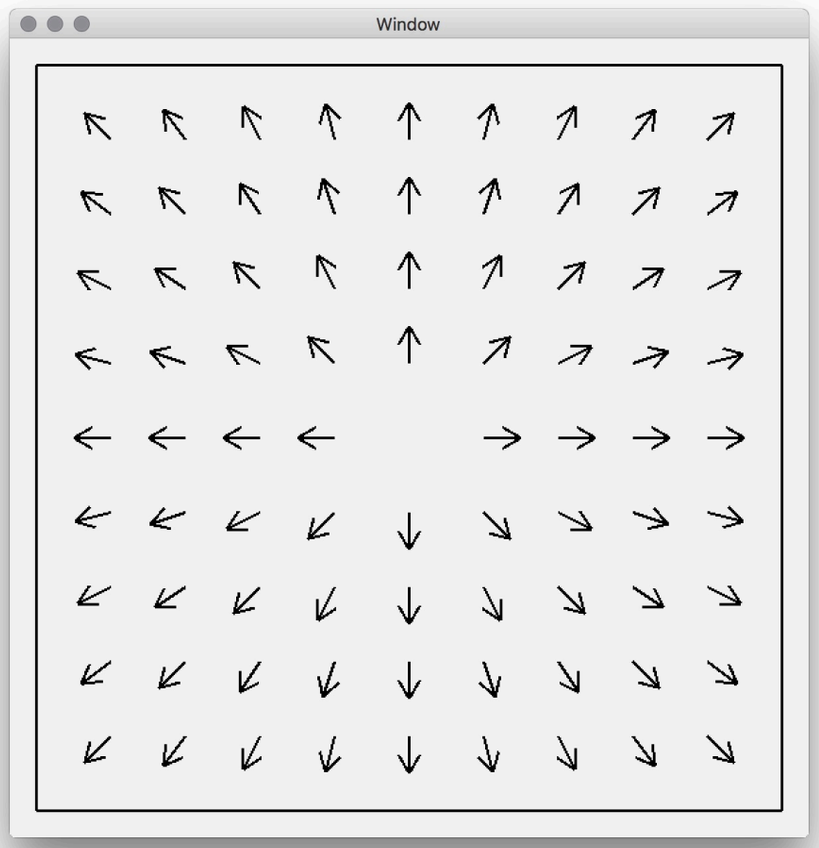
~2_VectorField.c~

```c
#include<stdio.h>
#include<glsc3d_3.h>

int main()
{
     g_init("Window", 600, 600);   //Pixel Size
     g_def_scale_2D(0,             //ID
                    -1, 1,         //xmin,xmax
                    -1, 1,         //ymin,ymax
                    20.0, 20.0, //Window (Left, Top) Position
                    560, 560);  //Window Size (x,y)
     g_cls();                      //Clear window
     g_sel_scale(0);                    //Select Virtual scale
     g_boundary();                 //Draw Boundary
     int Imax = 10;
     int Jmax = 10;
     double VecX; double VecY;
     for(int i = 1; i < Imax; i ++)
     {
          for(int j = 1; j < Jmax; j ++)
          {
            double x = i*0.2 - 1.0, y = j*0.2 - 1.0;
            VecX = x; VecY = y;     // Divergence
            //VecX = -y; VecY = x; // Rotation

            g_arrow_2D(x, y,        // Base Point
                       VecX, VecY,  // Direction
                       0.1,         // Size of Arrow Length
                       0.05,        // Size of Arrow Head
                       2);          // Arrow kinds
          }
     }
     g_finish();                        //flush Draw buffer
     g_sleep(10.0);                //Sleep 10 sec
     return 0;
}
```

2_VectorField.c

2DのGLSC3D

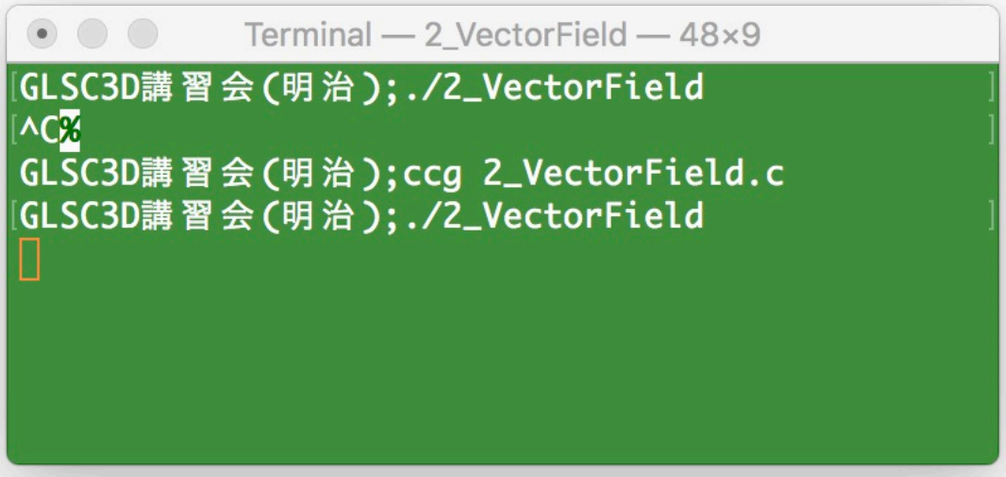
~2_VectorField.c~



実行結果

$ ccg 2_VectorField.cとうつ.
2_VectorField.cという名前の
実行ファイルができる.

$ ./2_VectorFieldとうつ.



2_VectorField.c

2DのGLSC3D

~2_VectorField.c~

解説

←1_CreateWindow.cと同じ

```c
#include<stdio.h>
#include<glsc3d_3.h>

int main()
{
     g_init("Window", 600, 600);   //Pixel Size
     g_def_scale_2D(0,             //ID
                    -1, 1,         //xmin,xmax
                    -1, 1,         //ymin,ymax
                    20.0, 20.0, //Window (Left, Top) Position
                    560, 560);  //Window Size (x,y)
     g_cls();                      //Clear window
     g_sel_scale(0);                    //Select Virtual scale
     g_boundary();                 //Draw Boundary
     int Imax = 10;
     int Jmax = 10;
     double VecX; double VecY;
     for(int i = 1; i < Imax; i ++)
     {
          for(int j = 1; j < Jmax; j ++)
          {
            double x = i*0.2 - 1.0, y = j*0.2 - 1.0;
            VecX = x; VecY = y;      // Divergence
            //VecX = -y; VecY = x; // Rotation

            g_arrow_2D(x, y,         // Base Point
                       VecX, VecY,  // Direction
                       0.1,          // Size of Arrow Length
                       0.05,         // Size of Arrow Head
                       2);           // Arrow kinds
          }
     }
     g_finish();                        //flush Draw buffer
     g_sleep(10.0);                //Sleep 10 sec
     return 0;
}
```

←ーーーーーーーーベクトルの始点を計算する.
←ーーーーーーーーベクトルの終点を計算する.

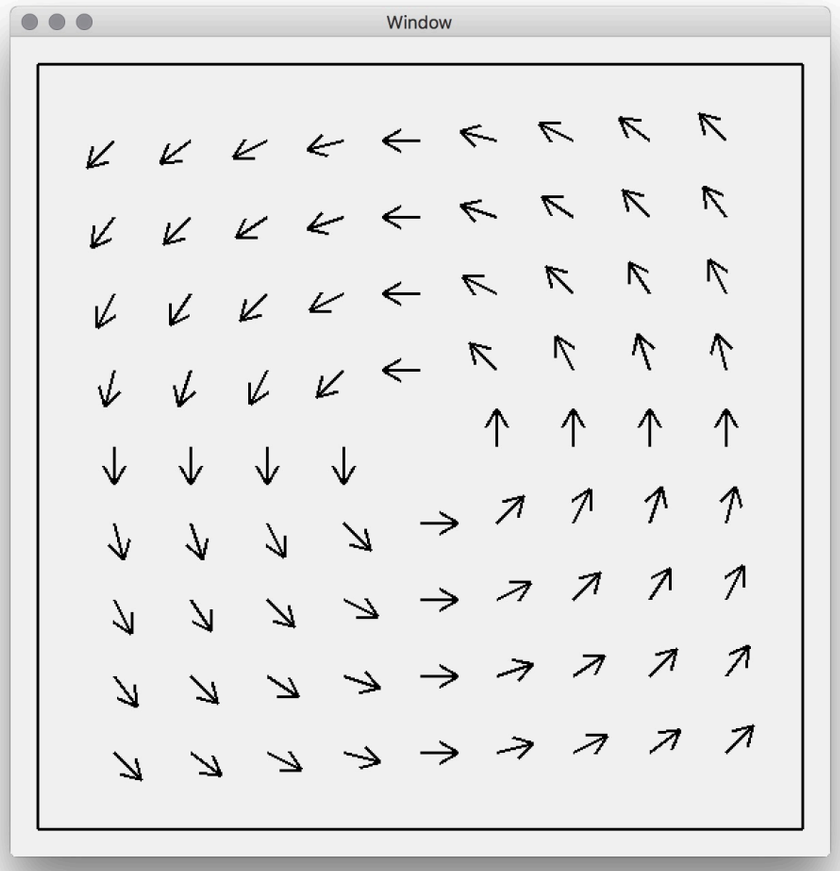
←ーーーーーーーーベクトルの始点を代入する.
←ーーーーーーーーベクトルの終点を代入する.
←ーーーーーーーーーーベクトルの長さを指定
←ーーーーーーーーーーベクトルの頭の大きさを指定
←ーーーーーーーーーーベクトルの頭の大きさを指定

2_VectorField.c

2DのGLSC3D

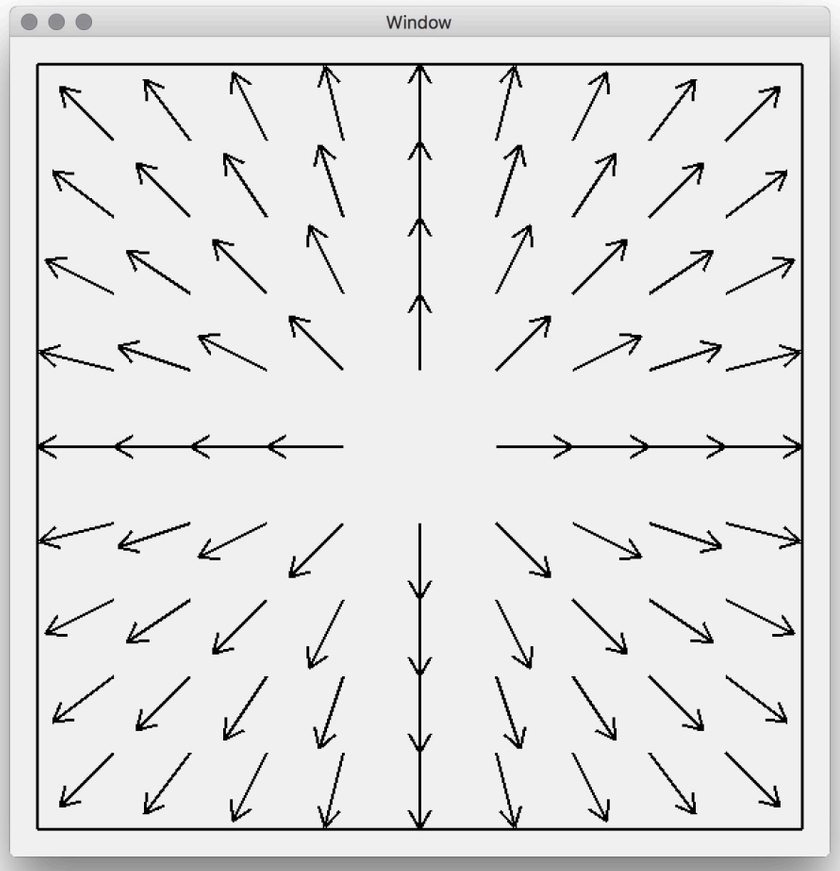
~2_VectorField.c~

演習：28行目を有効化せよ.



実行結果

演習：g_arrow_2Dの引数を変更せよ



実行結果

2_VectorField.c

2DのGLSC3D

~3_Animation.c~

```
#include <...>
...
#include <glsc3d_3.h>

int main()
{
    g_init(...); //用紙の設定
    g_def_scale_3D(0,...); //0 番の自由座標系の定義
    g_def_scale_3D(1,...); //1 番の自由座標系の定義
    ...

/****************************************/
                数値計算
/****************************************/

    g_cls(); //用紙を背景色で塗りつぶす

    //自由座標系を選択→属性を指定→描画関数で描画.
    g_sel_scale(0); //0 番の自由座標系を選択（以下，0 番の自由座標系に描かれる．）
    g_area_color(...); //塗りつぶしの色を指定
    g_box_3D(...); //3 次元空間に box を描画
    ...

    //自由座標系を選択→属性を指定→描画関数で描画.
    g_sel_scale(1); //1 番の自由座標系を選択（以下，1 番の自由座標系に描かれる．）
    g_area_color(...); //塗りつぶしの色を指定
    g_sphere_3D(...); //3 次元空間に sphere を描画
    ...

    g_finish(); //描画する
    return 0;
}
```

構成は左のようになるが
アニメーションの場合は，
ループとなる．

3_Animation.c

2DのGLSC3D

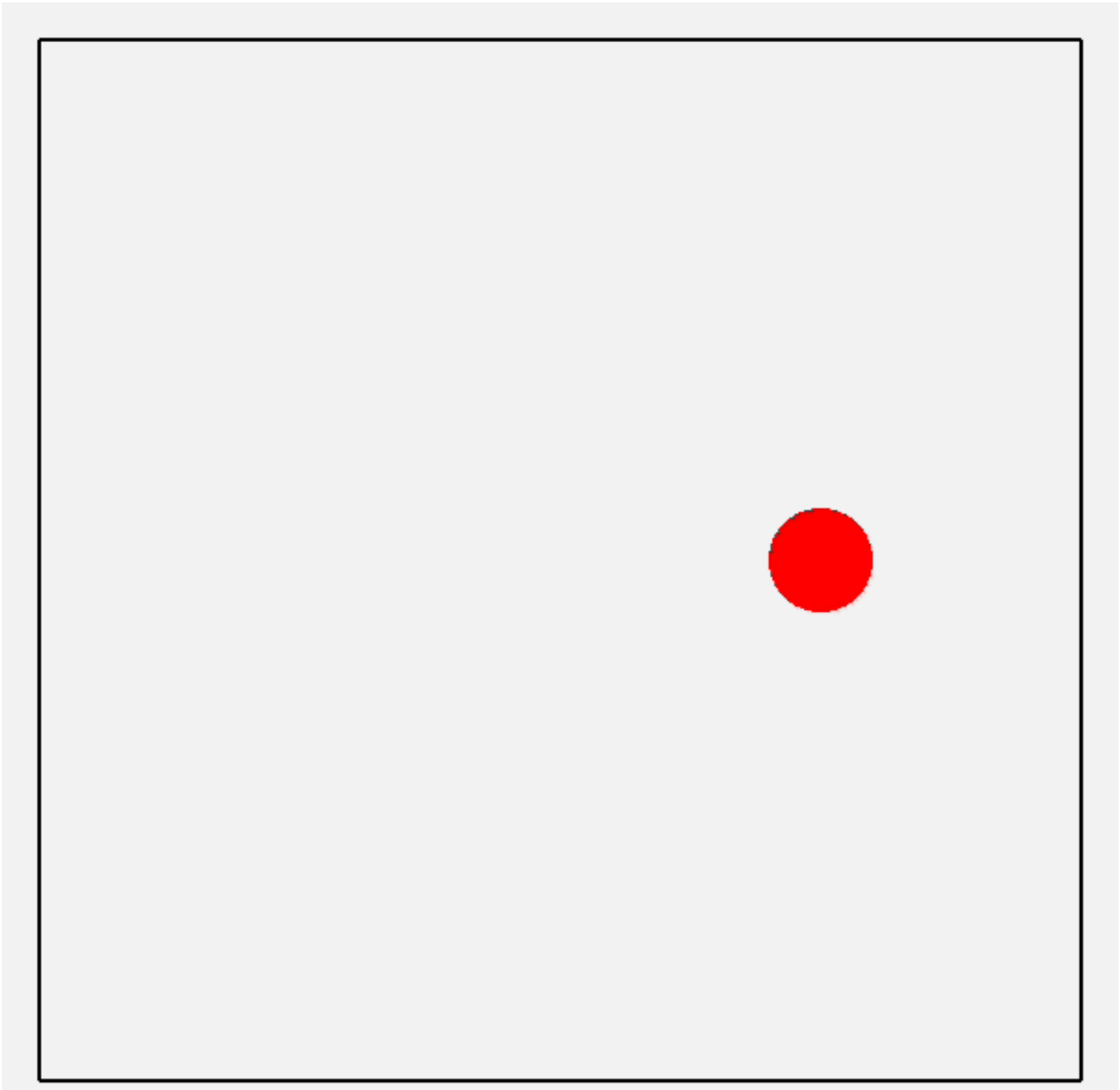
~3_Animation.c~

```c
#include<stdio.h>
#include<glsc3d_3.h>

int main()
{
    int INTV = 100;
    g_init("Window", 600, 600);                    //Pixel Size
    g_def_scale_2D(0,                              //ID
                   -1, 1,                          //xmin,xmax
                   -1, 1,                          //ymin,ymax
                   20.0, 20.0,                     //Window (Left, Top) Position
                   560, 560);                      //Window Size (x,y)

    ////////////// Start time loop ////////////
    for (int i_time = 0; ; i_time++) {
        double dt = 0.0001;                        //Time Discritization
        double t = i_time * dt;                    //Time
        ////////////// Calculation Part ////////////
        double x = 0.5 * cos(2.0*M_PI * t);     //X Coordinate
        double y = 0.5 * sin(2.0*M_PI * t);     //Y Coordinate
        ////////////// Draw Part ////////////
        if (i_time%INTV == 0) {
            g_cls();                           //Clear window
            g_sel_scale(0);                        //Select Virtual scale
            g_boundary();                          //Draw Boundary
            g_area_color(1,0,0,1);                 //Area Color
            g_circle_2D(x, y, 0.1, G_NO, G_YES);//g_circle_2D
            g_finish();                            //flush Draw buffer
            g_sleep(0.01);                         //Sleep 0.01 sec
        }
    }
    return 0;
}
```

3_Animation.c

2DのGLSC3D

~3_Animation.c~

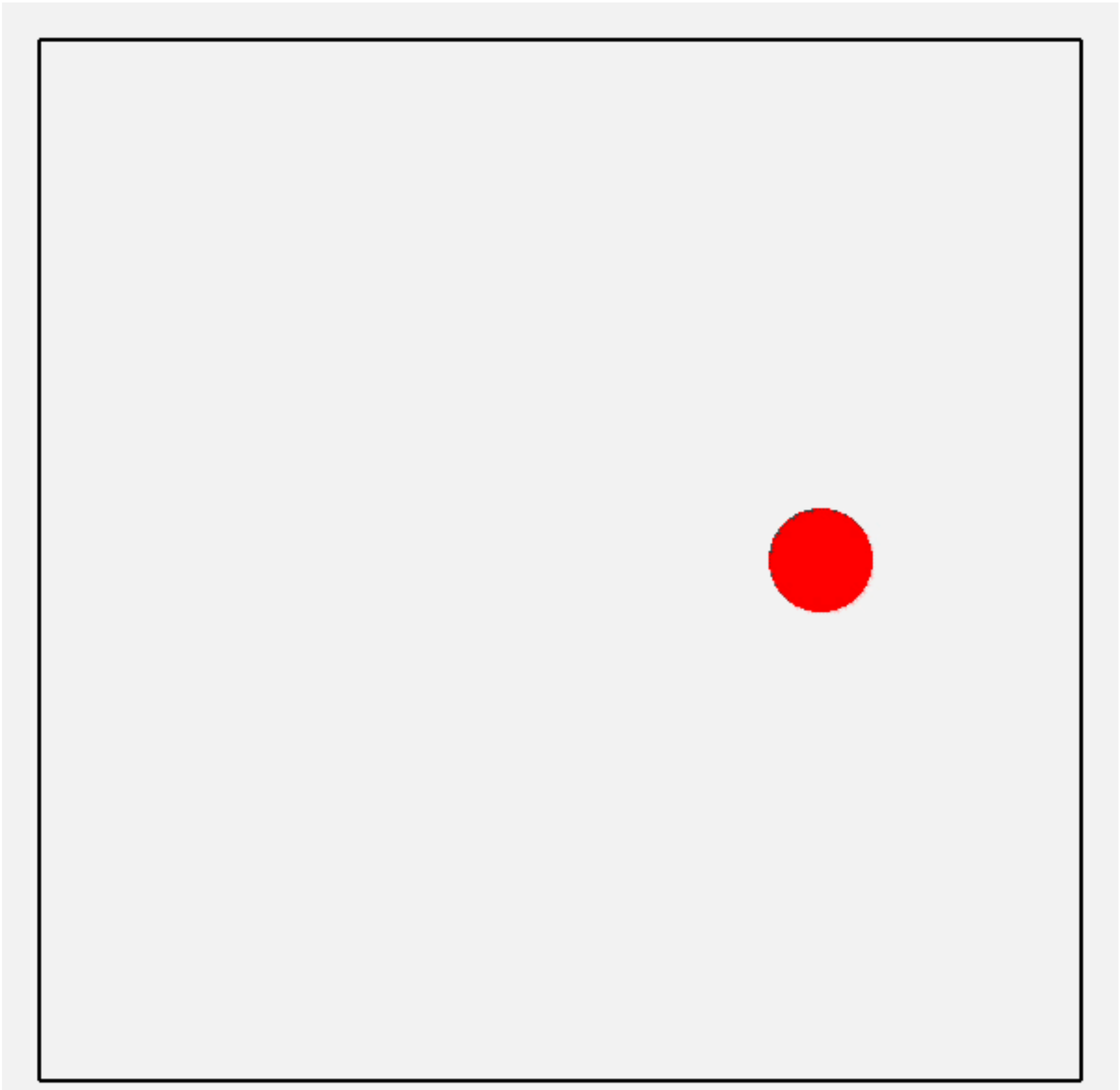


コンパイル＆実行せよ

3_Animation.c

2DのGLSC3D

~3_Animation.c~



コンパイル＆実行せよ

3_Animation.c

2DのGLSC3D

~3_Animation.c~

解説

```c
#include<stdio.h>
#include<glsc3d_3.h>

int main()
{
    int INTV = 100;
    g_init("Window", 600, 600);                    //Pixel Size
    g_def_scale_2D(0,                              //ID
                   -1, 1,                          //xmin,xmax
                   -1, 1,                          //ymin,ymax
                   20.0, 20.0,                     //Window (Left, Top) Position
                   560, 560);                      //Window Size (x,y)

    ////////////// Start time loop //////////////
    for (int i_time = 0; ; i_time++) {                                  ←――――――――時間ループ開始
        double dt = 0.0001;                        //Time Discritization
        double t = i_time * dt;                    //Time
        ////////////// Calculation Part //////////////
        double x = 0.5 * cos(2.0*M_PI * t);        //X Coordinate
        double y = 0.5 * sin(2.0*M_PI * t);        //Y Coordinate      ←――――――――Diskの場所を計算する
        ////////////// Draw Part //////////////
        if (i_time%INTV == 0) {
            g_cls();                               //Clear window      ←――――――消して，座標を選んで，枠を描いて
            g_sel_scale(0);                        //Select Virtual scale
            g_boundary();                          //Draw Boundary
            g_area_color(1,0,0,1);                 //Area Color        ←――――――――――Diskの色を指定して
            g_circle_2D(x, y, 0.1, G_NO, G_YES);//g_circle_2D         ←――――――――――Diskを塗りつぶして
            g_finish();                            //flush Draw buffer ←――――描画命令を吐き出して，
            g_sleep(0.01);                         //Sleep 0.01 sec    ←――――――――――0.01秒休む
        }
    }
    return 0;
}
```

←1_CreateWindow.cと同じ

3_Animation.c

2DのGLSC3D

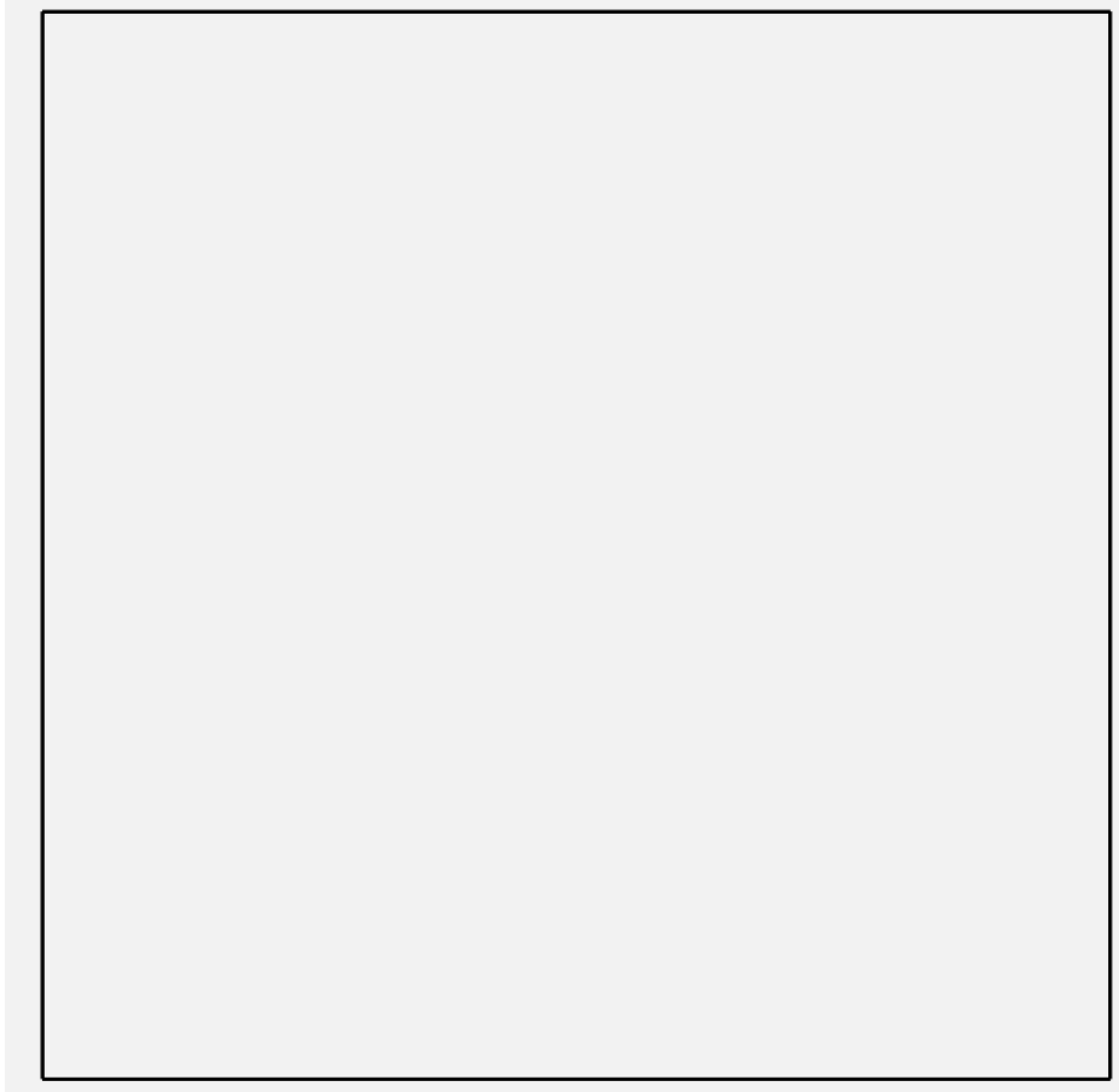
~4_Wave_2D_Contln.c~

```
g_init("Window", 600, 600);      //Pixel Size

   g_def_scale_2D(0,                  //ID
                  -Lx*0.5, Lx*0.5,              //xmin,xmax
                  -Ly*0.5, Ly*0.5,     //ymin,ymax
                  20.0, 20.0,          //Window (Left, Top) Position
                  560, 560);           //Window Size (x,y)

   //Set initial calculation
  {
   }
   /////////////// Start time loop ////////////
   for (int i_time = 0; ; i_time++) {

      /////////////// Draw Part ////////////
      if (i_time%INTV == 0) {
         g_cls();                                //Clear window
         g_sel_scale(0);                         //Select Virtual scale
         g_line_color(0.0, 0.0, 0.0, 1.0);
         g_boundary();                           //Draw Boundary

         g_line_color(1.0, 0.0, 0.0, 1.0);
        g_contln_2D(-Lx*0.5 + dx*0.5, Lx*0.5 - dx*0.5, -Ly*0.5 + dy*0.5, Ly*0.5 - dy*0.5,
                    N+2, M+2, u0, 0.01); //Contln_0.01
         g_line_color(0.0, 0.0, 1.0, 1.0);
         g_contln_2D(-Lx*0.5 + dx*0.5, Lx*0.5 - dx*0.5, -Ly*0.5 + dy*0.5, Ly*0.5 - dy*0.5,
                    N+2, M+2, u0, 0.04); //Contln_0.04

         g_finish();                                   //flush Draw buffer
         g_sleep(0.01);                          //Sleep 0.01 sec
      }

      /////////////// Calculation Part ////////////
      {
      }
}
```

4_Wave_2D_Contln.c

2DのGLSC3D

~4_Wave_2D_Contln.c~

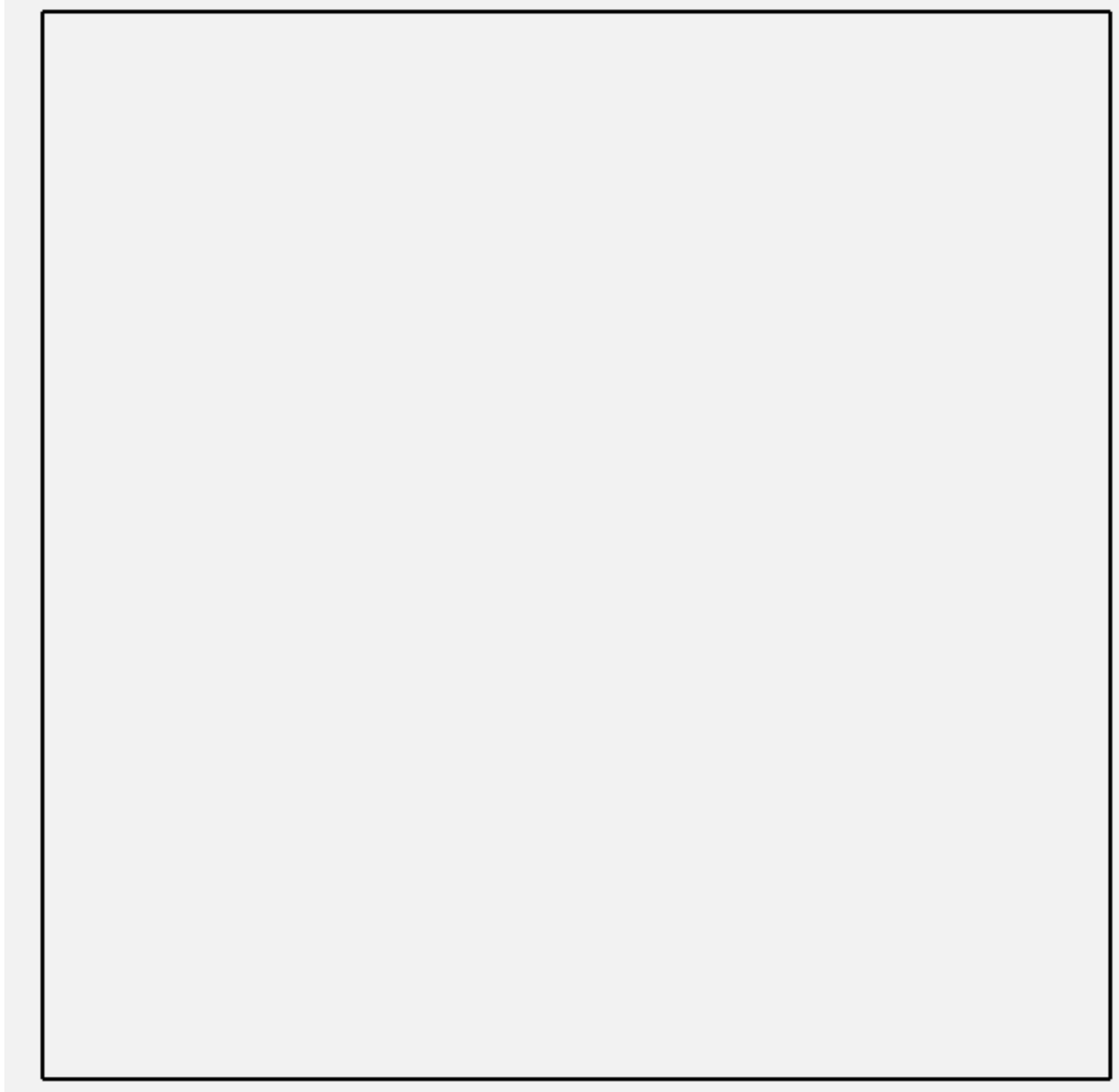


コンパイル＆実行せよ

4_Wave_2D_Contln.c

2DのGLSC3D

~4_Wave_2D_Contln.c~



コンパイル＆実行せよ

4_Wave_2D_Contln.c

2DのGLSC3D

~4_Wave_2D_Contln.c~

解説

```
g_init("Window", 600, 600);      //Pixel Size

    g_def_scale_2D(0,                  //ID
                -Lx*0.5, Lx*0.5,              //xmin,xmax
                -Ly*0.5, Ly*0.5,     //ymin,ymax
                20.0, 20.0,          //Window (Left, Top) Position
                560, 560);           //Window Size (x,y)

    //Set initial calculation
  {
    }
    ////////////// Start time loop //////////////
    for (int i_time = 0; ; i_time++) {

        ////////////// Draw Part //////////////
        if (i_time%INTV == 0) {
            g_cls();                                //Clear window
            g_sel_scale(0);                         //Select Virtual scale
            g_line_color(0.0, 0.0, 0.0, 1.0);
            g_boundary();                           //Draw Boundary

            g_line_color(1.0, 0.0, 0.0, 1.0);
           g_contln_2D(-Lx*0.5 + dx*0.5, Lx*0.5 - dx*0.5, -Ly*0.5 + dy*0.5, Ly*0.5 - dy*0.5,
                       N+2, M+2, u0, 0.01); //Contln_0.01
            g_line_color(0.0, 0.0, 1.0, 1.0);
            g_contln_2D(-Lx*0.5 + dx*0.5, Lx*0.5 - dx*0.5, -Ly*0.5 + dy*0.5, Ly*0.5 - dy*0.5,
                       N+2, M+2, u0, 0.04); //Contln_0.04

            g_finish();                                 //flush Draw buffer
            g_sleep(0.01);                          //Sleep 0.01 sec
        }

    ////////////// Calculation Part //////////////
    {
    }
}
```

基本的な構成は3_Animation.c
と同じ．配列u0に波動方程式の
数値解が入っている．

g_contln_2Dは等高線を描く関数

4_Wave_2D_Contln.c

2DのGLSC3D

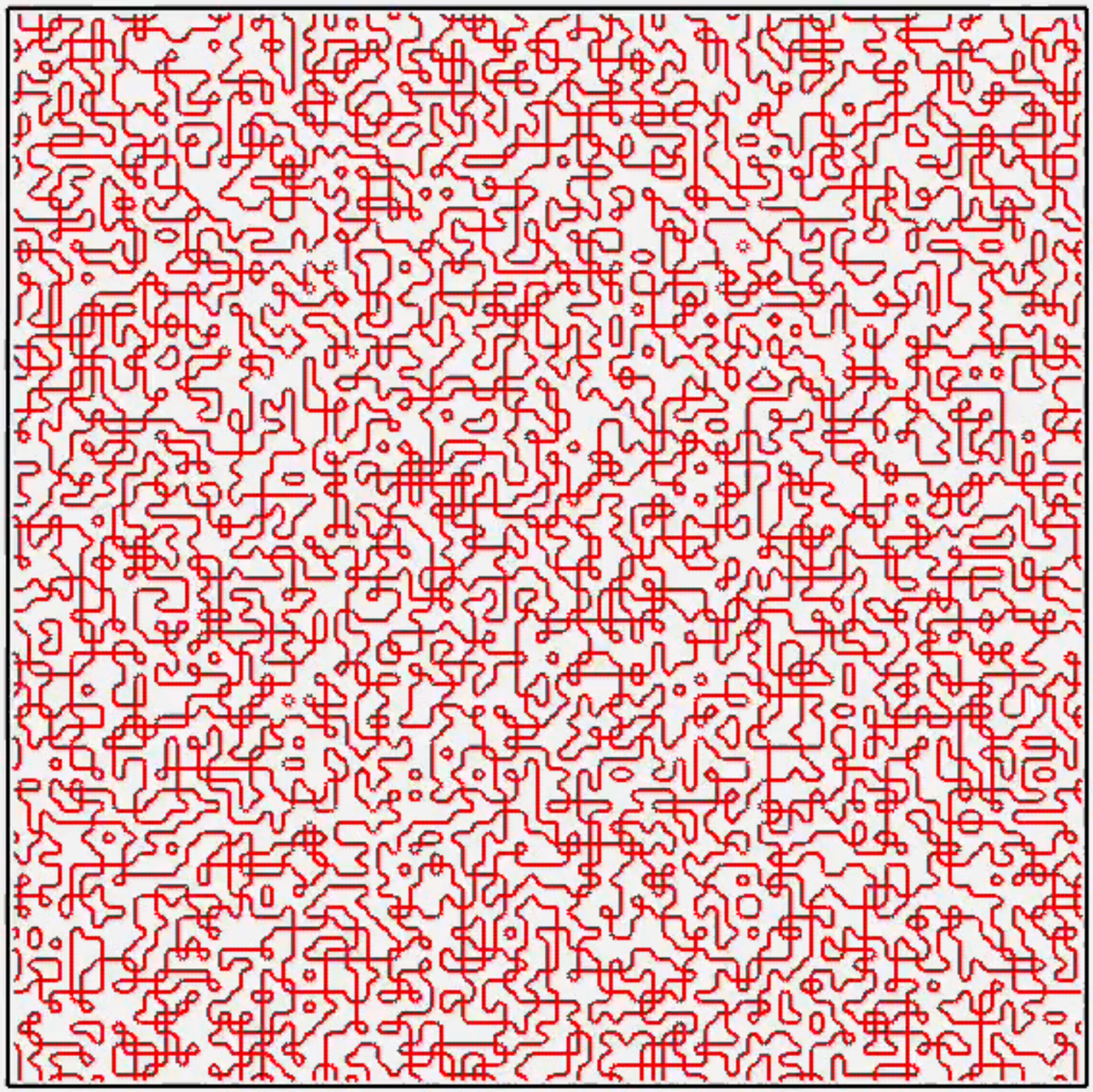
~5_Turing_2D_Contln.c~

```
double f(double u, double v)
{
    return u * (1 - u * u) - v;
}
double g(double u, double v)
{
    return 3.0 * u - 2.0 * v;
//    return 3.0 * (u + 0.05) - 2.0 * v;
//    return 3.0 * (u - 0.05) - 2.0 * v;
}
int main()
{
    g_init("Window", 600, 600);  //Pixel Size

    g_def_scale_2D(0,                  //ID
                -Lx*0.5, Lx*0.5,              //xmin,xmax
                -Ly*0.5, Ly*0.5,     //ymin,ymax
                20.0, 20.0,          //Window (Left, Top) Position
                560, 560);           //Window Size (x,y)
//Set initial calculation
 /////////// Start time loop ///////////
 for (int i_time = 0; ; i_time++) {
    /////////// Draw Part ///////////
    if (i_time%INTV == 0) {
        g_cls();                                //Clear window
        g_sel_scale(0);                         //Select Virtual scale
        g_line_color(0.0, 0.0, 0.0, 1.0);
        g_boundary();                           //Draw Boundary

        g_line_color(1.0, 0.0, 0.0, 1.0);
        g_contln_2D(-Lx*0.5 + dx*0.5, Lx*0.5 - dx*0.5, -Ly*0.5 + dy*0.5, Ly*0.5 - dy*0.5,
                          N+2, M+2, u0, 0.0); //Contln_0.0

        g_finish();                                     //flush Draw buffer
                          g_capture();
        g_sleep(0.01);                          //Sleep 0.01 sec
    }
    /////////// Calculation Part ///////////
}
```

5_Turing_2D_Contln.c

2DのGLSC3D

~5_Turing_2D_Contln.c~

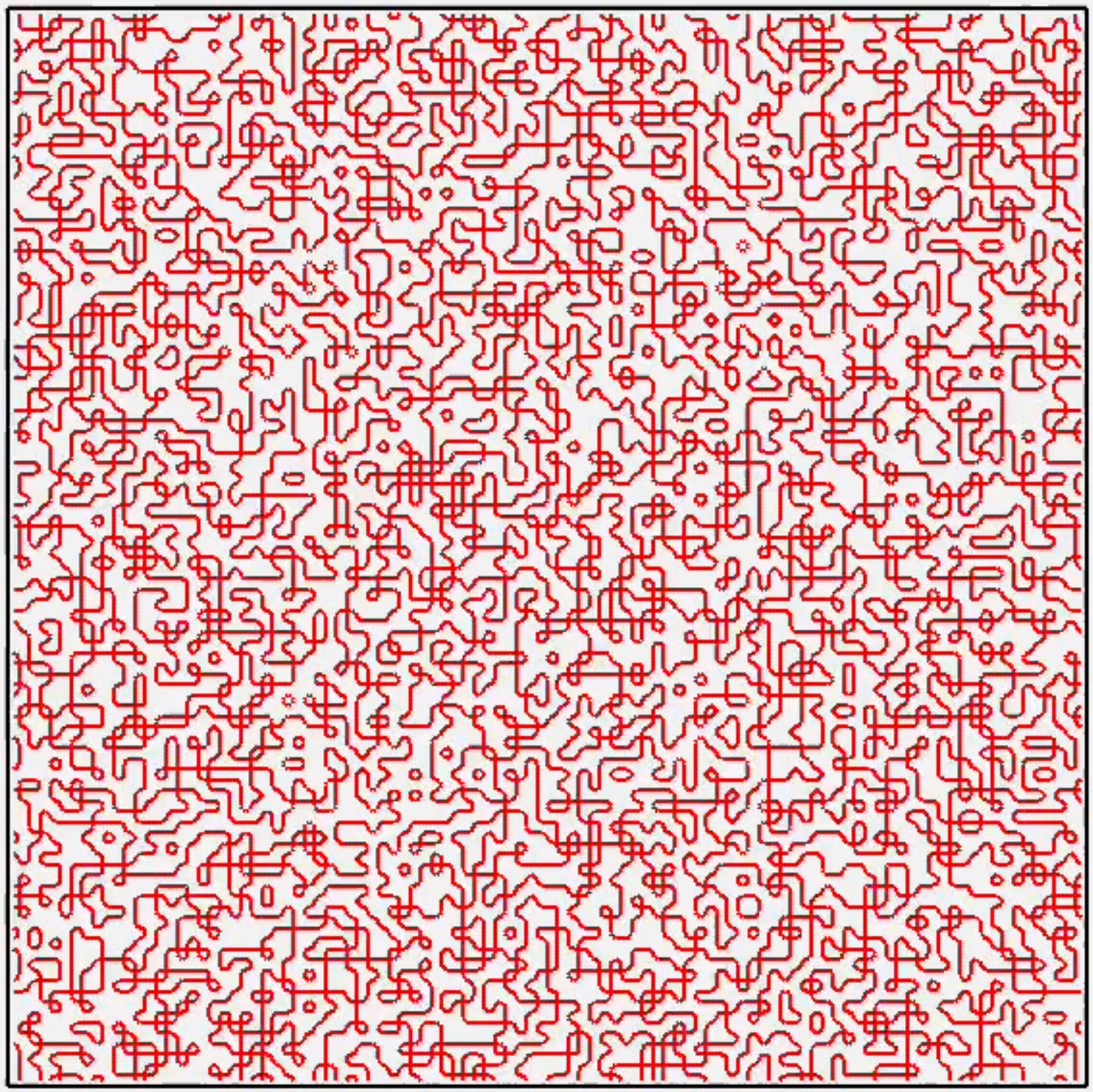


コンパイル＆実行せよ

5_Turing_2D_Contln.c

2DのGLSC3D

~5_Turing_2D_Contln.c~



コンパイル＆実行せよ

5_Turing_2D_Contln.c

2DのGLSC3D

~5_Turing_2D_Contln.c~

基本的な構成は3_Animation.c
と同じ．配列u0にTuringの
数値解が入っている．

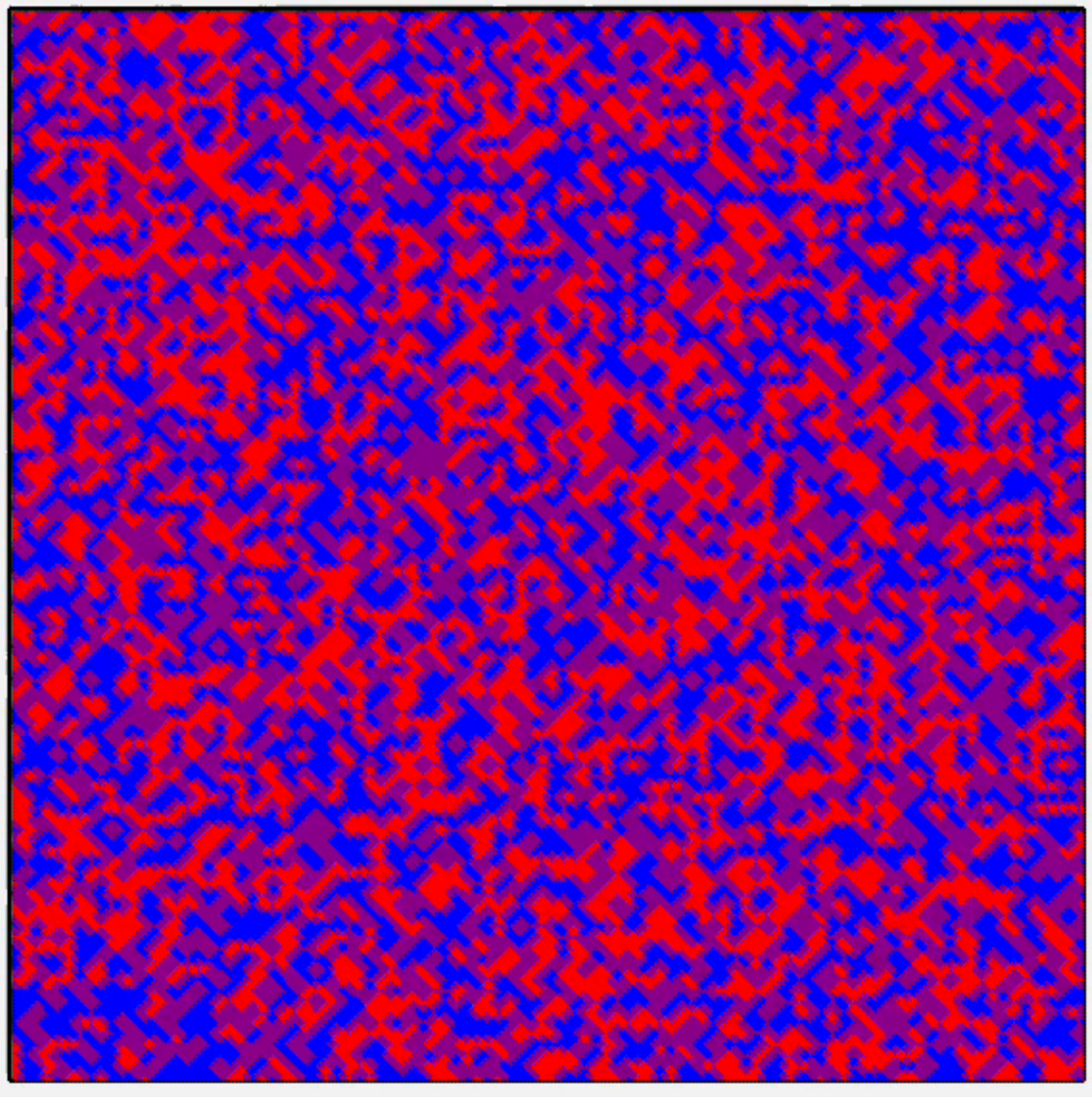
g_contln_2Dは等高線を描く関数

5_Turing_2D_Contln.c

```
double f(double u, double v)
{
    return u * (1 - u * u) - v;
}
double g(double u, double v)
{
    return 3.0 * u - 2.0 * v;
//    return 3.0 * (u + 0.05) - 2.0 * v;
//    return 3.0 * (u - 0.05) - 2.0 * v;
}
int main()
{
    g_init("Window", 600, 600);  //Pixel Size

    g_def_scale_2D(0,                  //ID
                -Lx*0.5, Lx*0.5,              //xmin,xmax
                -Ly*0.5, Ly*0.5,     //ymin,ymax
                20.0, 20.0,          //Window (Left, Top) Position
                560, 560);           //Window Size (x,y)
//Set initial calculation
 /////////// Start time loop ///////////
 for (int i_time = 0; ; i_time++) {
    /////////// Draw Part ///////////
    if (i_time%INTV == 0) {
        g_cls();                                //Clear window
        g_sel_scale(0);                         //Select Virtual scale
        g_line_color(0.0, 0.0, 0.0, 1.0);
        g_boundary();                           //Draw Boundary

        g_line_color(1.0, 0.0, 0.0, 1.0);
        g_contln_2D(-Lx*0.5 + dx*0.5, Lx*0.5 - dx*0.5, -Ly*0.5 + dy*0.5, Ly*0.5 - dy*0.5,
                        N+2, M+2, u0, 0.0); //Contln_0.0

        g_finish();                                     //flush Draw buffer
                        g_capture();
        g_sleep(0.01);                          //Sleep 0.01 sec
    }
    /////////// Calculation Part ///////////
}
```

2DのGLSC3D

~6_Turing_2D_ColorMap.c~



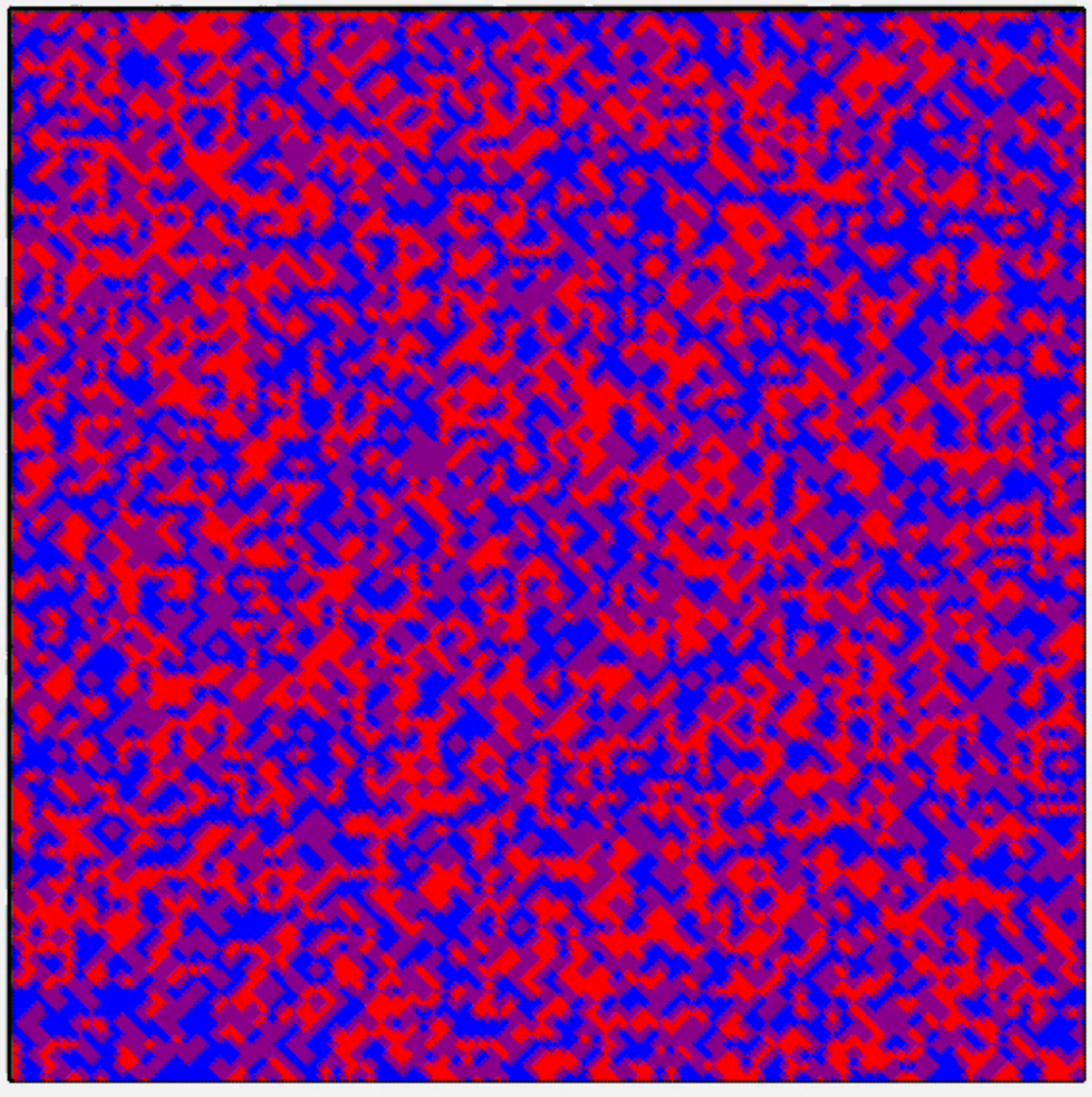
演習：８２行目と８３行目を有効化せよ

数値計算の部分は同じ！

可視化の部分だけが違う

2DのGLSC3D

~6_Turing_2D_ColorMap.c~



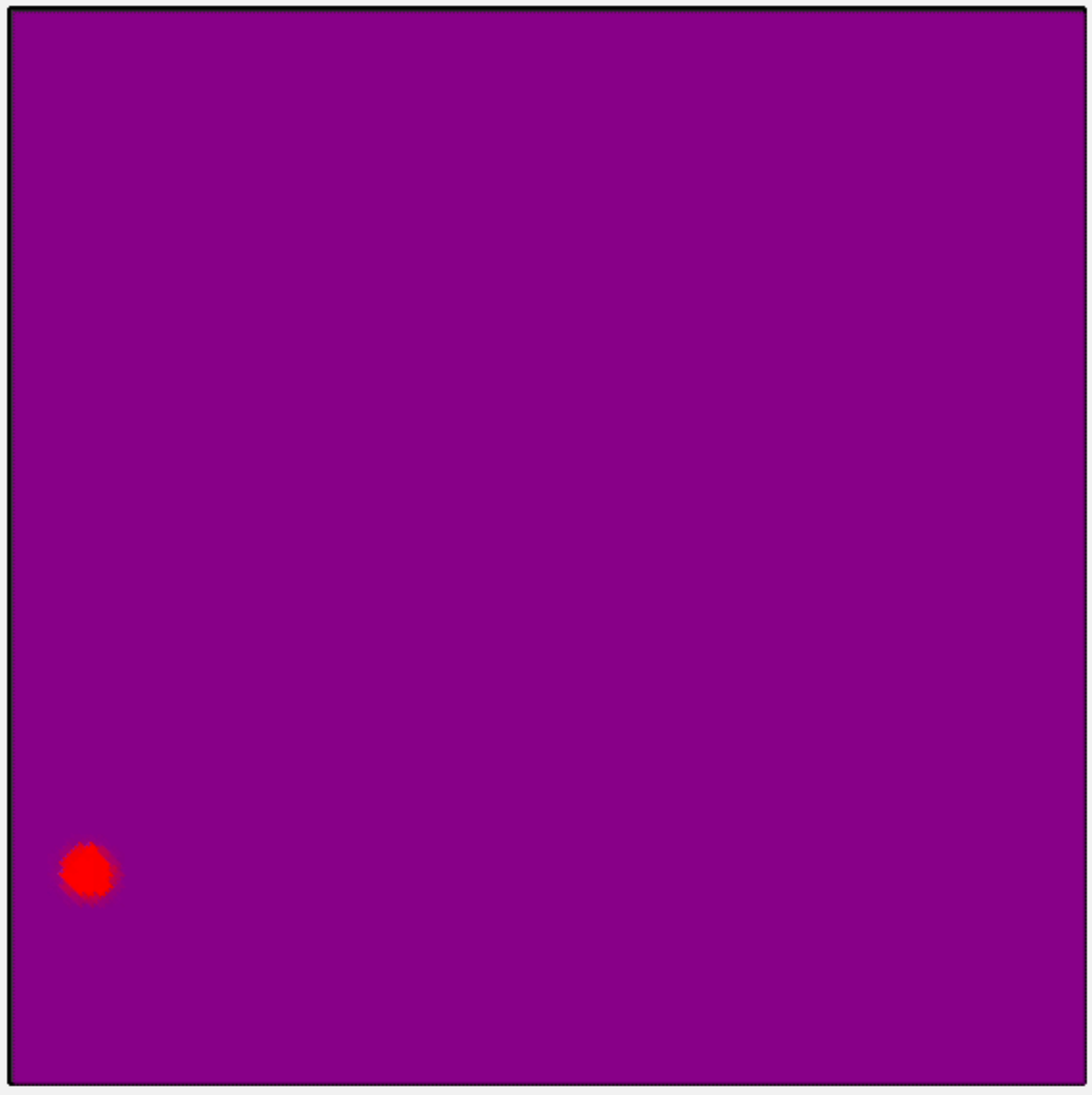
演習：８２行目と８３行目を有効化せよ

数値計算の部分は同じ！

可視化の部分だけが違う

2DのGLSC3D

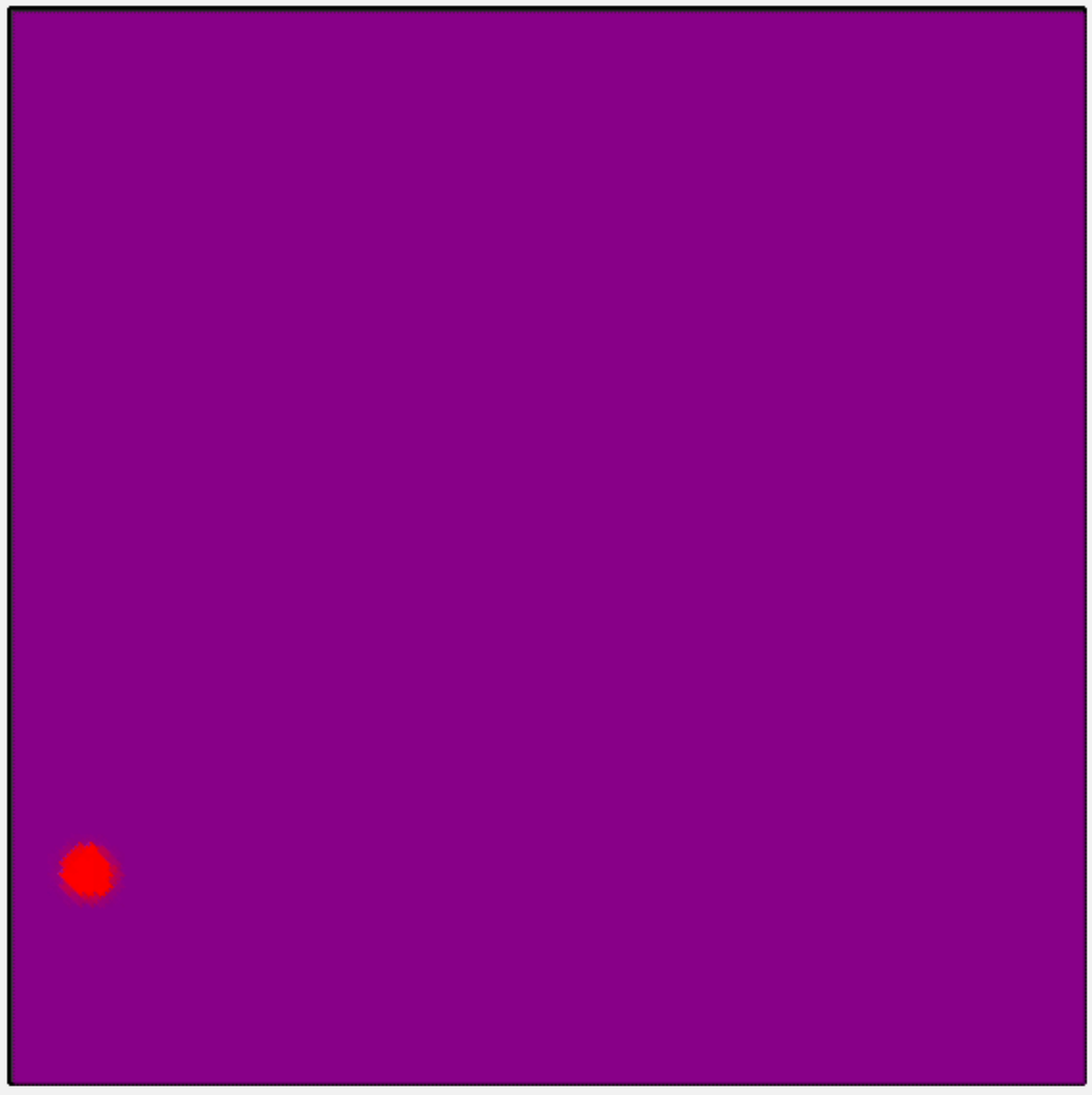
~7_Wave_2D_ColorMap.c~



数値計算の部分は同じ！
可視化の部分だけが違う

2DのGLSC3D

~7_Wave_2D_ColorMap.c~



数値計算の部分は同じ！
可視化の部分だけが違う

2DのGLSC3D

~8_Wave_2D_BirdView.c~

```
g_def_scale_2D(0,                  //ID
-Lx*0.5, Lx*0.5,                   //xmin,xmax
-Ly*0.5, Ly*0.5,     //ymin,ymax
20.0, 20.0,          //Window (Left, Top) Position
560, 560);           //Window Size (x,y)
```

```
g_def_scale_3D_fix(0,                  //ID
-Lx*0.5, Lx*0.5,                   //xmin,xmax
-Ly*0.5, Ly*0.5,     //ymin,ymax
-Lz*0.5, Lz*0.5,     //zmin,zmax
20.0, 20.0,          //Window (Left, Top) Position
560, 560);           //Window Size (x,y)
```

```
g_cls();                                  //Clear window
g_sel_scale(0);                           //Select Virtual scale
g_line_color(0.0, 0.0, 0.0, 1.0);
g_boundary();                             //Draw Boundary
g_line_color(1.0, 0.0, 0.0, 1.0);
g_contln_2D(-Lx*0.5 + dx*0.5,
 Lx*0.5 - dx*0.5,
-Ly*0.5 + dy*0.5,
 Ly*0.5 - dy*0.5,
 N+2, M+2, u0, 0.01); //Contln_0.01
g_line_color(0.0, 0.0, 1.0, 1.0);
g_contln_2D(
-Lx*0.5 + dx*0.5,
Lx*0.5 - dx*0.5,
-Ly*0.5 + dy*0.5,
Ly*0.5 - dy*0.5,
N+2, M+2, u0, 0.04); //Contln_0.04
g_finish();                                       //flush Draw buffer
g_sleep(0.01);                            //Sleep 0.01 sec
```

```
g_cls();                                  //Clear window
g_sel_scale(0);                           //Select Virtual scale
g_line_color(0.0, 0.0, 0.0, 1.0);
g_boundary();                             //Draw Boundary
g_area_color(1.0, 0.0, 0.0, 1.0);
g_bird_view_3D(
-Lx*0.5 + dx * 0.5,
 Lx*0.5 - dx * 0.5,
-Ly*0.5 + dy * 0.5,
 Ly*0.5 - dy * 0.5,
 N+2, M+2, u0, G_NO, G_YES); //Bird View
g_finish();                                       //flush Draw buffer
g_sleep(0.01);                            //Sleep 0.01 sec
```

4_Wave_2D_Contln.c

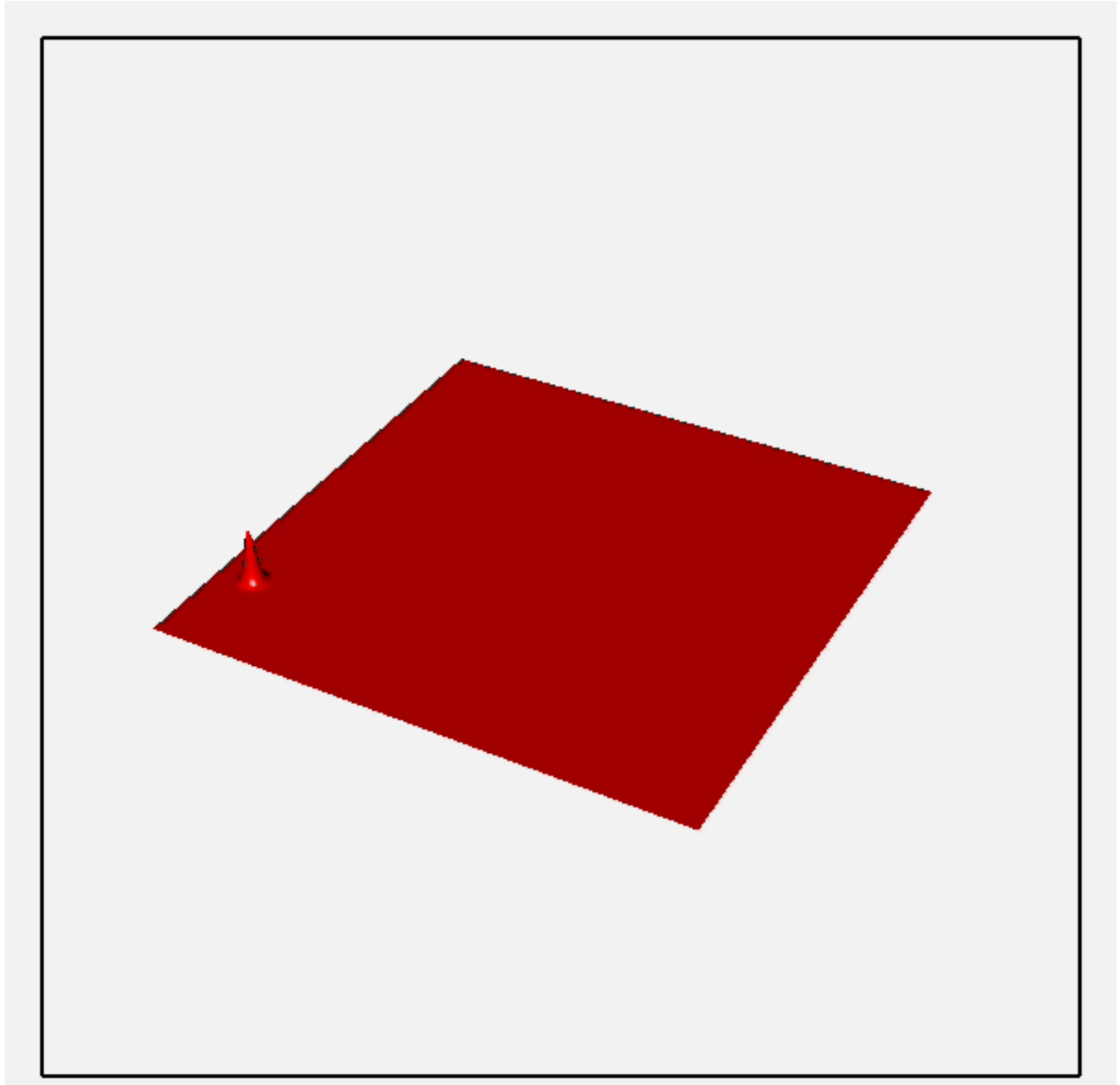
数値計算の部分は同じ！

可視化の部分だけが違う

8_Wave_2D_BirdView.c

2DのGLSC3D

~8_Wave_2D_BirdView.c~



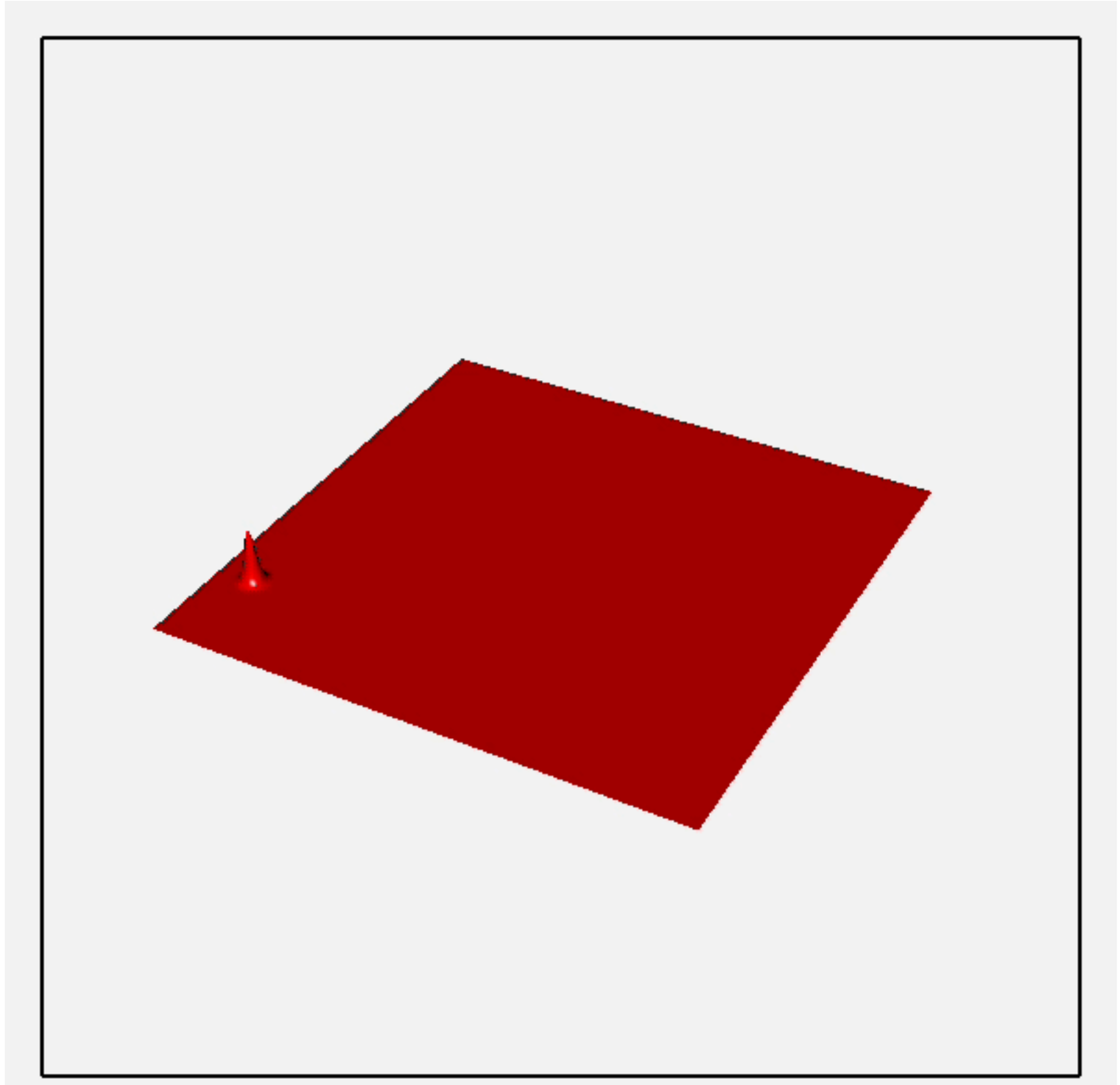
演習：５８～６２行目を
有効化するとどうなるか？

※g_vision関数に関しては
あとで述べる.

8_Wave_2D_BirdView.c

2DのGLSC3D

~8_Wave_2D_BirdView.c~



演習：５８～６２行目を
有効化するとどうなるか？

※g_vision関数に関しては
あとで述べる.

8_Wave_2D_BirdView.c

2DのGLSC3D

~9_Diffusion_1D~

```
g_init("Window", 600, 320);   //Pixel Size
g_def_scale_2D(0,                    //ID
               -L*0.5, L*0.5,        //xmin,xmax
               -0.2, 1.2,      //ymin,ymax
               20.0, 20.0,           //Window (Left, Top) Position
               560, 280);            //Window Size (x,y)
//Set initial calculation
{
}
////////////// Start time loop ////////////
for (int i_time = 0;t < 10 ; i_time++) {

    t = i_time * dt;

    ////////////// Draw Part ////////////
    if (i_time%INTV == 0) {
        g_cls();                              //Clear window
        g_sel_scale(0);                       //Select Virtual scale
        g_line_color(0.0, 0.0, 0.0, 1.0);
        g_boundary();                         //Draw Boundary
        g_line_width(1.0);
        g_line_color(0.0, 0.0, 0.0, 0.5);
        g_move_2D(-L*0.5, 0.0); g_plot_2D(L*0.5, 0.0);  //X Axis
        g_move_2D(0.0, -1.2); g_plot_2D(0.0, 1.2);      //Y Axis
        //Profile of u
        g_line_width(2.0);
        g_line_color(1.0, 0.0, 0.0, 1.0);
        for (int i=0; i<N-1; i++) {
            g_move_2D(i*dx-L*0.5, u[i]);
            g_plot_2D((i+1)*dx-L*0.5, u[i+1]);
        }

        g_text_size(18);
        g_text_color(1.0, 0.0, 0.0, 1.0);
        g_text_2D_virtual(L*0.5 - 0.4, 1.0, "u: -");

        g_finish();                               //flush Draw buffer
        g_sleep(0.0);                       //Sleep 0.0 sec
    }
    ////////////// Calculation Part ////////////
    {
    }
}
```

やはり基本的な構成は同じ.

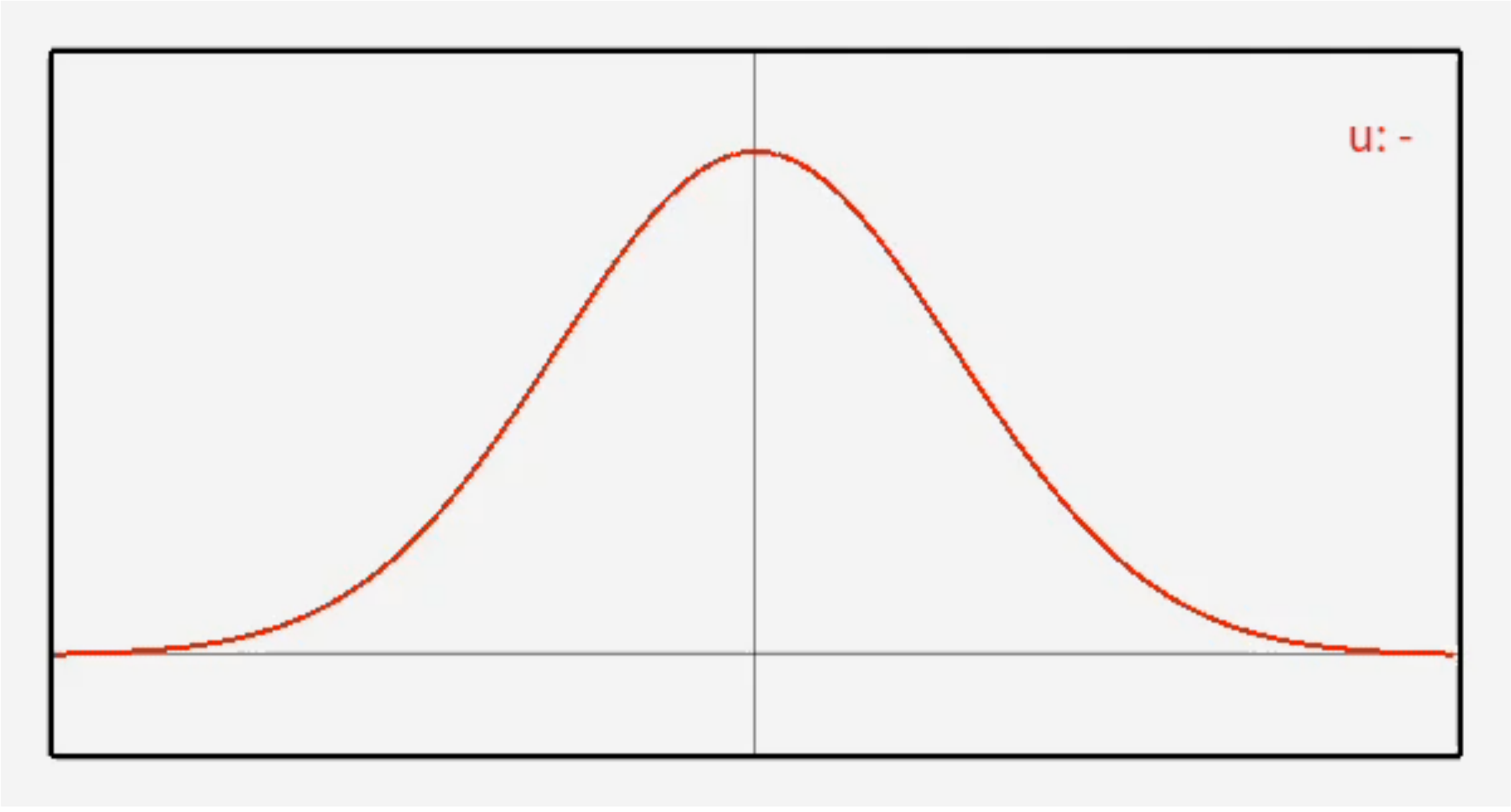
数値計算の部分は同じ！

可視化の部分だけが違う

9_Diffusion_1D.c

2DのGLSC3D

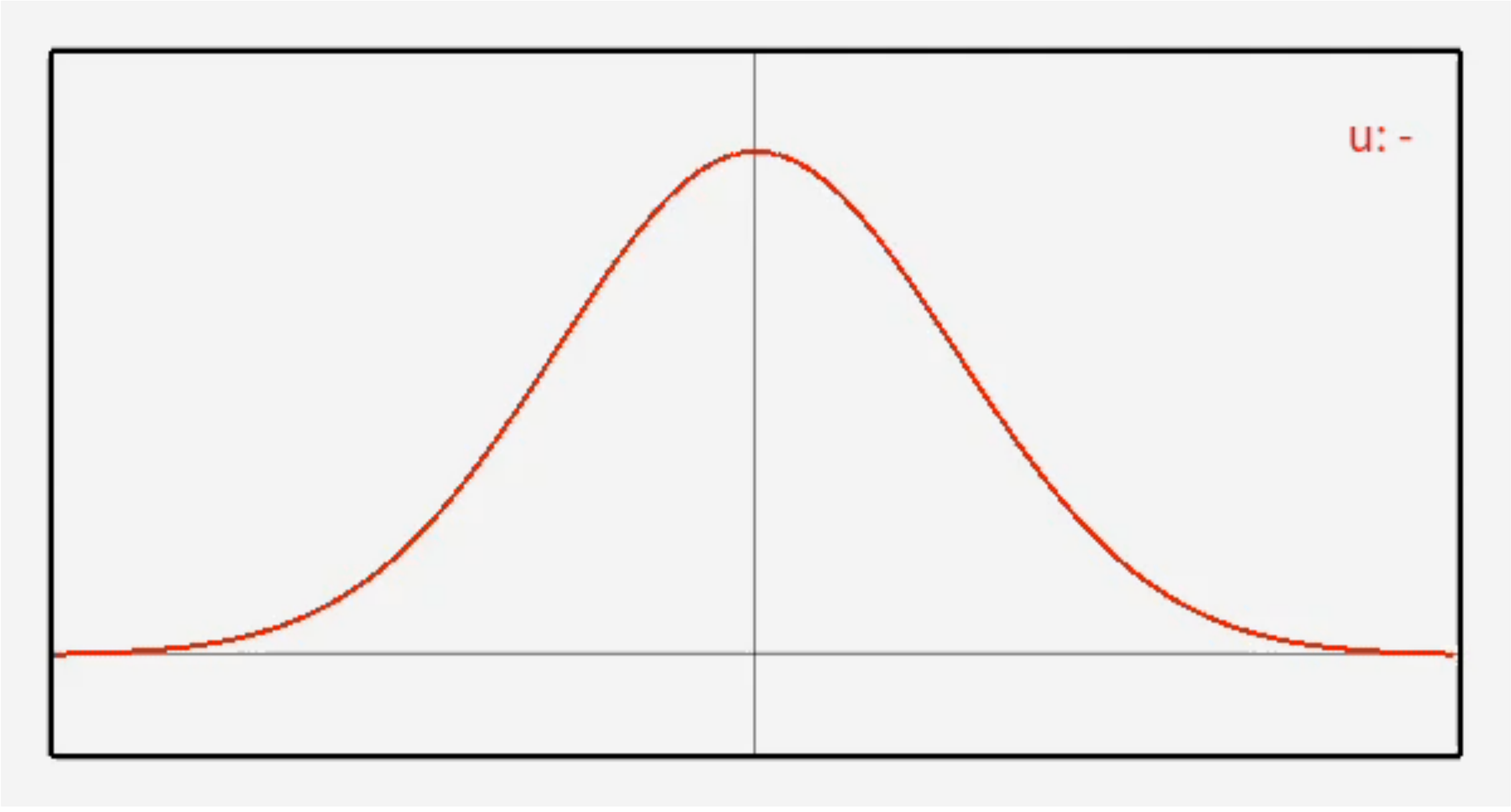
~9_Diffusion_1D~



9_Diffusion_1D.c

2DのGLSC3D

~9_Diffusion_1D~

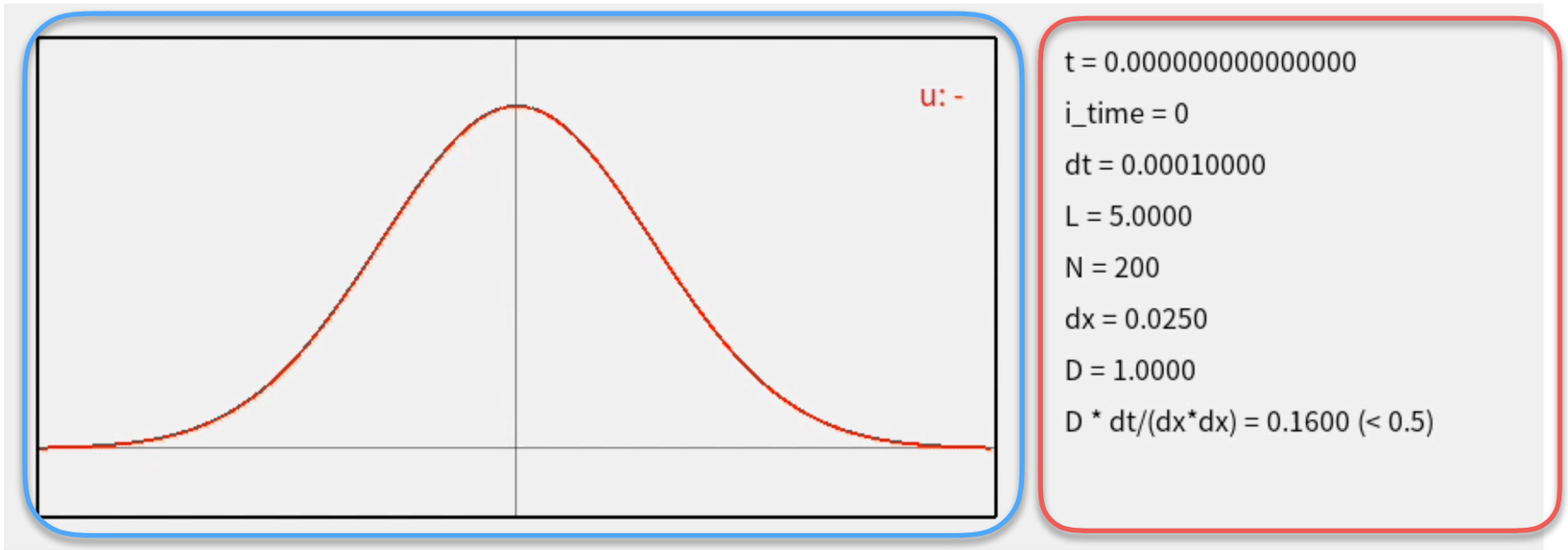


9_Diffusion_1D.c

2DのGLSC3D

~10_Diffusion_1D_Text~

```
g_init("Window", 900, 320);  //Pixel Size
g_def_scale_2D(0,                  //ID
               -L*0.5, L*0.5,                  //xmin,xmax
               -0.2, 1.2,              //ymin,ymax
               20.0, 20.0,              //Window (Left, Top) Position
               560, 280);              //Window Size (x,y)

g_def_scale_2D(1,                  //ID
               -L*0.5, L*0.5,                  //xmin,xmax
               -0.2, 1.2,              //ymin,ymax
               620.0, 20.0,              //Window (Left, Top) Position
               260, 280);              //Window Size (x,y)

//Set initial calculation
{
}
////////////// Start time loop //////////////
for (int i_time = 0; t < 10; i_time++) {
    t = i_time * dt;
    ////////////// Draw Part //////////////
    if (i_time%INTV == 0) {
        g_cls();                                                    //Clear window
        ////////////// Graph Part //////////////
        {
            g_sel_scale(0);                                         //Select Virtual scale
            g_line_color(0.0, 0.0, 0.0, 1.0);
            g_boundary();                                           //Draw Boundary

            g_line_width(1.0);
            g_line_color(0.0, 0.0, 0.0, 0.5);
            g_move_2D(-L*0.5, 0.0); g_plot_2D(L*0.5, 0.0);  //X Axis
            g_move_2D(0.0, -1.2); g_plot_2D(0.0, 1.2);      //Y Axis

            //Profile of u
            g_line_width(2.0);
            g_line_color(1.0, 0.0, 0.0, 1.0);
            for (int i=0; i<N-1; i++) {
                g_move_2D(i*dx-L*0.5, u[i]);
                g_plot_2D((i+1)*dx-L*0.5, u[i+1]);
            }

            g_text_size(18);
            g_text_color(1.0, 0.0, 0.0, 1.0);
            g_text_2D_virtual(L*0.5 - 0.4, 1.0, "u: -");
        }
        ////////////// Charactor Part //////////////
        {
            g_sel_scale(1);                                         //Select Virtual scale
            g_text_size(16);
            g_text_color(0, 0, 0, 1);

            double TextLeft = 620.0;
            double TextTop = 40.0;
            double TextWidth = 30.0;
            int IncrimentTextWidth = 0;

            g_text_standard(TextLeft, TextTop + TextWidth*(IncrimentTextWidth ++), "t = %2.15f", t);
            g_text_standard(TextLeft, TextTop + TextWidth*(IncrimentTextWidth ++), "i_time = %d", i_time);
            g_text_standard(TextLeft, TextTop + TextWidth*(IncrimentTextWidth ++), "dt = %2.8f", dt);
            g_text_standard(TextLeft, TextTop + TextWidth*(IncrimentTextWidth ++), "L = %2.4f", L);
            g_text_standard(TextLeft, TextTop + TextWidth*(IncrimentTextWidth ++), "N = %d", N);
            g_text_standard(TextLeft, TextTop + TextWidth*(IncrimentTextWidth ++), "dx = %2.4f", dx);
            g_text_standard(TextLeft, TextTop + TextWidth*(IncrimentTextWidth ++), "D = %2.4f", D);
            g_text_standard(TextLeft, TextTop + TextWidth*(IncrimentTextWidth ++), "D * dt/(dx*dx) = %2.4f (< 0.5)", D*dt/(dx*dx));
        }
        g_finish();                                                 //flush Draw buffer
        g_sleep(0.0);                                           //Sleep 0.0 sec
    }
    ////////////// Calculation Part //////////////
    {
    }
}
```



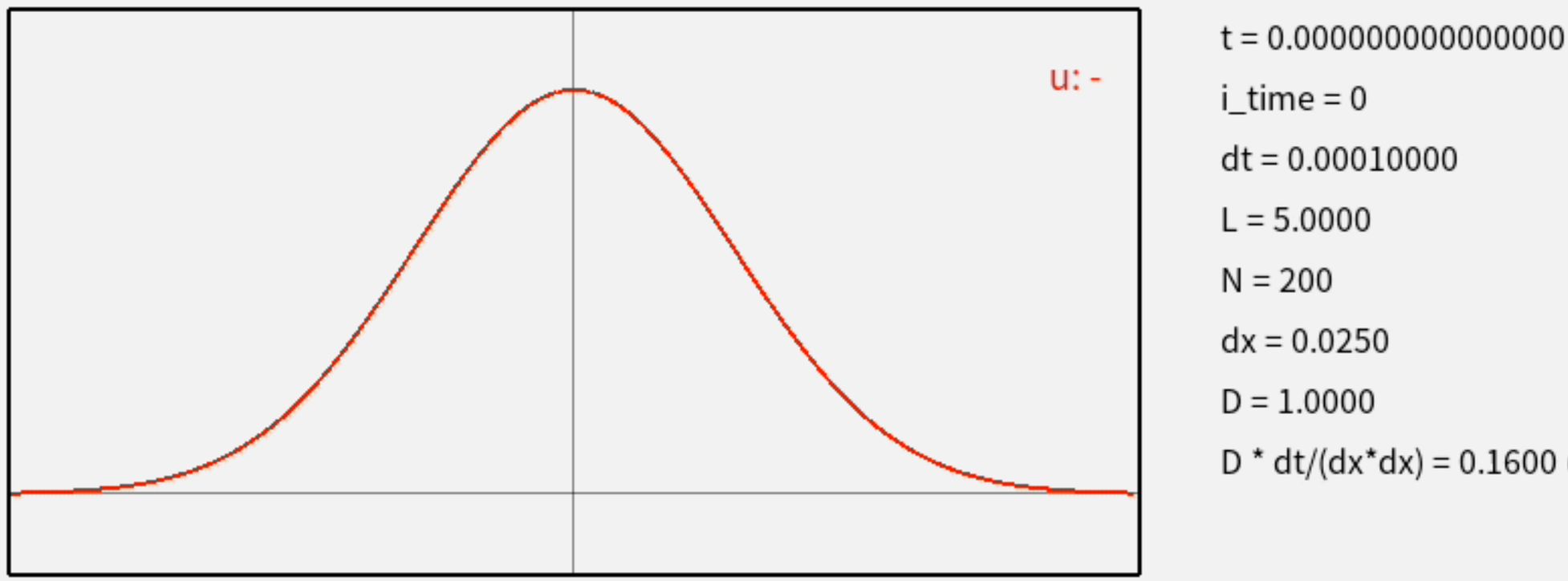
0番目はグラフのための空間

1番目は文字のための空間

10_Diffusion_1D_Text.c

2DのGLSC3D

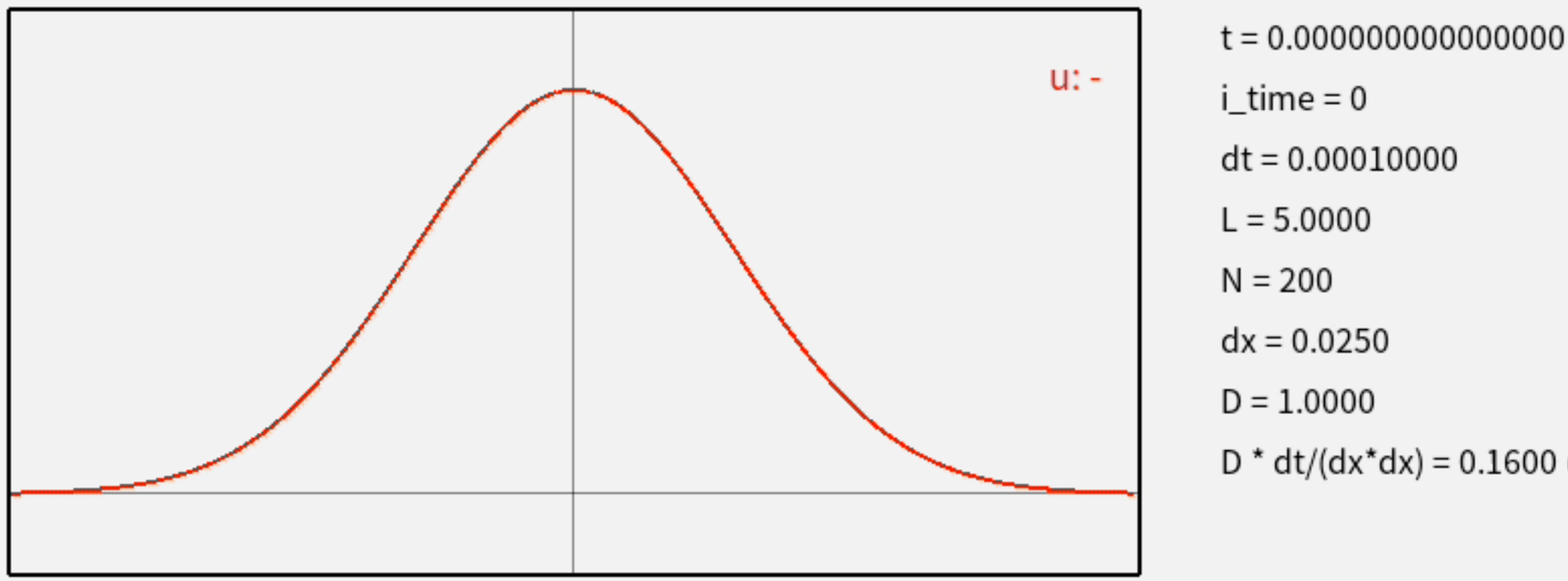
~10_Diffusion_1D_Text~



10_Diffusion_1D_Text.c

2DのGLSC3D

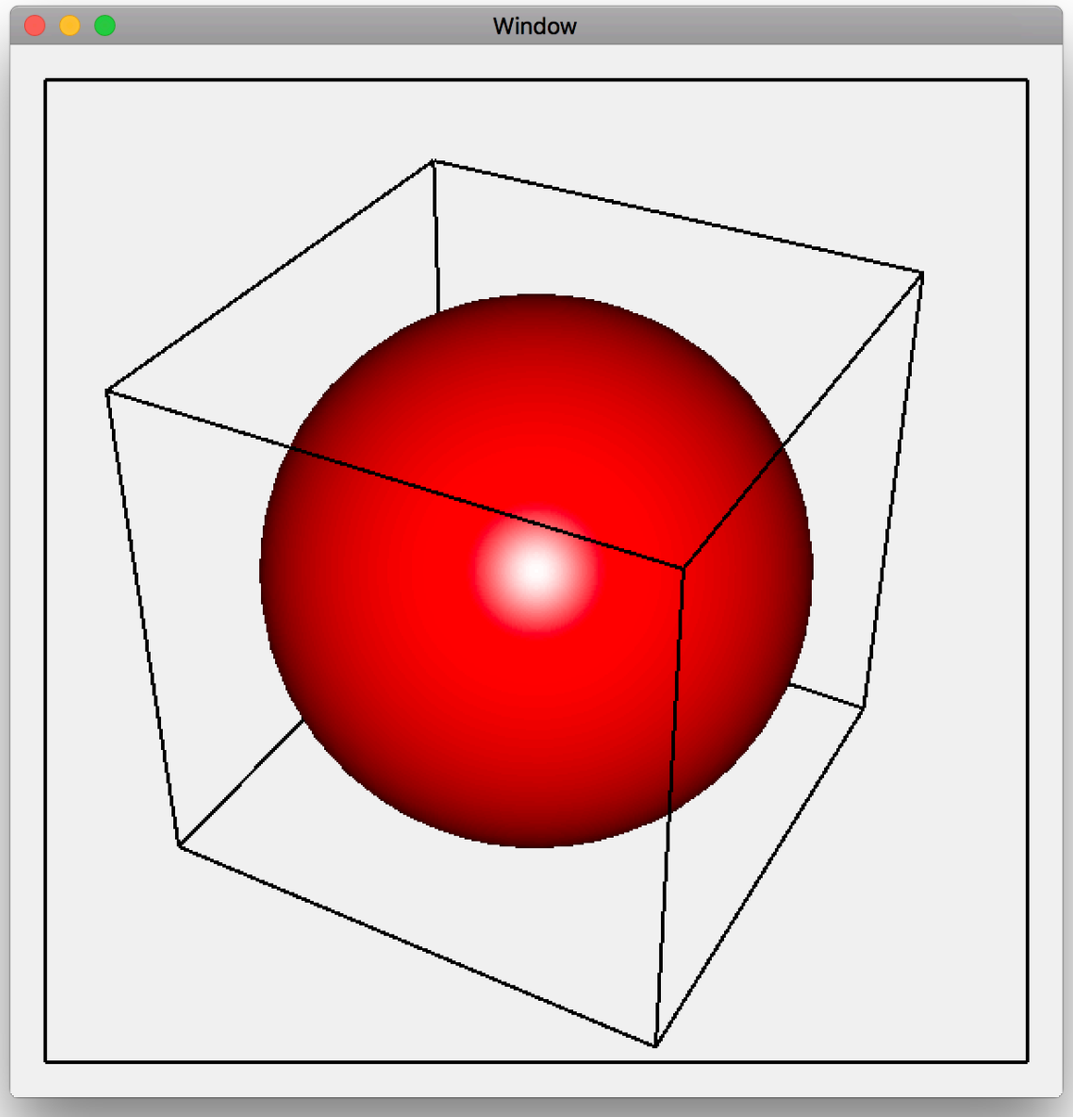
~10_Diffusion_1D_Text~



10_Diffusion_1D_Text.c

GLSC3Dの3Dの世界へ

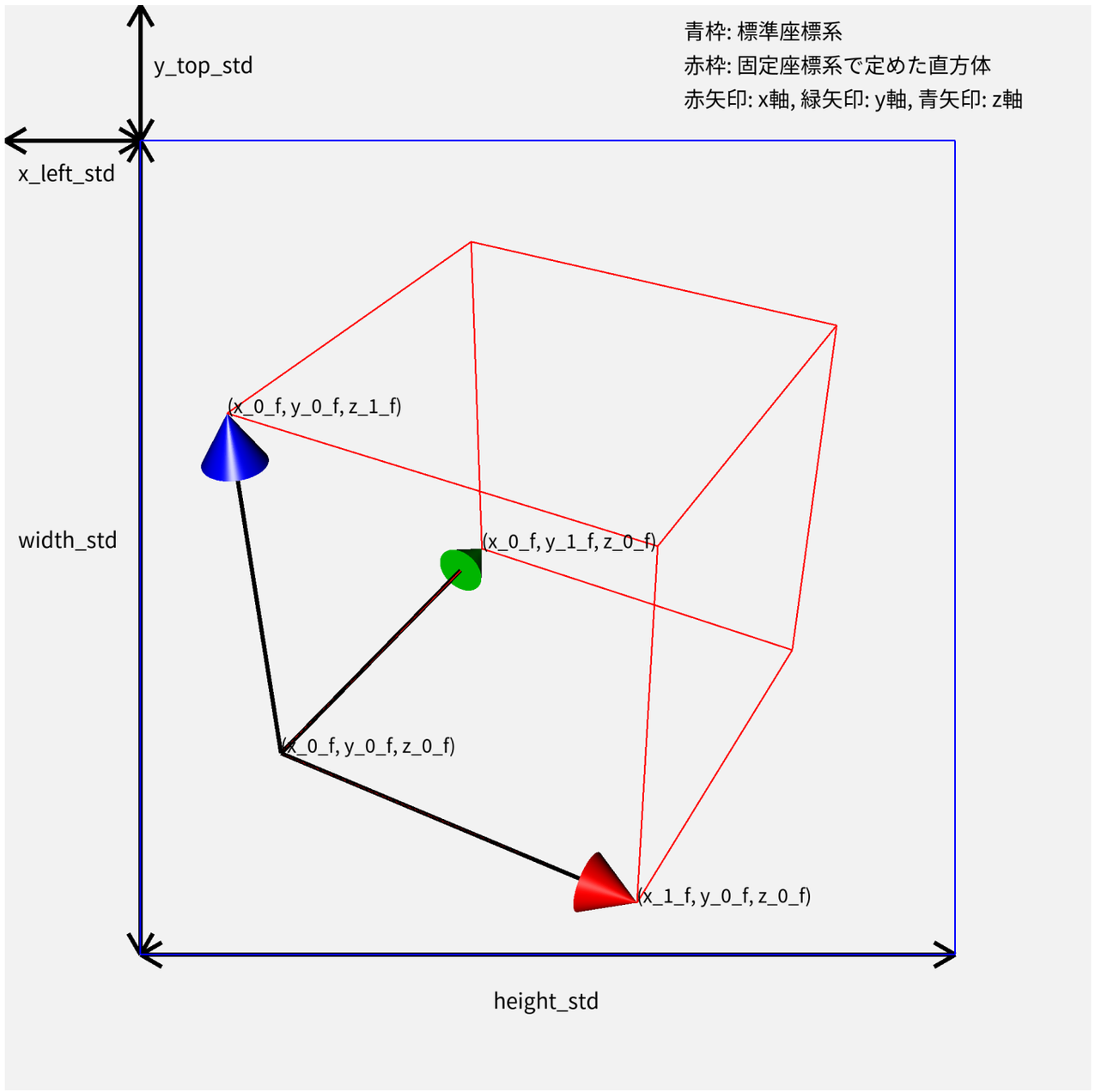
3Dの仮想座標系とは？




```
#include<stdio.h>
#include<glsc3d_3.h>
#define WX      (600)
#define WY      (600)
int main()
{
        g_init("Window", WX, WY);
        g_def_scale_3D_fix(0,
                                        -1, 1,
                                        -1, 1,
                                        -1, 1,
                                        20.0, 20.0,
                                        WX - 40.0, WY - 40.0);

        for (int i_time = 0;; i_time++)
        {
                g_cls();
                g_sel_scale(0);
                g_boundary();
                g_sphere_3D(0.0, 0.0, 0.0,
                                        1.0, 0, 1);

                g_box_3D(-1, 1, -1, 1, -1, 1, 1, 0);
                g_finish();
        }
        return 0;
}
```

12_g_def_scale_3D_fix.c

g_def_scale_3D_fix



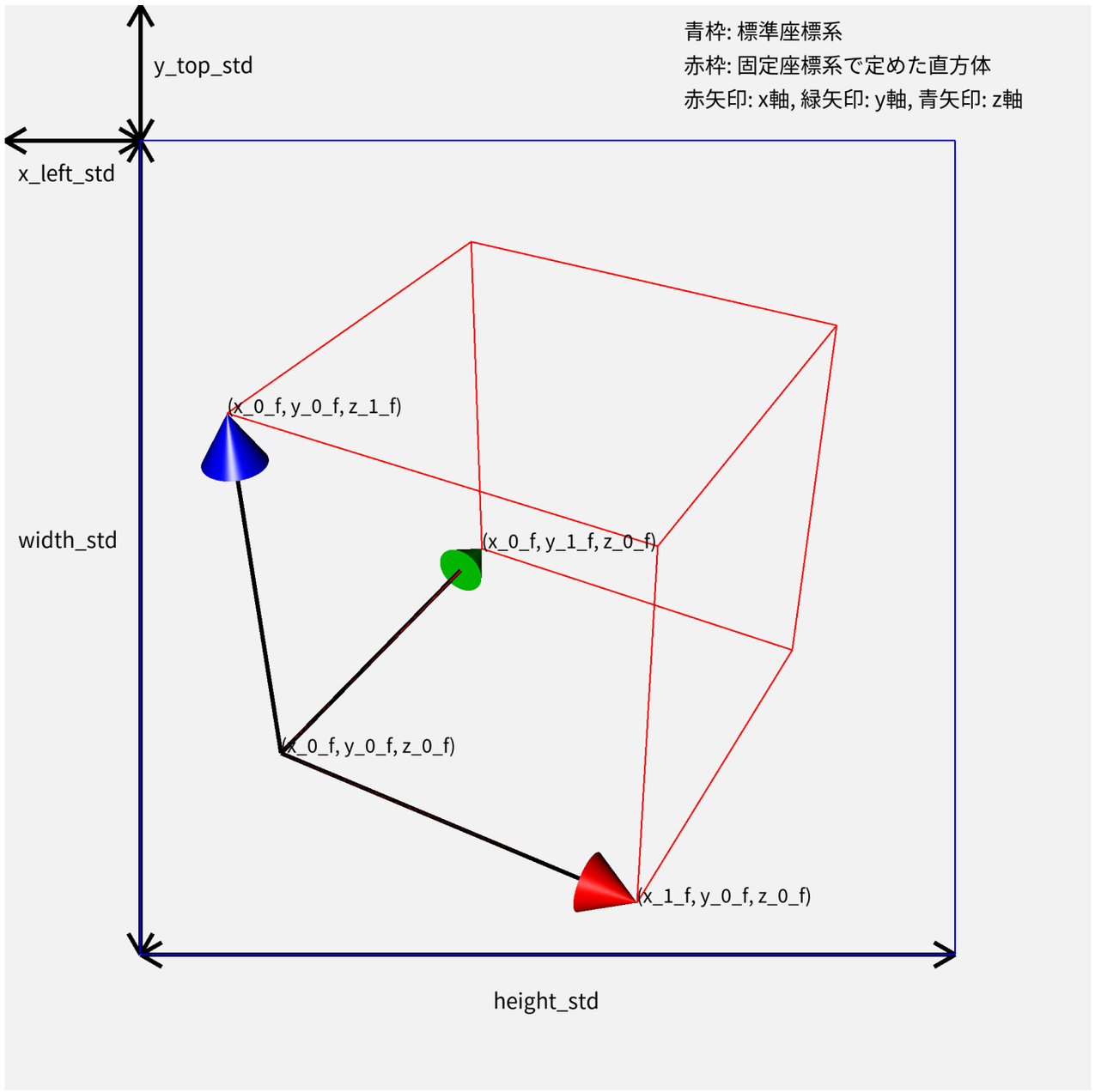

```
void g_def_scale_3D_fix(
        int id,
        double x_0_f, double x_1_f,
        double y_0_f, double y_1_f,
        double z_0_f, double z_1_f,
        double x_left_std, double y_top_std,
        double width_std, double height_std);
```

固定座標系と呼ぶ

標準座標系と呼ぶ

note:g_def_scale_3D_fixでは,
「右手系」の座標しか定義
できないことに注意.

g_def_scale_3D_fix



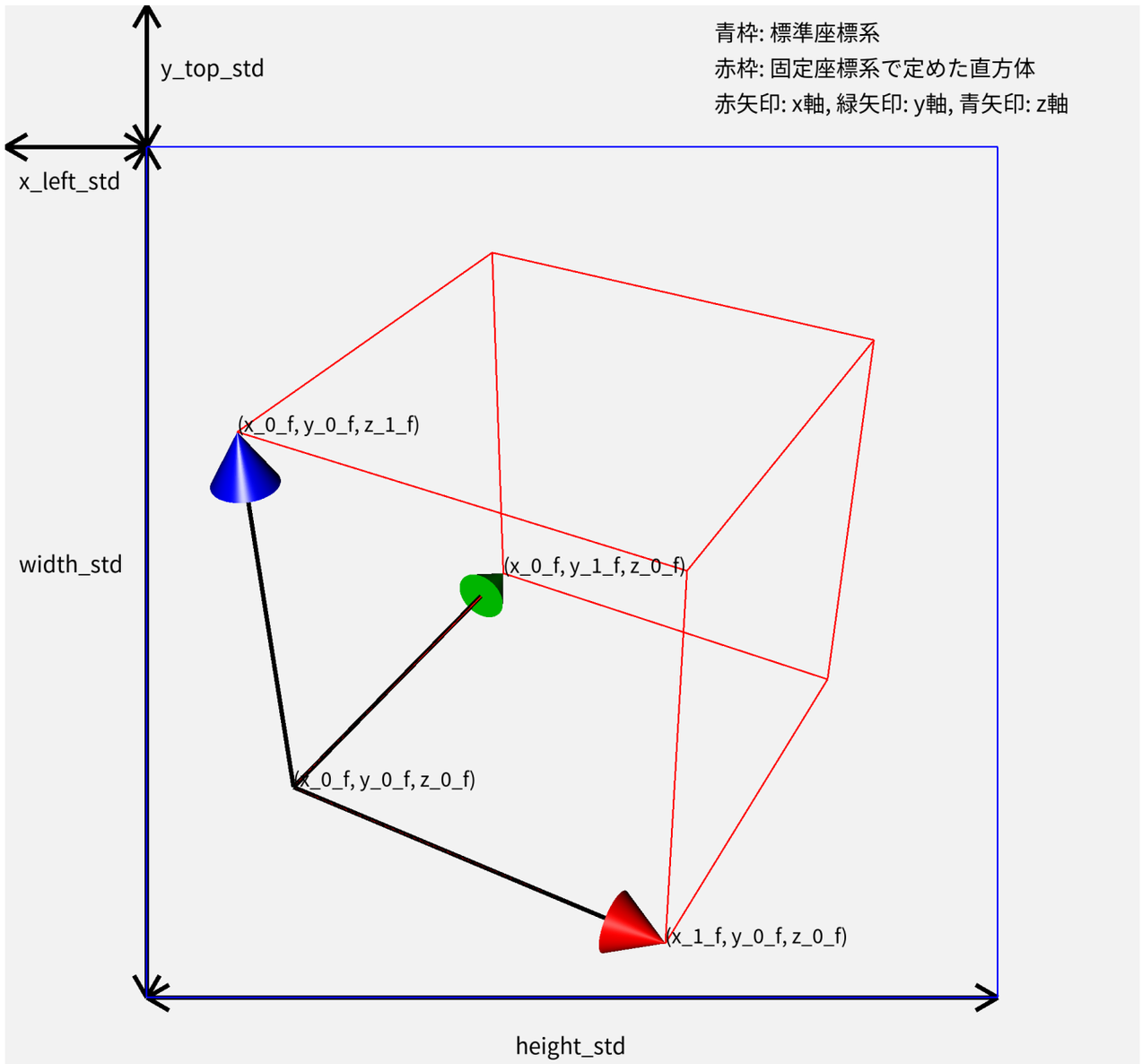

```
void g_def_scale_3D_fix(
        int id,
        double x_0_f, double x_1_f,
        double y_0_f, double y_1_f,
        double z_0_f, double z_1_f,
        double x_left_std, double y_top_std,
        double width_std, double height_std);
```

固定座標系と呼ぶ

標準座標系と呼ぶ

note:g_def_scale_3D_fixでは,
「右手系」の座標しか定義
できないことに注意.

g_def_scale_3D_fix




```
void g_def_scale_3D_fix(
        int id,
        double x_0_f, double x_1_f,
        double y_0_f, double y_1_f,
        double z_0_f, double z_1_f,
        double x_left_std, double y_top_std,
        double width_std, double height_std);
```

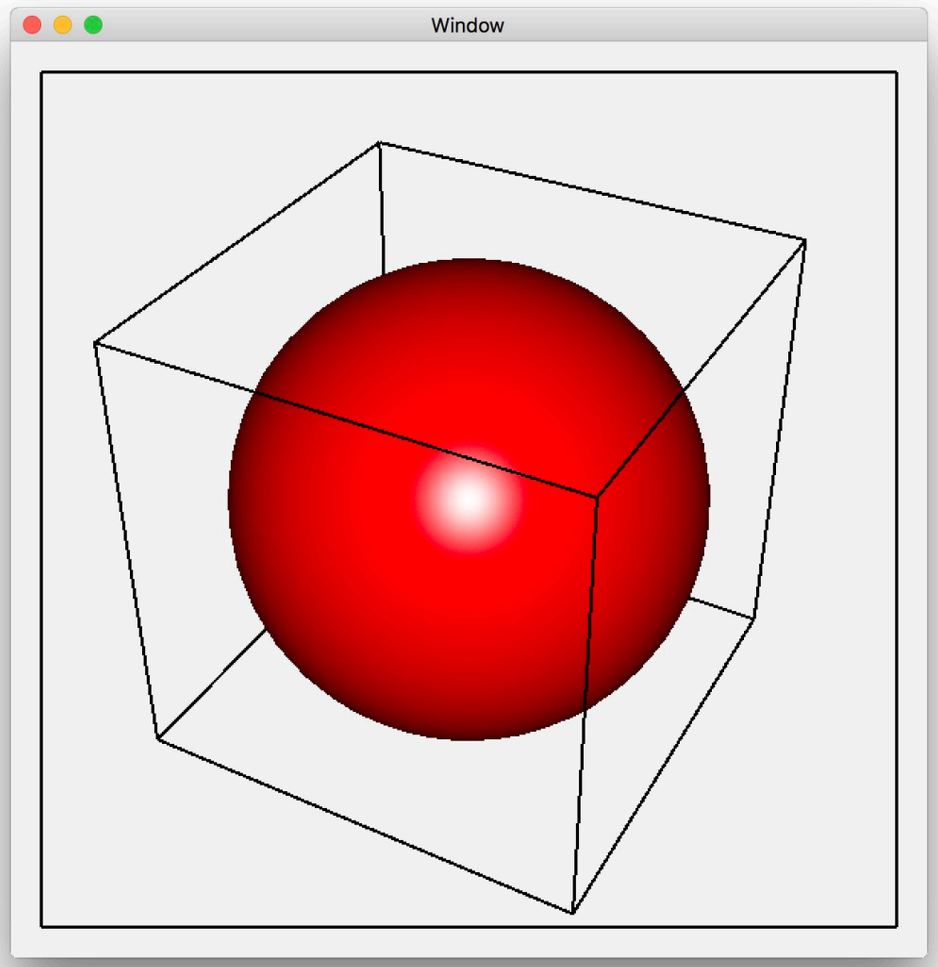
固定座標系と呼ぶ

標準座標系と呼ぶ

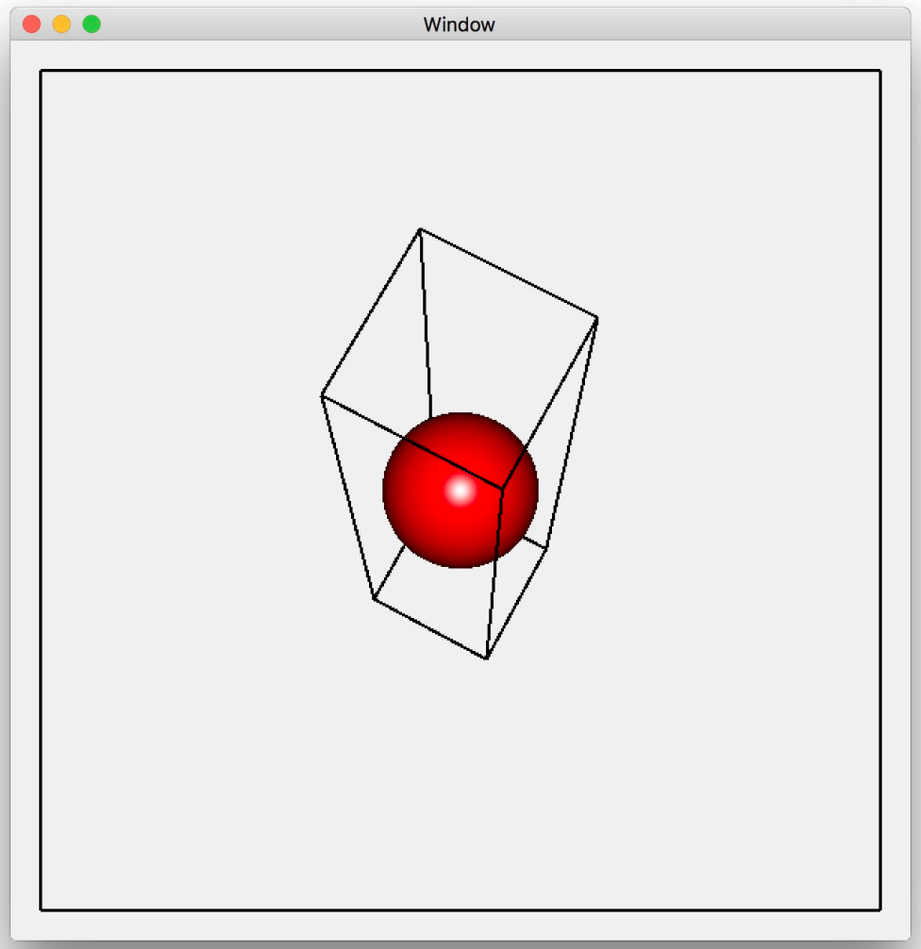
note:g_def_scale_3D_fixでは,
「右手系」の座標しか定義
できないことに注意.

g_def_scale_3D_fix

{{x_0_f, x_1_f}, {y_0_f, y_1_f}, {z_0_f, z_1_f}}



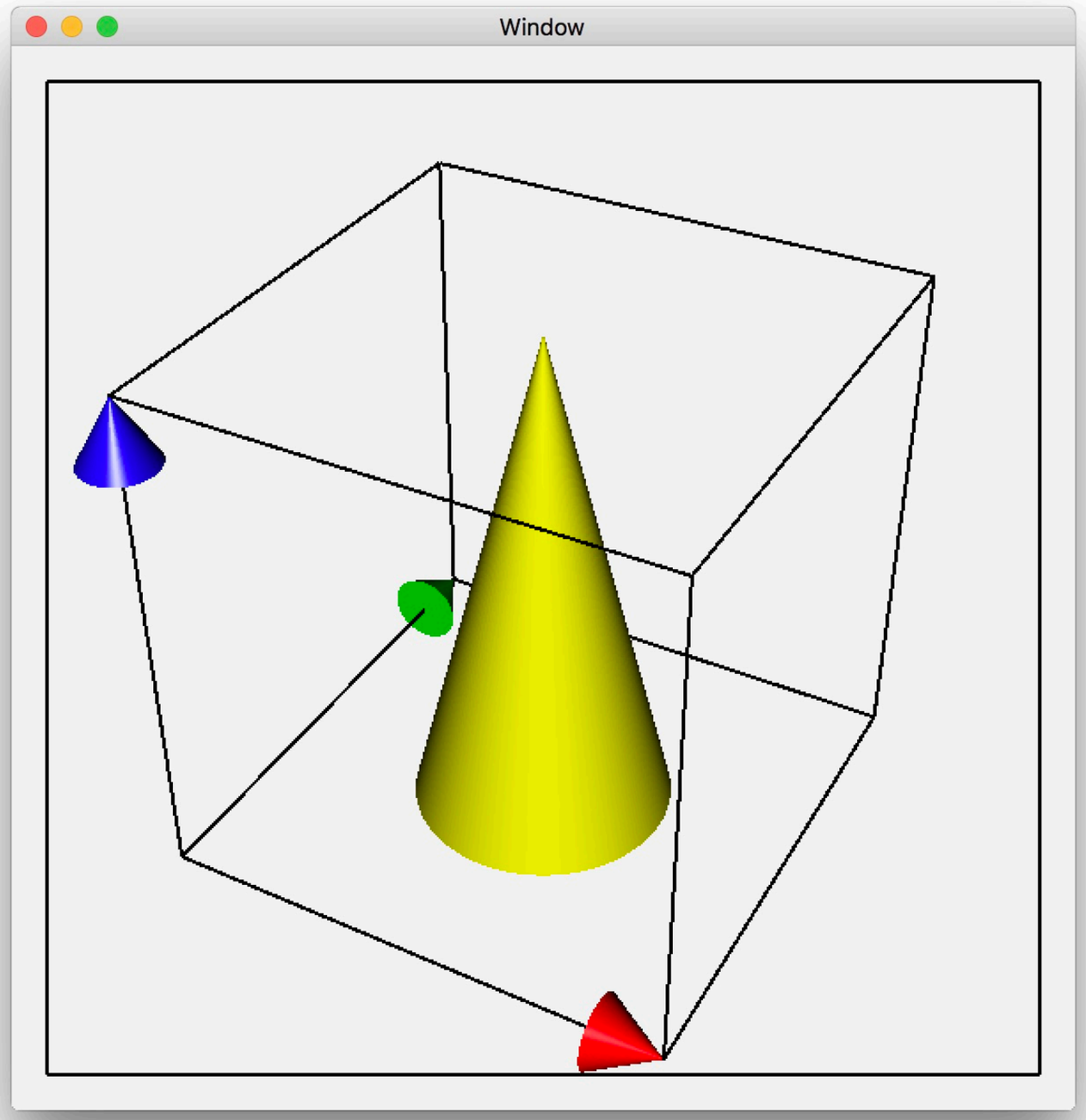
{{-1, 1}, {-1, 1}, {-1, 1}}



{{-1, 1}, {-1, 1}, {-5, 5}}

12_g_def_scale_3D_fix.c

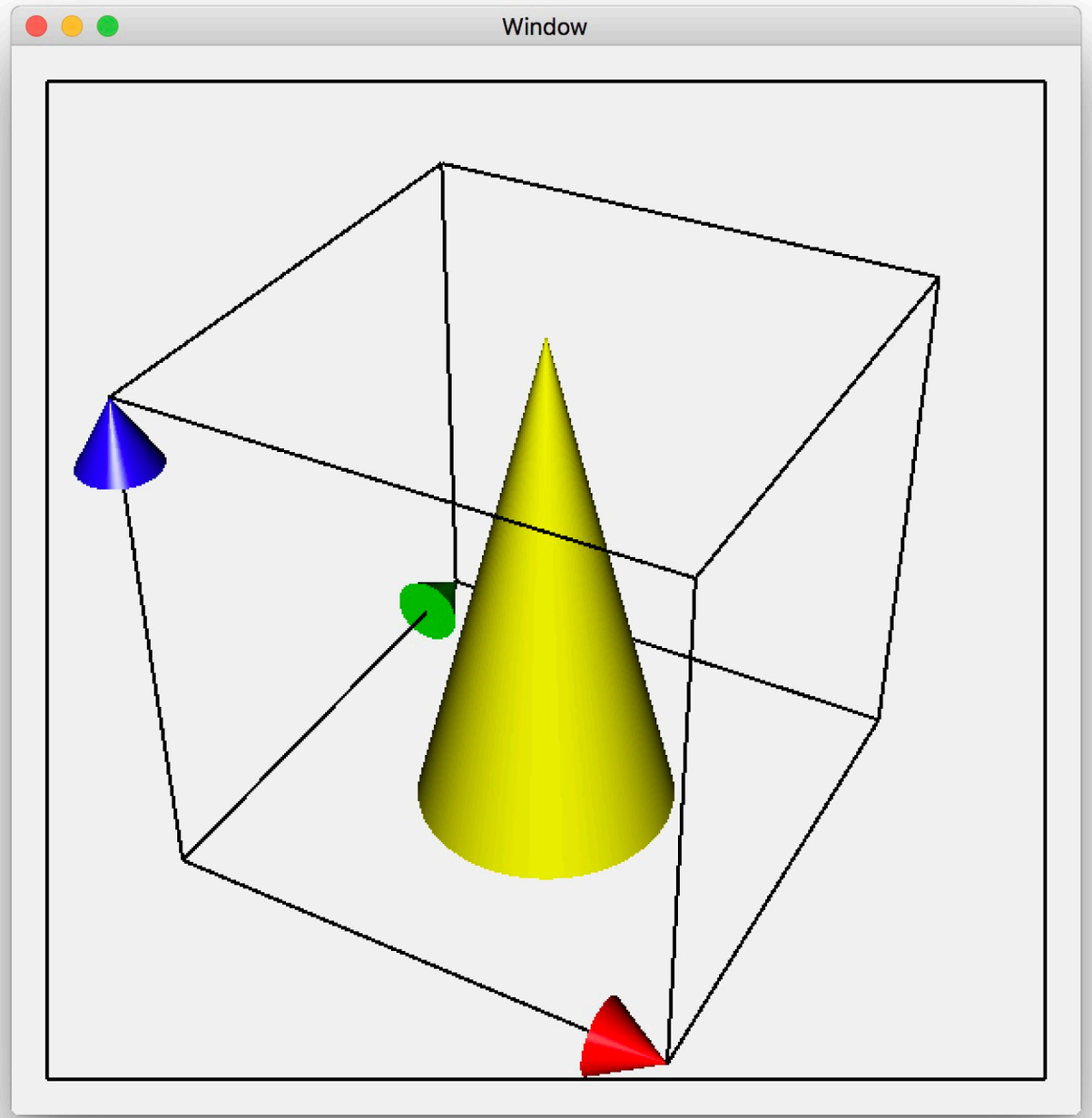
g_vision




```
#include<stdio.h>
#include<glsc3d_3.h>
#define WX     (600)
#define WY     (600)
int main()
{
        g_init("Window", WX, WY);
        g_def_scale_3D_fix(0,
                                        -1, 1,
                                        -1, 1,
                                        -1, 1,
                                        20.0, 20.0,
                                        WX - 40.0, WY - 40.0);

        for (int i_time = 0;; i_time++)
        {
                g_cls();
                g_sel_scale(0);
                g_boundary();
                g_box_3D(-1, 1, -1, 1, -1, 1, 1, 0);
                //X-Axes
                g_area_color(1, 0, 0, 1);
                g_arrow_3D(-1, -1, -1,
                                1.0, 0.0, 0.0,
                                2.0, 0.25,
                                0, 1);
                //Y-Axes
                g_area_color(0, 1, 0, 1);
                g_arrow_3D(-1, -1, -1,
                                0.0, 1.0, 0.0,
                                2.0, 0.25,
                                0, 1);
                //Z-Axes
                g_area_color(0, 0, 1, 1);
                g_arrow_3D(-1, -1, -1,
                                0.0, 0.0, 1.0,
                                2.0, 0.25,
                                0, 1);

                g_area_color(1, 1, 0, 1);
                g_cone_3D(0, 0, -1,
                                0, 0, 1,
                                0.5, 2,
                                0, 1);
                g_finish();
        }
        return 0;
}
```

13_g_vision.c

g_vision



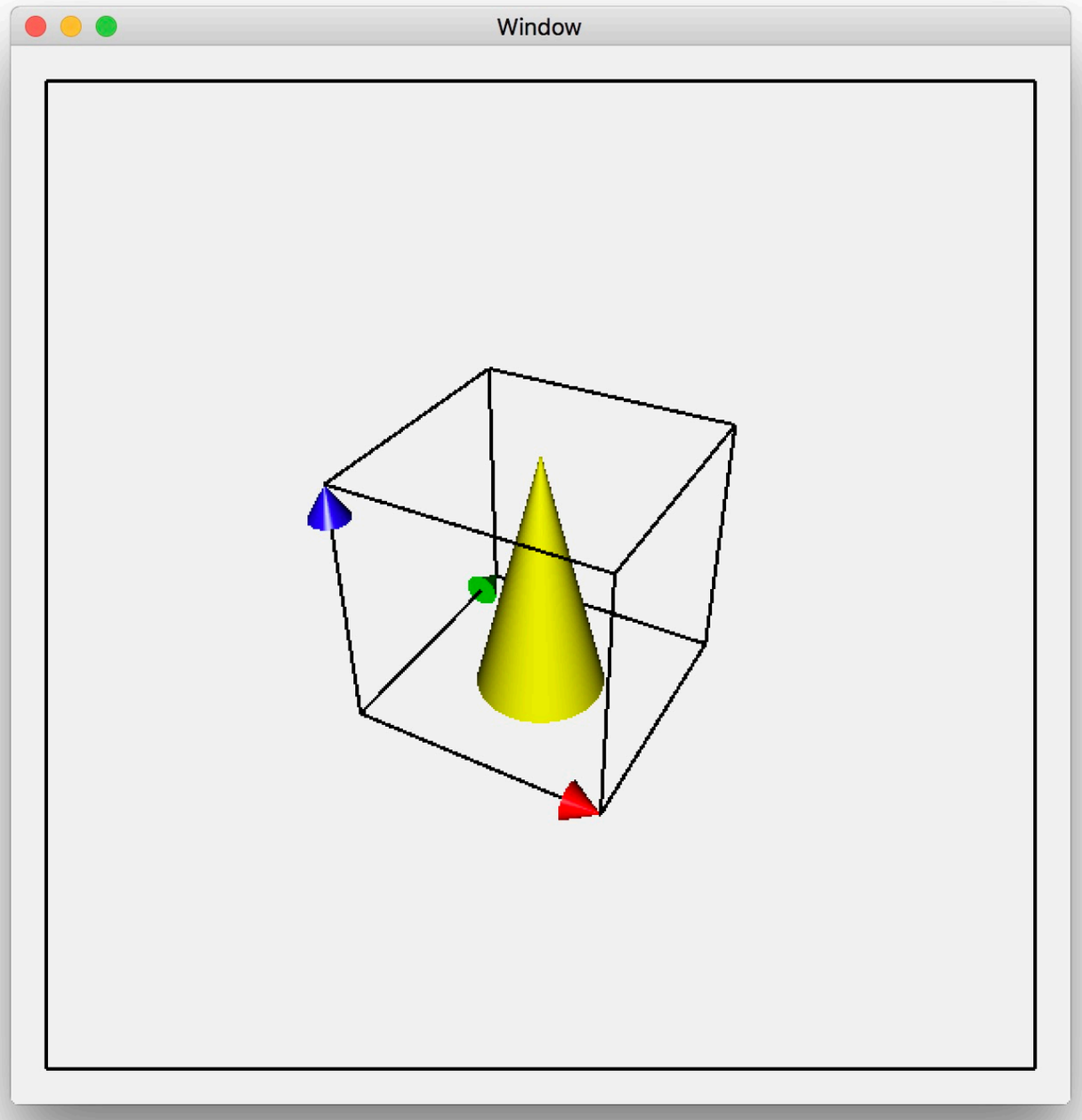

```
void g_vision(
        int id,
        double eye_x, double eye_y, double eye_z,
        double up_x, double up_y, double up_z,
        double zoom
);
```

```
for (int i_time = 0;; i_time++)
{
    g_vision(0,
             2.6, -4.8, 4,
             0, 0, 1,
             1);
    g_cls();
    g_sel_scale(0);
    g_boundary();
```

見た目は変わらないはず

13_g_vision.c

g_vision



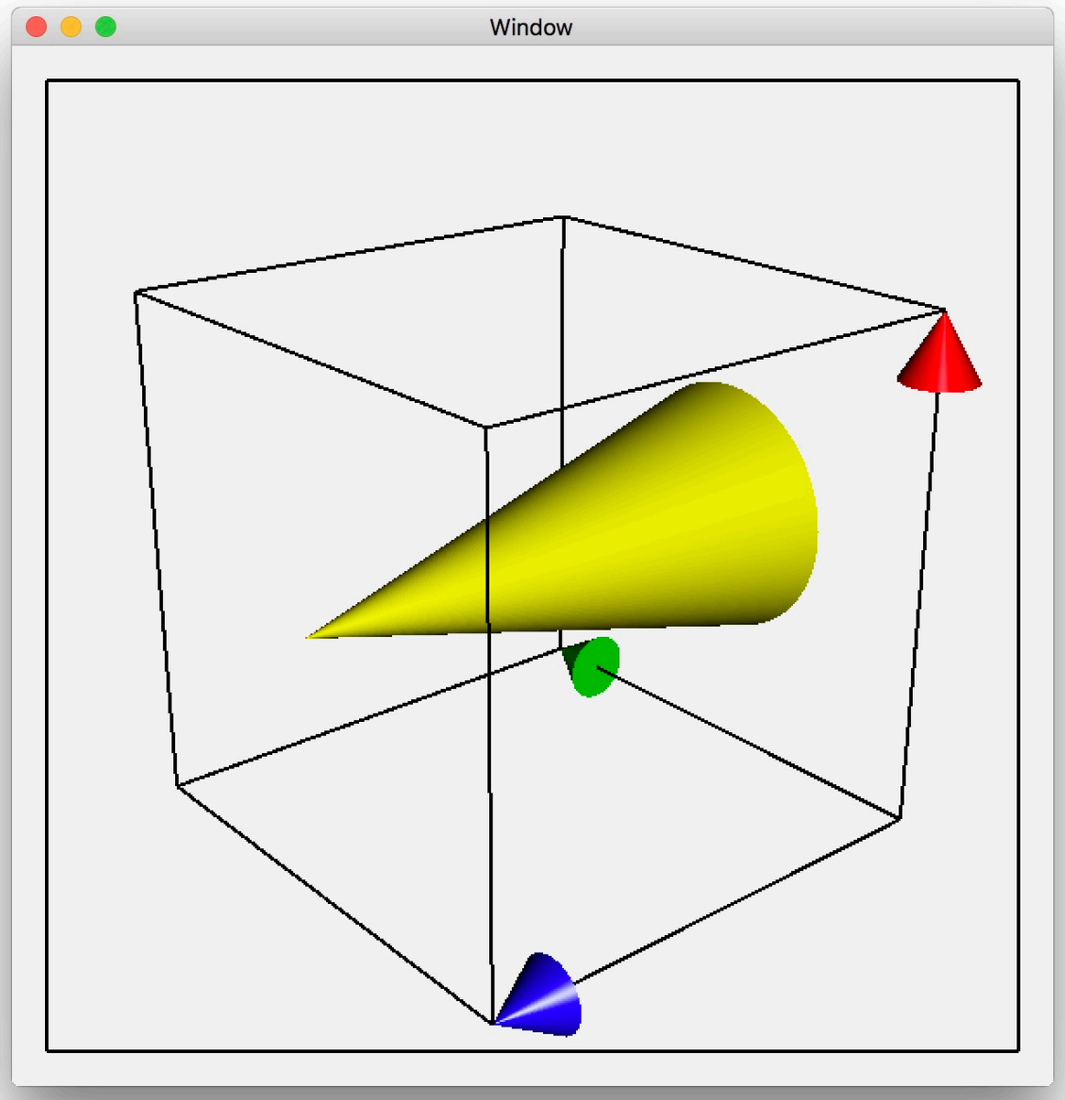

```
void g_vision(
    int id,
    double eye_x, double eye_y, double eye_z,
    double up_x, double up_y, double up_z,
    double zoom
);
```

```
for (int i_time = 0;; i_time++)
{
    g_vision(0,
             2.6, -4.8, 4,
             0, 0, 1,
             0.5);
    g_cls();
    g_sel_scale(0);
    g_boundary();
```

カメラのzoomが「0.5」になったので小さくなった.

13_g_vision.c

g_vision




```
void g_vision(
        int id,
        double eye_x, double eye_y, double eye_z,
        double up_x, double up_y, double up_z,
        double zoom
);
```

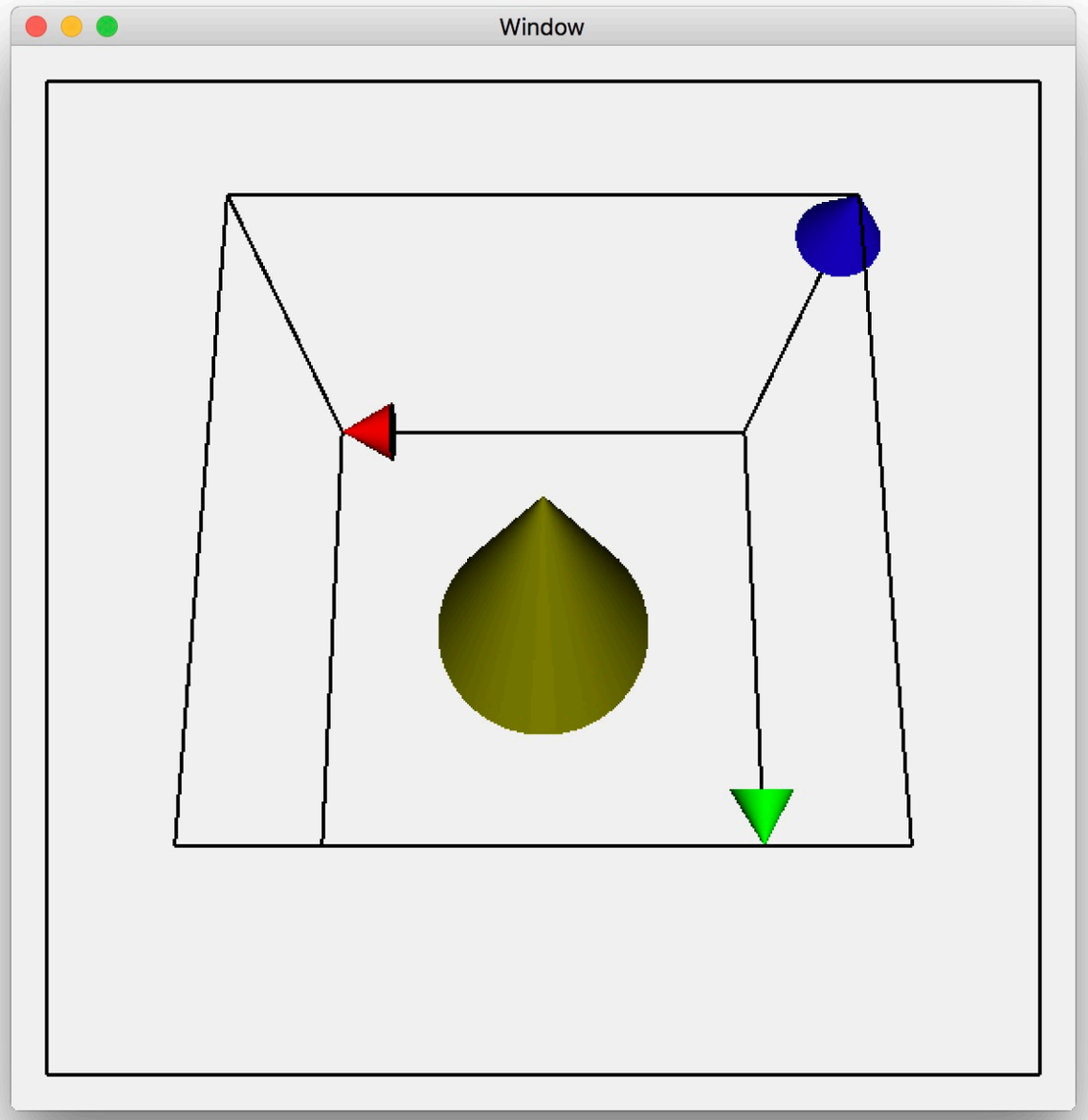
```
for (int i_time = 0;; i_time++)
{
        g_vision(0,
                        2.6, -4.8, 4,
                        1, 0, 0,
                        1);
    g_cls();
    g_sel_scale(0);
    g_boundary();
```

カメラの上方向が(1,0,0)になったので
X軸が上をむいた.

※デフォルトでは(0,0,1)が上方向として設定されている.

13_g_vision.c

g_vision




```
void g_vision(
        int id,
        double eye_x, double eye_y, double eye_z,
        double up_x, double up_y, double up_z,
        double zoom
);
```

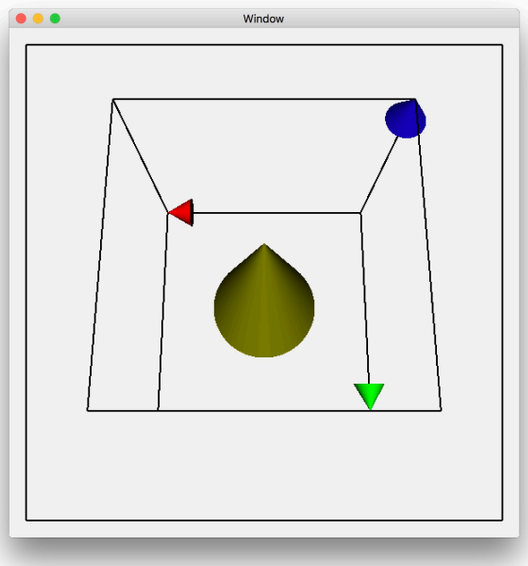
```
  for (int i_time = 0;; i_time++)
  {
      double a = 1.0;
      g_vision(0,
                  0, a, 4,
                  0, 0, 1,
                  1);
      g_cls();
      g_sel_scale(0);
      g_boundary();
```

カメラの上方向=(0,0,1)

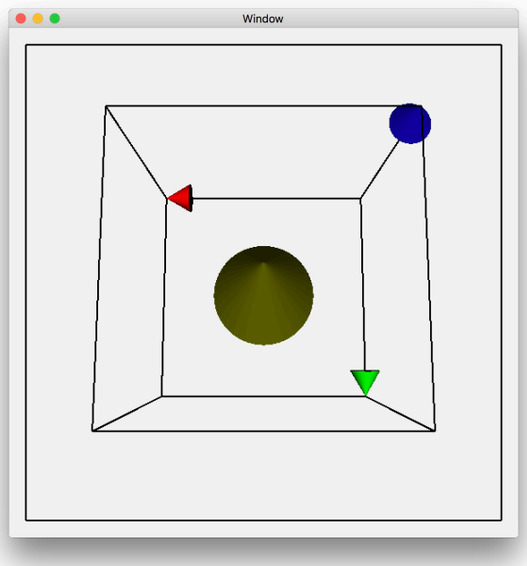
目の位置=(0,a,4)

13_g_vision.c

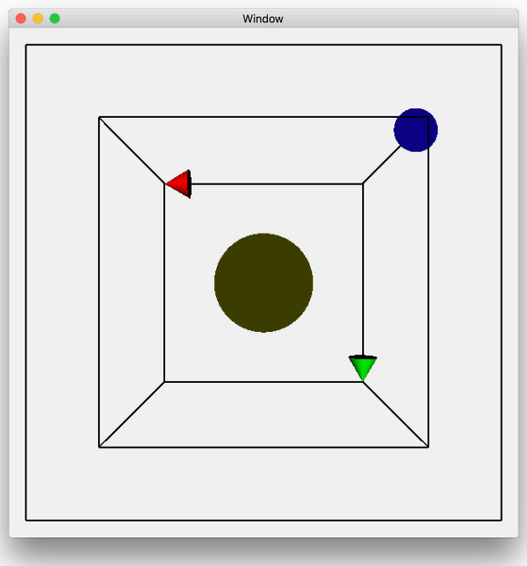
g_vision



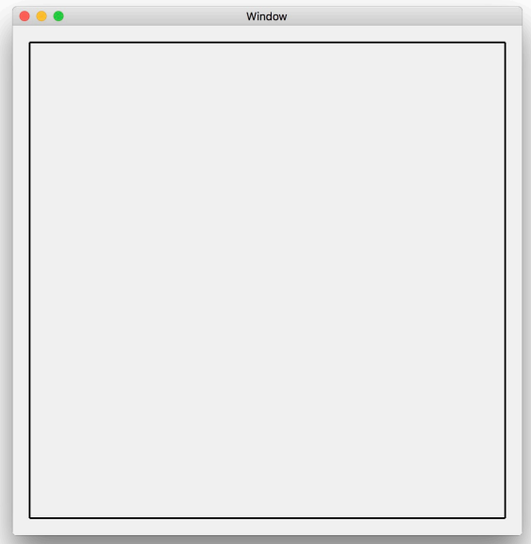
a=1;



a=0.5;

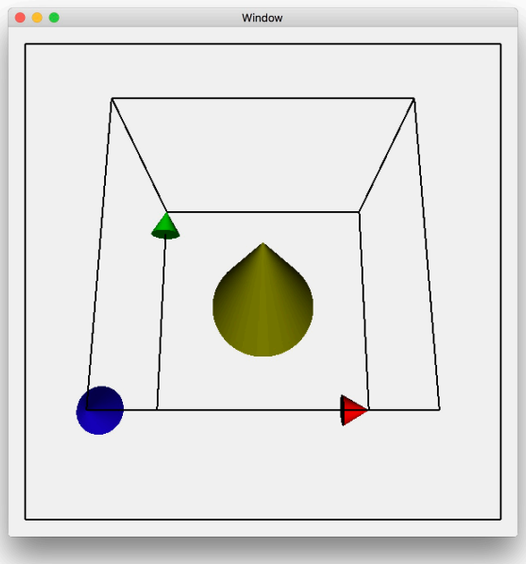


a=0.01;

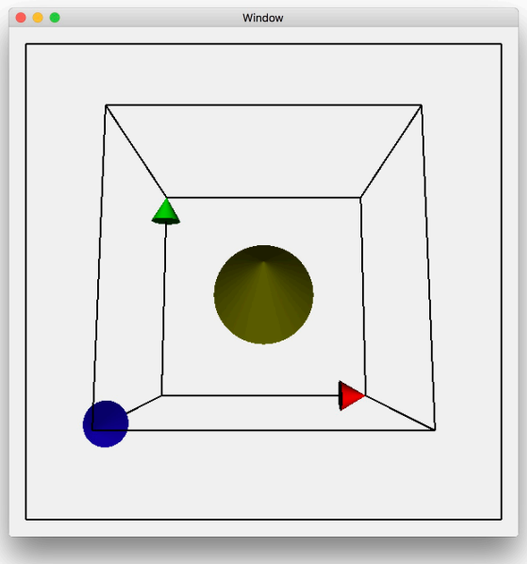


a=0;

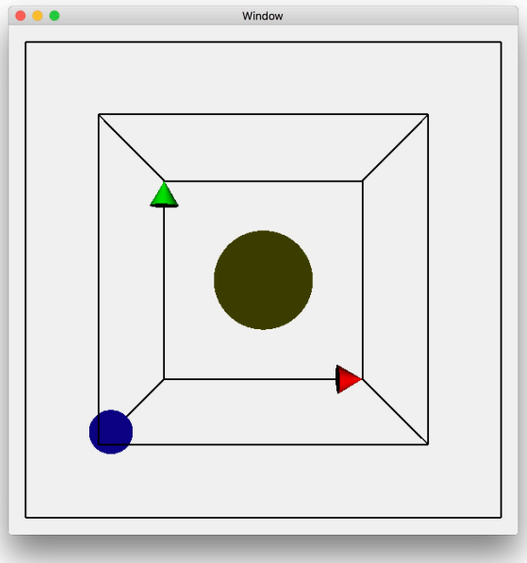
カメラの上方向=(0,0,1)，目の位置=(0,a,4)



a=-1;



a=-0.5;



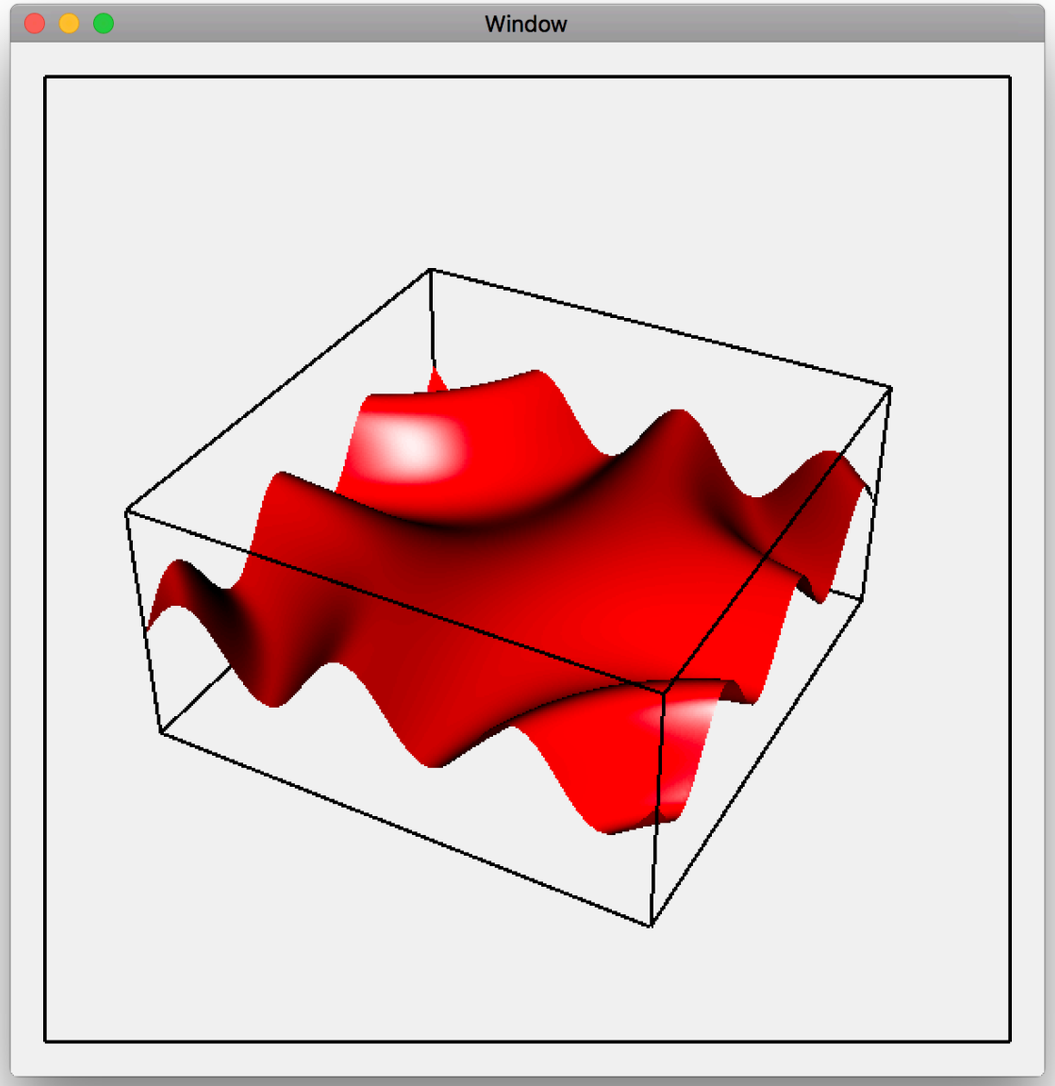
a=-0.01;

カメラの上方向≠目の位置

13_g_vision.c

g_def_scale_3D_fixでグラフ？？

z=f(x,y)を描きたい.



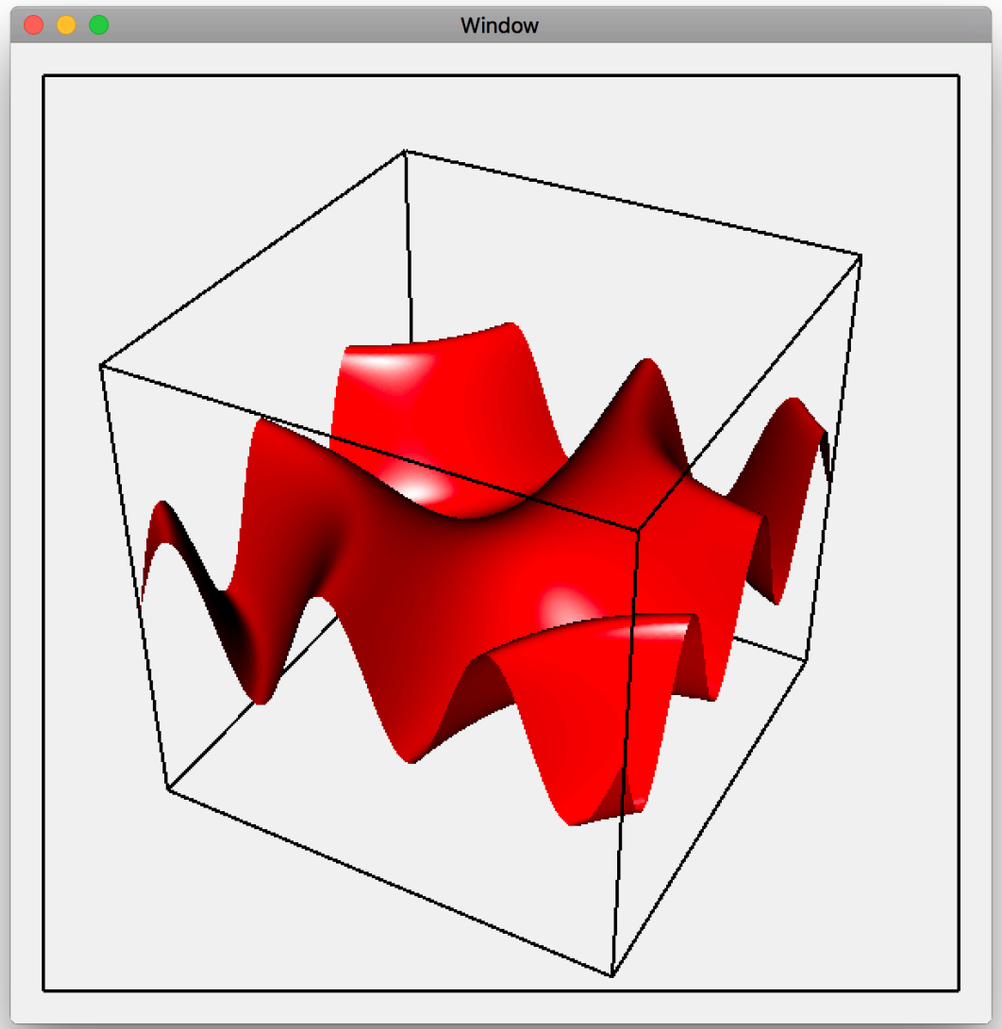
ex. f(x,y) = sin(x y)

14_g_def_scale_3D_1.c

```
#include<stdio.h>
#include<glsc3d_3.h>
#define WX     (600)
#define WY     (600)

#define XLEN         (2.0 * M_PI)
#define YLEN         (2.0 * M_PI)
#define ZLEN         (2.0 * M_PI)
#define Imax (100)
#define Jmax(100)

int main()
{
        g_init("Window", WX, WY);
        g_def_scale_3D_fix(0,
                                        -XLEN / 2, XLEN / 2,
                                        -YLEN / 2, YLEN / 2,
                                        -ZLEN / 2, ZLEN / 2,
                                        20.0, 20.0,
                                        WX - 40.0, WY - 40.0);
        double u[Imax][Jmax];
        double dx = XLEN / Imax, dy = YLEN / Jmax,rx,ry;

        for(int i = 0;i < Imax;i ++)
        {
                rx = (i + 0.5) * dx - XLEN / 2;
                for(int j = 0;j < Jmax;j ++)
                {
                        ry = (j + 0.5) * dy - YLEN / 2;
                        u[i][j] = sin(rx * ry) * 0.5;
                }
        }
        for (int i_time = 0;; i_time++)
        {
                g_cls();
                g_sel_scale(0);
                g_boundary();
                g_box_center_3D_core(0, 0, 0, XLEN, YLEN, ZLEN*0.5, 0, 1, 0);
                g_bird_view_3D(-XLEN / 2, XLEN / 2,
                                        -YLEN / 2, YLEN / 2,
                                        Imax, Jmax,
                                        u, 0, 1);
                g_finish();
        }
        return 0;
}
```

g_def_scale_3D_fixでグラフ？？

z=f(x,y)を描きたい.



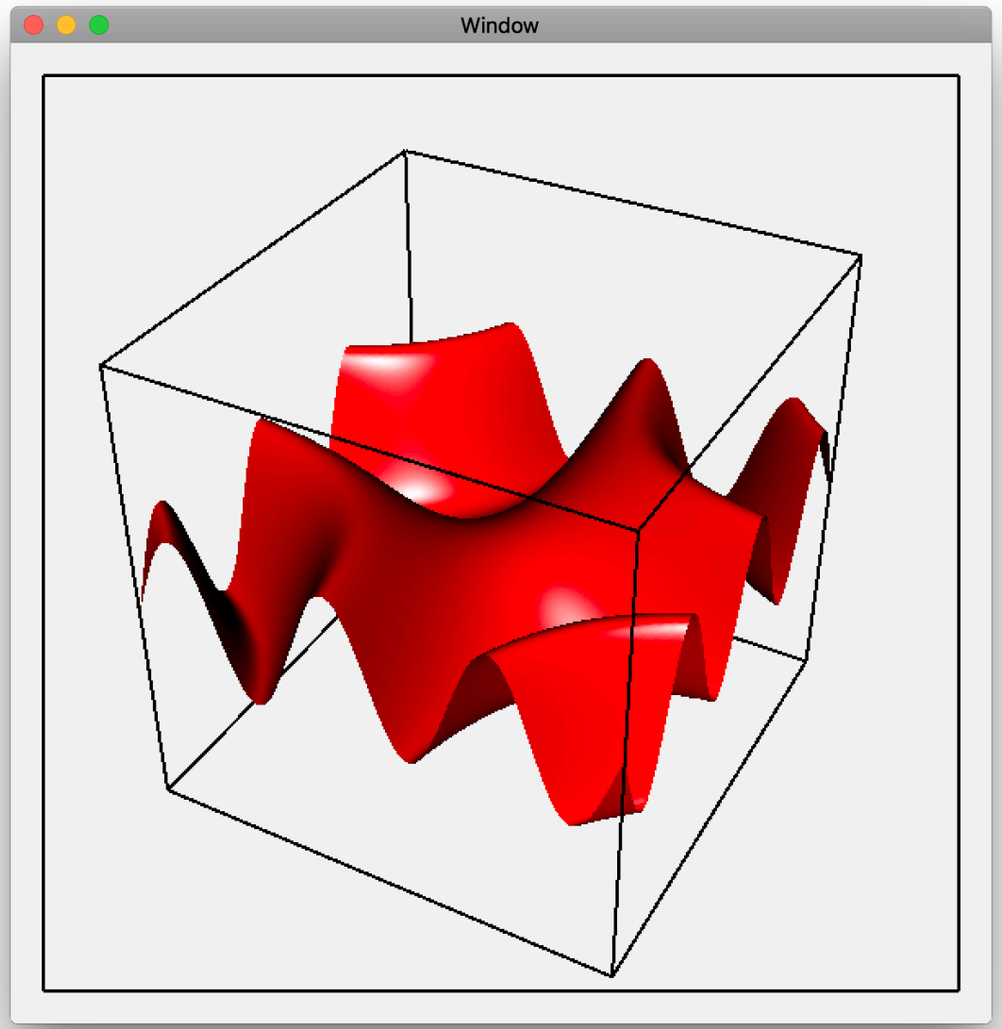
ex. f(x,y) = sin(x y)

Exercise!!

このグラフをZ軸方向に2倍引き伸ばしたものを表示せよ.

g_def_scale_3D_fixでグラフ？？

z=f(x,y)を描きたい.



ex. f(x,y) = sin(x y)

Exercise!!

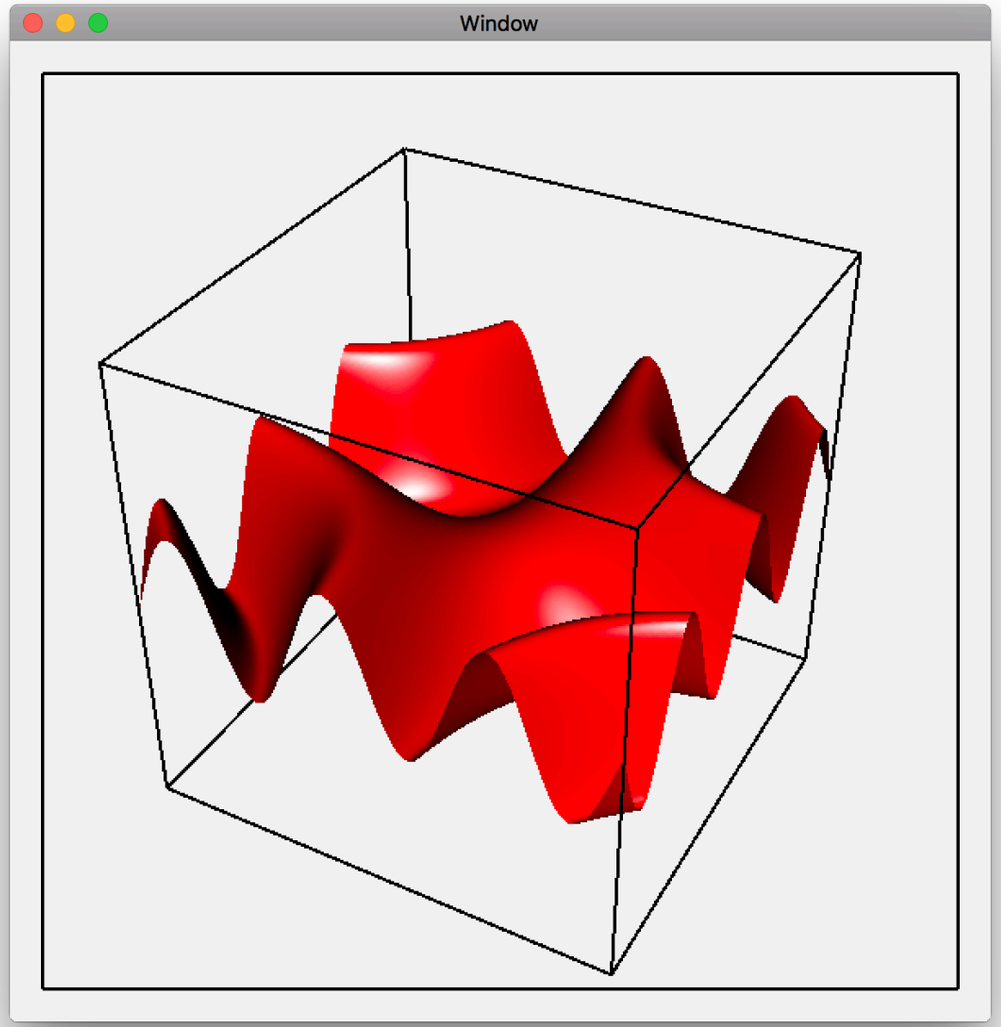
このグラフをZ軸方向に2倍引き伸ばしたものを表示せよ.

もっとたくさんのグラフがあったら？
もっとたくさんのオブジェクトがあったら？

いちいちすべてのz座標に2を掛ける
のは面倒

g_def_scale_3D_fixでグラフ？？

z=f(x,y)を描きたい.



ex. f(x,y) = sin(x y)

Exercise!!

g_def_scale_3D_fix(0,
-XLEN / 2, XLEN / 2,
-YLEN / 2, YLEN / 2,
-ZLEN / 2, ZLEN / 2,
20.0, 20.0,
WX - 40.0, WY - 40.0);

を

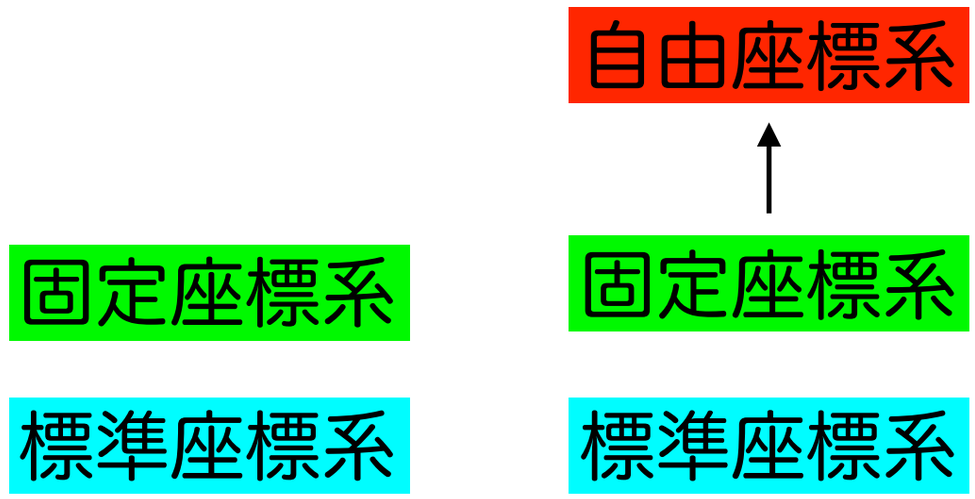
g_def_scale_3D(0,
-XLEN / 2, XLEN / 2,
-YLEN / 2, YLEN / 2,
-ZLEN / 4, ZLEN / 4,
-XLEN / 2, XLEN / 2,
-YLEN / 2, YLEN / 2,
-ZLEN / 2, ZLEN / 2,
20.0, 20.0,
WX - 40.0, WY - 40.0);

と書き換えよ.

15_g_def_scale_3D_2.c

g_def_scale_3D_fix V.S. g_def_scale_3D

```
void g_def_scale_3D_fix(int id,
double x_0_f, double x_1_f,
double y_0_f, double y_1_f,
double z_0_f, double z_1_f,
double x_left_std, double y_top_std,
double width_std, double height_std);
```



内部的には自由座標系に固定座標系を代入して
g_def_scale_3Dを Callしている.

```
void g_def_scale_3D(int id,
double x_0, double x_1,
double y_0, double y_1,
double z_0, double z_1,
double x_0_f, double x_1_f,
double y_0_f, double y_1_f,
double z_0_f, double z_1_f,
double x_left_std, double y_top_std,
double width_std, double height_std
);
```

自由座標系

固定座標系

標準座標系

note: g_def_scale_3D は3D空間を定義する関数として「最強」

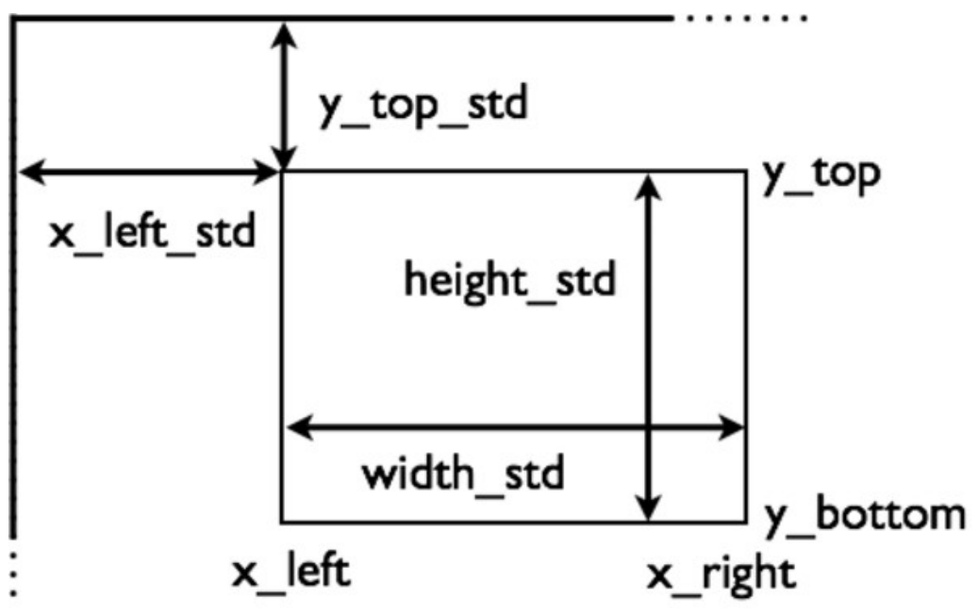
g_def_scale_3D

```
void g_def_scale_3D(int id,
double x_0, double x_1,
double y_0, double y_1,
double z_0, double z_1,
double x_0_f, double x_1_f,
double y_0_f, double y_1_f,
double z_0_f, double z_1_f,
double x_left_std, double y_top_std,
double width_std, double height_std
);
```

標準座標系

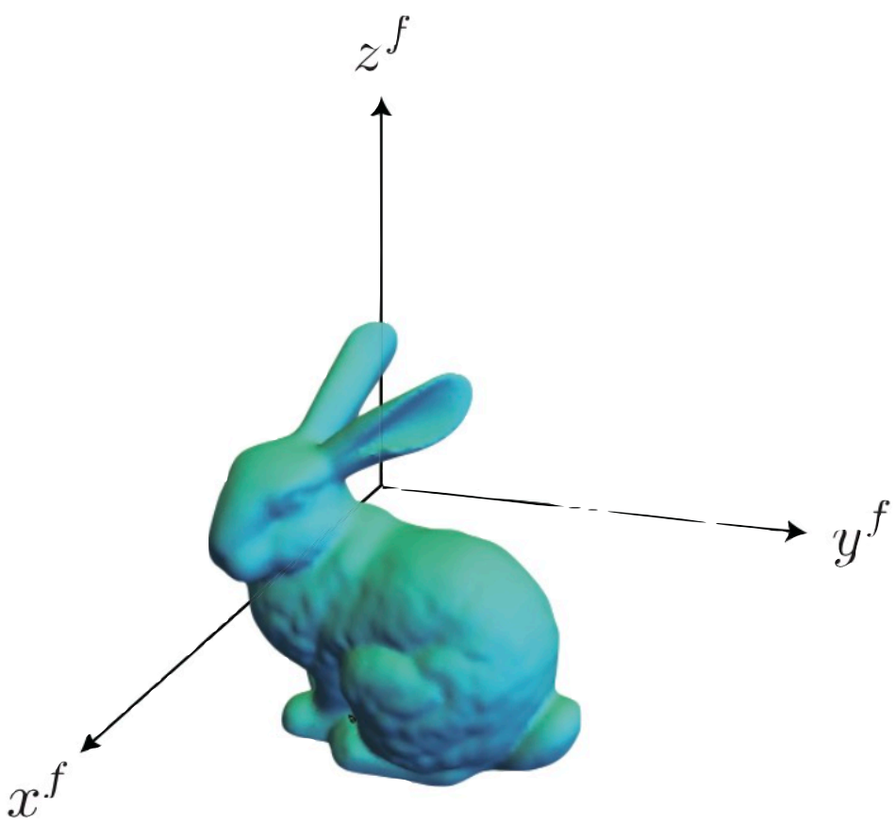
GLSC2Dでもお馴染みの概念

画面上の指定した左上の位置(x_left_std, y_top_std)に
(width_std, height_std)
の大きさの長方形領域を用意して，その中に3Dオブジェクトを描くことができる.



g_def_scale_3D

```
void g_def_scale_3D(int id,
double x_0, double x_1,
double y_0, double y_1,
double z_0, double z_1,
double x_0_f, double x_1_f,
double y_0_f, double y_1_f,
double z_0_f, double z_1_f,
double x_left_std, double y_top_std,
double width_std, double height_std
);
```



描画オブジェクト空間

固定座標系

GLSC3Dで登場した新概念

描画されるオブジェクトが置かれる3次元ユークリッド空間を描画オブジェクト空間と呼ぶ.
また空間における通常の直交座標系を固定座標系と呼ぶ.
通常とは，「(1,0,0)，(0,1,0)，(0,0,1)の３つの基本ベクトルの大きさが１であり，この順に右手系を定める」という意味

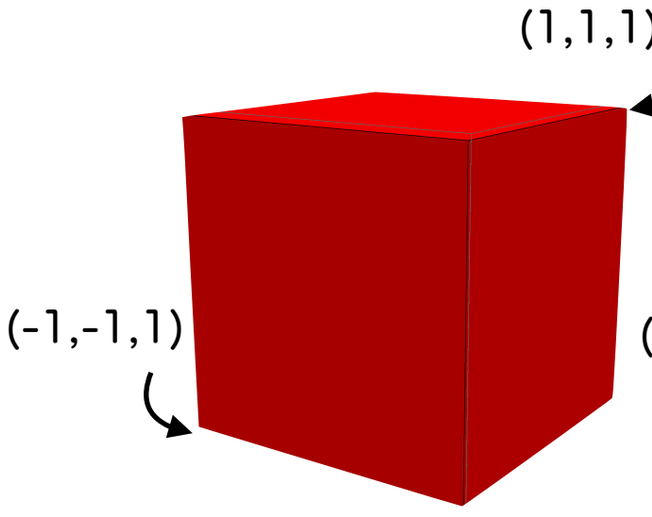
g_def_scale_3D

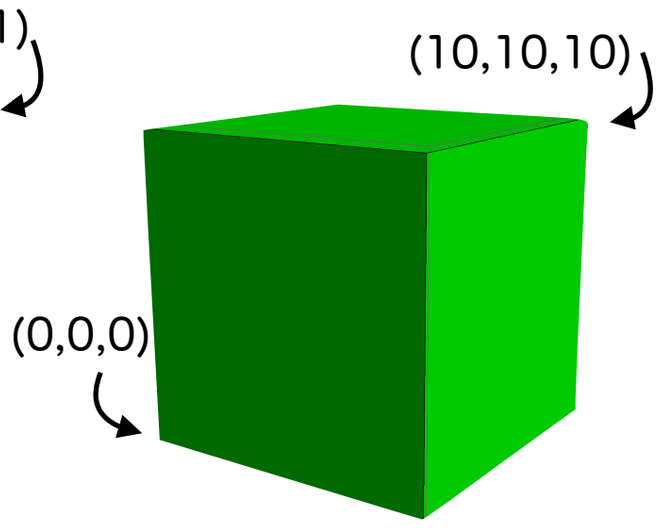
```
void g_def_scale_3D(int id,
double x_0, double x_1,
double y_0, double y_1,
double z_0, double z_1,
double x_0_f, double x_1_f,
double y_0_f, double y_1_f,
double z_0_f, double z_1_f,
double x_left_std, double y_top_std,
double width_std, double height_std
);
```

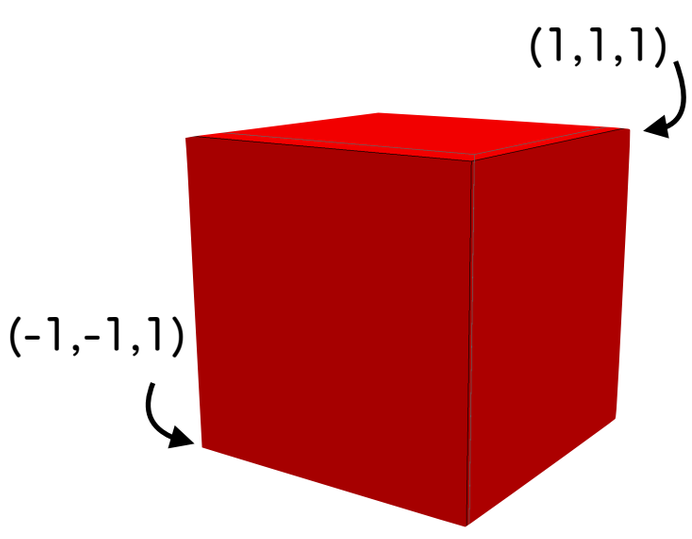
自由座標系

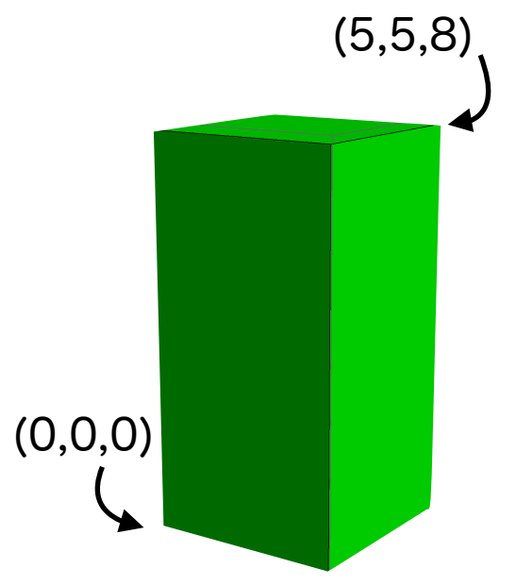
GLSC2Dでは，仮想座標系と呼ばれていたもの.

z=f(x,y)の例からわかるように，例えば，z軸に3Dオブジェクトを潰して表示させたいことがある．このようなときは，自由座標系はそのままにしておいて，固定座標系を設定する.



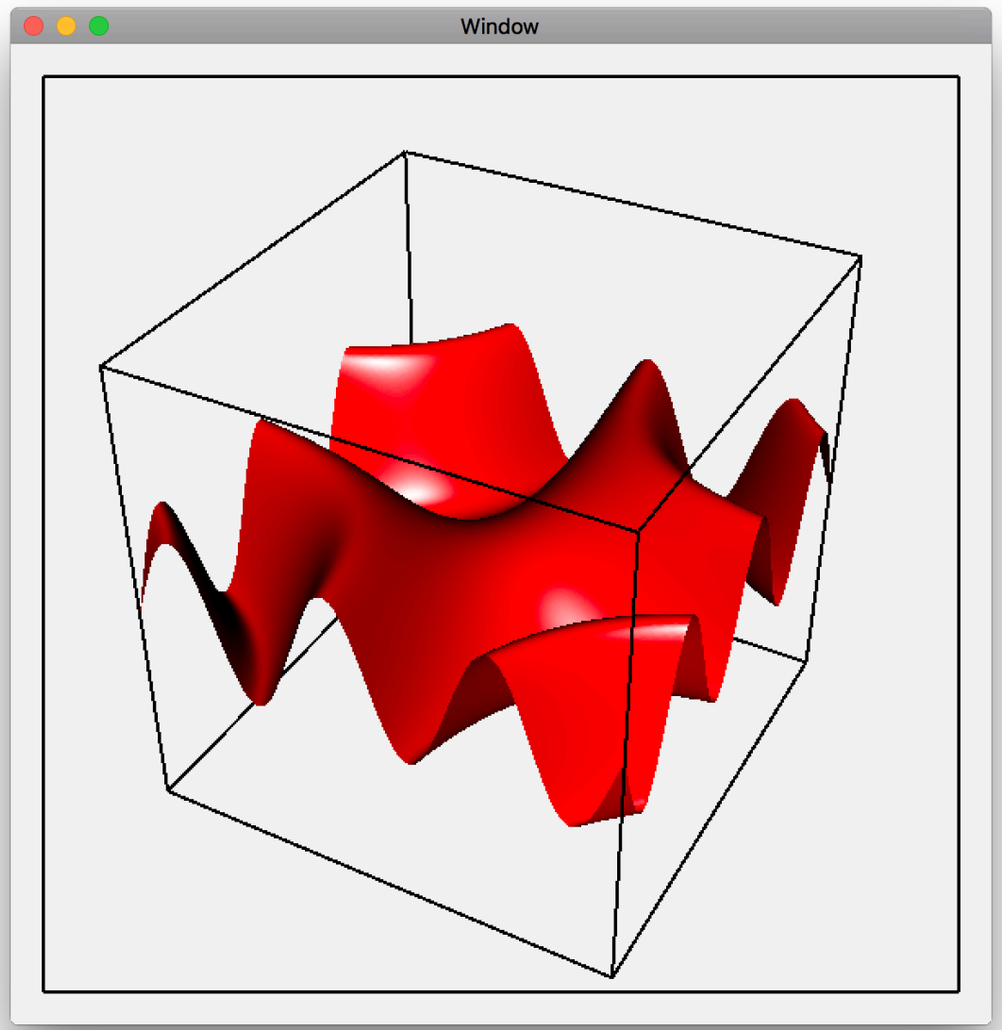






先程の例は・・・

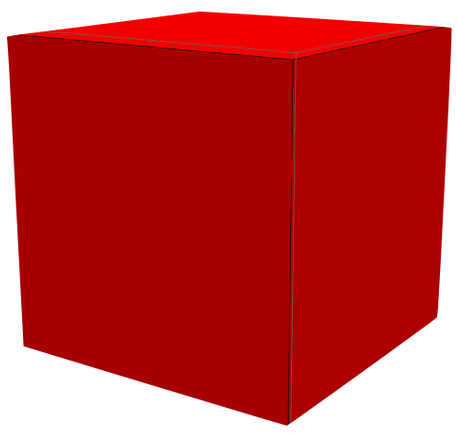
z=f(x,y)を描きたい.



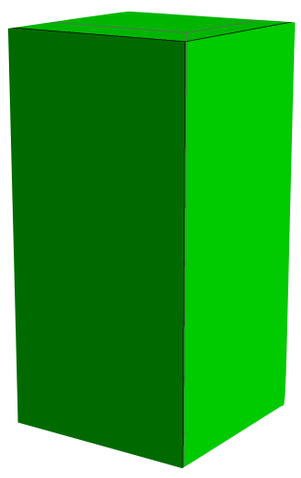
ex. f(x,y) = sin(x y)

15_g_def_scale_3D_2.c

```
g_def_scale_3D(0,
                -XLEN / 2, XLEN / 2,
                -YLEN / 2, YLEN / 2,
                -ZLEN / 4, ZLEN / 4,
                -XLEN / 2, XLEN / 2,
                -YLEN / 2, YLEN / 2,
                -ZLEN / 2, ZLEN / 2,
                20.0, 20.0,
                WX - 40.0, WY - 40.0);
```



-XLEN / 2, XLEN / 2,
-YLEN / 2, YLEN / 2,
-ZLEN / 2, ZLEN / 2



-XLEN / 2, XLEN / 2,
-YLEN / 2, YLEN / 2,
-ZLEN / 4, ZLEN / 4

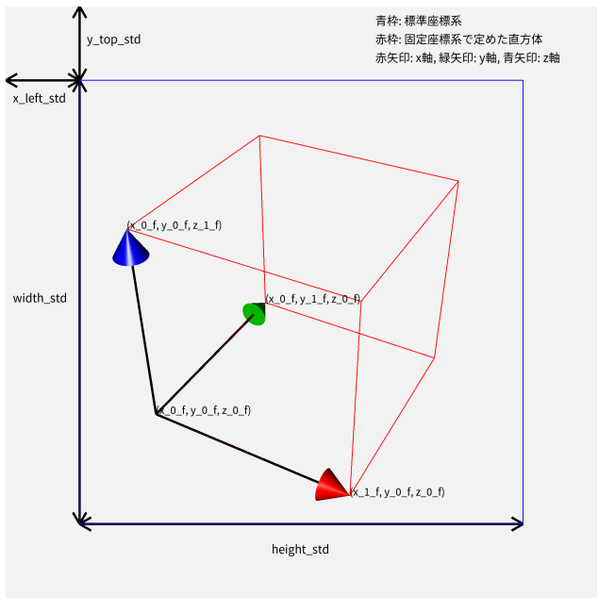
g_def_scale_3Dは最強

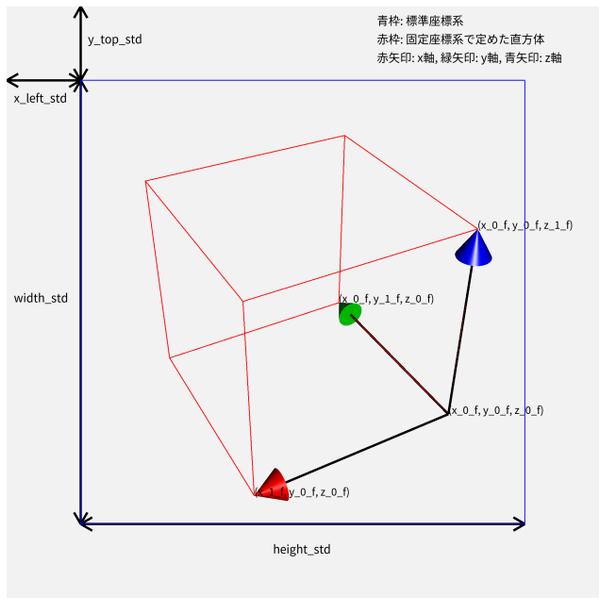
```
void g_def_scale_3D(int id,
double x_0, double x_1,
double y_0, double y_1,
double z_0, double z_1,
double x_0_f, double x_1_f,
double y_0_f, double y_1_f,
double z_0_f, double z_1_f,
double x_left_std, double y_top_std,
double width_std, double height_std
);
```

自由座標系

固定座標系では
x_0_f < x_1_f かつy_0_f < y_1_f かつz_0_f < z_1_f
としないとエラーになる.
（∵右手系を定めるため，）

自由座標系では，このような制限はない．仮に　x_0 > x_1
と定めると，X軸の方向が反転する．つまり，左手系空間を定めることができる.



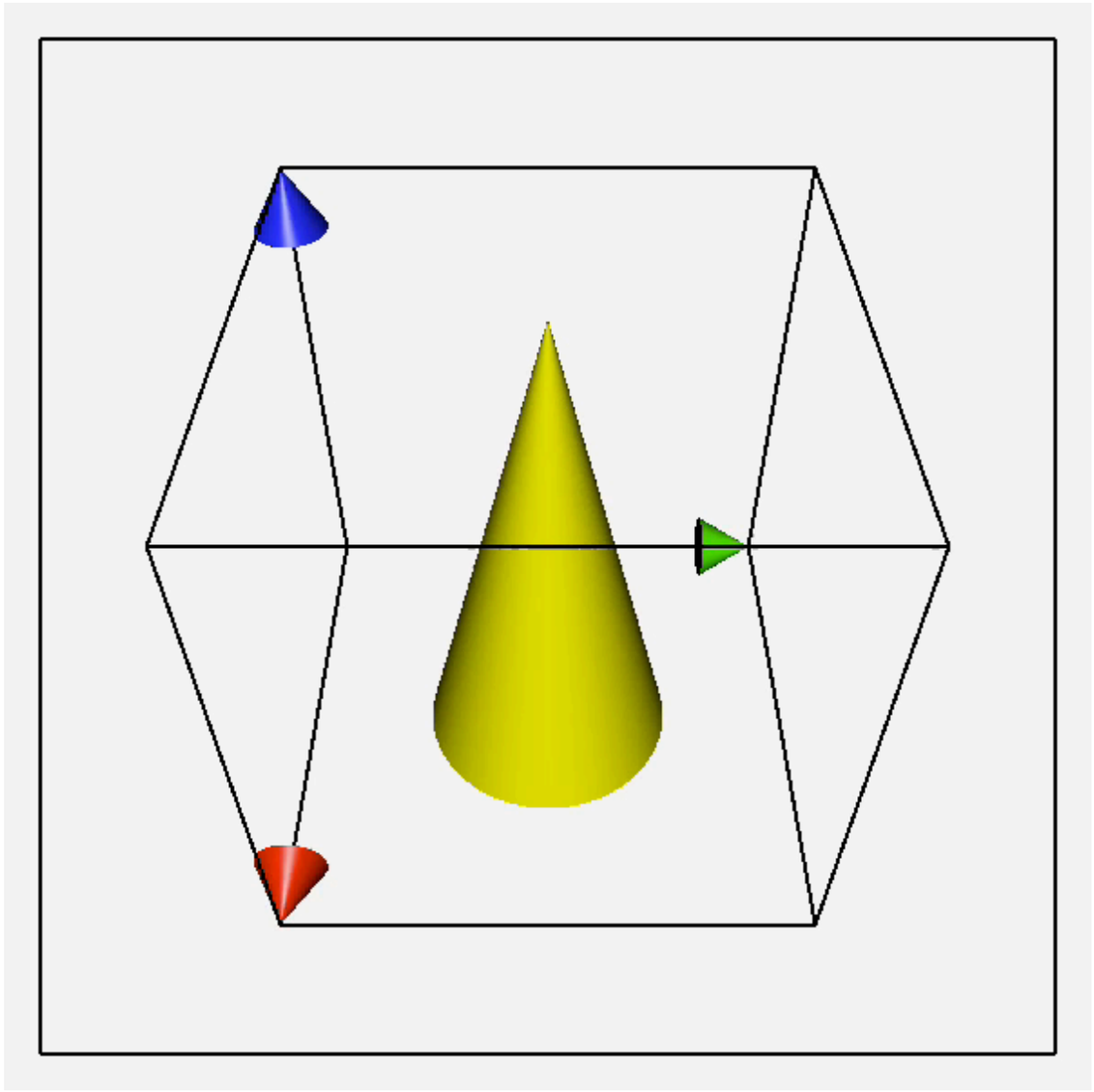


note: g_def_scale_3Dは3D空間を定義する関数として「最強」

GLSC3Dの関数はほぼGLSC2Dの関数の仕様を
受け継いでいる．そのため，他の関数の説明に関しては割愛．

GLSC3Dの便利な新機能

g_vision.cを改変して
赤字のように変更.

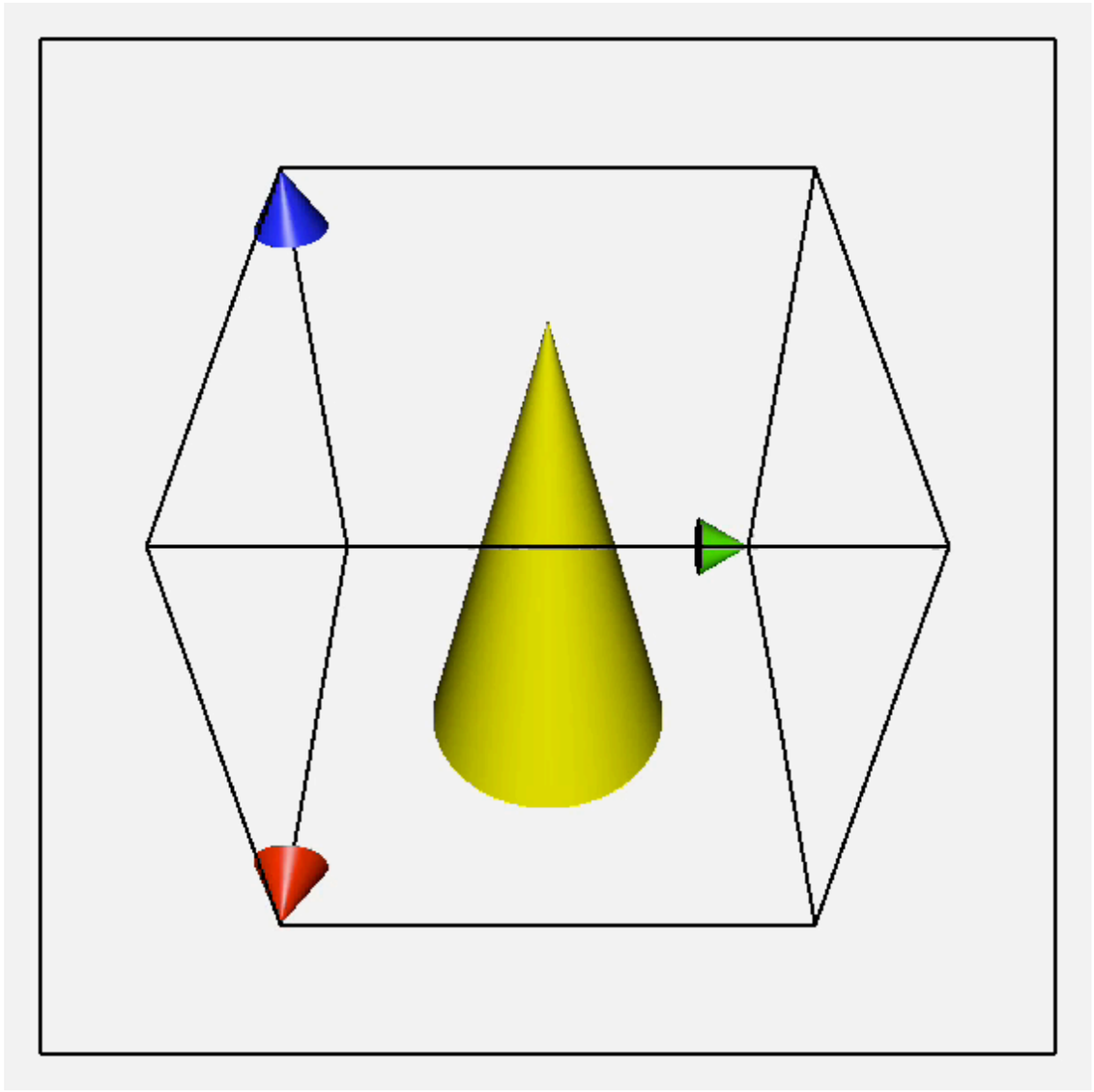


```
#include<stdio.h>
#include<glsc3d_3.h>
#define WX     (600)
#define WY     (600)
int main()
{
        g_init("Window", WX, WY);
        g_def_scale_3D_fix(0,
                                                -1, 1,
                                                -1, 1,
                                                -1, 1,
                                                20.0, 20.0,
                                                WX - 40.0, WY - 40.0);

        for (int i_time = 0;; i_time++)
        {
                double dt = 0.01;
                double t = i_time * dt;
                g_vision(0,
                                        3 * cos(t), 3 * sin(t), 3,
                                        0, 0, 1,
                                        1.0);
                g_cls();
                g_sel_scale(0);
                g_boundary();
                g_box_3D(-1, 1, -1, 1, -1, 1, 1, 0);
                //X-Axes
                g_area_color(1, 0, 0, 1);
                g_arrow_3D(-1, -1, -1,
                                        1.0, 0.0, 0.0,
                                        2.0, 0.25,
                                        0, 1);
                //Y-Axes
                g_area_color(0, 1, 0, 1);
                g_arrow_3D(-1, -1, -1,
                                        0.0, 1.0, 0.0,
                                        2.0, 0.25,
                                        0, 1);
                //Z-Axes
                g_area_color(0, 0, 1, 1);
                g_arrow_3D(-1, -1, -1,
                                        0.0, 0.0, 1.0,
                                        2.0, 0.25,
                                        0, 1);

                g_area_color(1, 1, 0, 1);
                g_cone_3D(0, 0, -1,
                                        0, 0, 1,
                                        0.5, 2,
                                        0, 1);
                g_finish();
        }
        return 0;
}
```

16_NewFunction1.c

GLSC3Dの便利な新機能

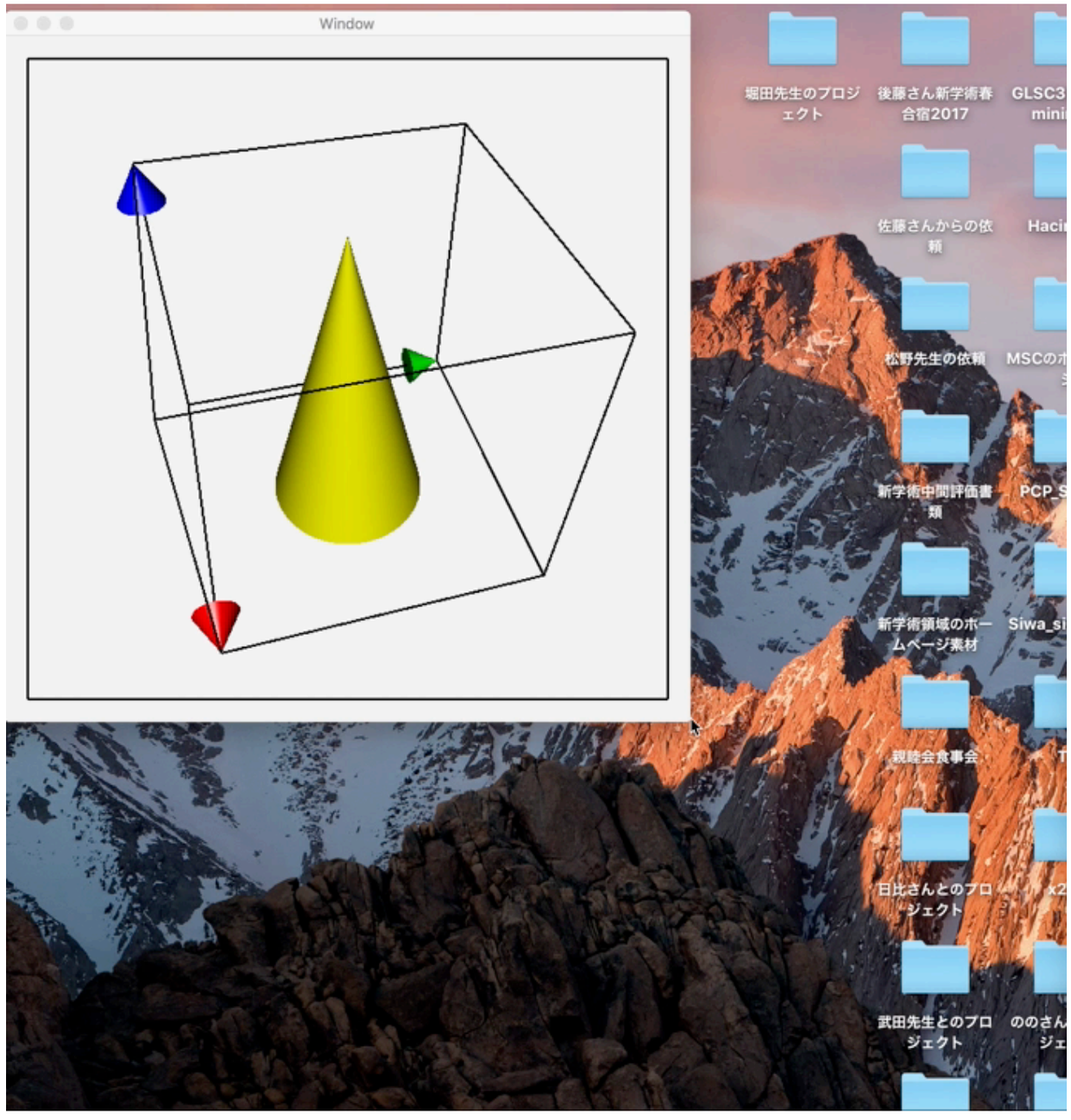
g_vision.cを改変して
赤字のように変更.



```
#include<stdio.h>
#include<glsc3d_3.h>
#define WX     (600)
#define WY     (600)
int main()
{
        g_init("Window", WX, WY);
        g_def_scale_3D_fix(0,
                                                -1, 1,
                                                -1, 1,
                                                -1, 1,
                                                20.0, 20.0,
                                                WX - 40.0, WY - 40.0);

        for (int i_time = 0;; i_time++)
        {
                double dt = 0.01;
                double t = i_time * dt;
                g_vision(0,
                                        3 * cos(t), 3 * sin(t), 3,
                                        0, 0, 1,
                                        1.0);
                g_cls();
                g_sel_scale(0);
                g_boundary();
                g_box_3D(-1, 1, -1, 1, -1, 1, 1, 0);
                //X-Axes
                g_area_color(1, 0, 0, 1);
                g_arrow_3D(-1, -1, -1,
                                        1.0, 0.0, 0.0,
                                        2.0, 0.25,
                                        0, 1);
                //Y-Axes
                g_area_color(0, 1, 0, 1);
                g_arrow_3D(-1, -1, -1,
                                        0.0, 1.0, 0.0,
                                        2.0, 0.25,
                                        0, 1);
                //Z-Axes
                g_area_color(0, 0, 1, 1);
                g_arrow_3D(-1, -1, -1,
                                        0.0, 0.0, 1.0,
                                        2.0, 0.25,
                                        0, 1);

                g_area_color(1, 1, 0, 1);
                g_cone_3D(0, 0, -1,
                                        0, 0, 1,
                                        0.5, 2,
                                        0, 1);
                g_finish();
        }
        return 0;
}
```

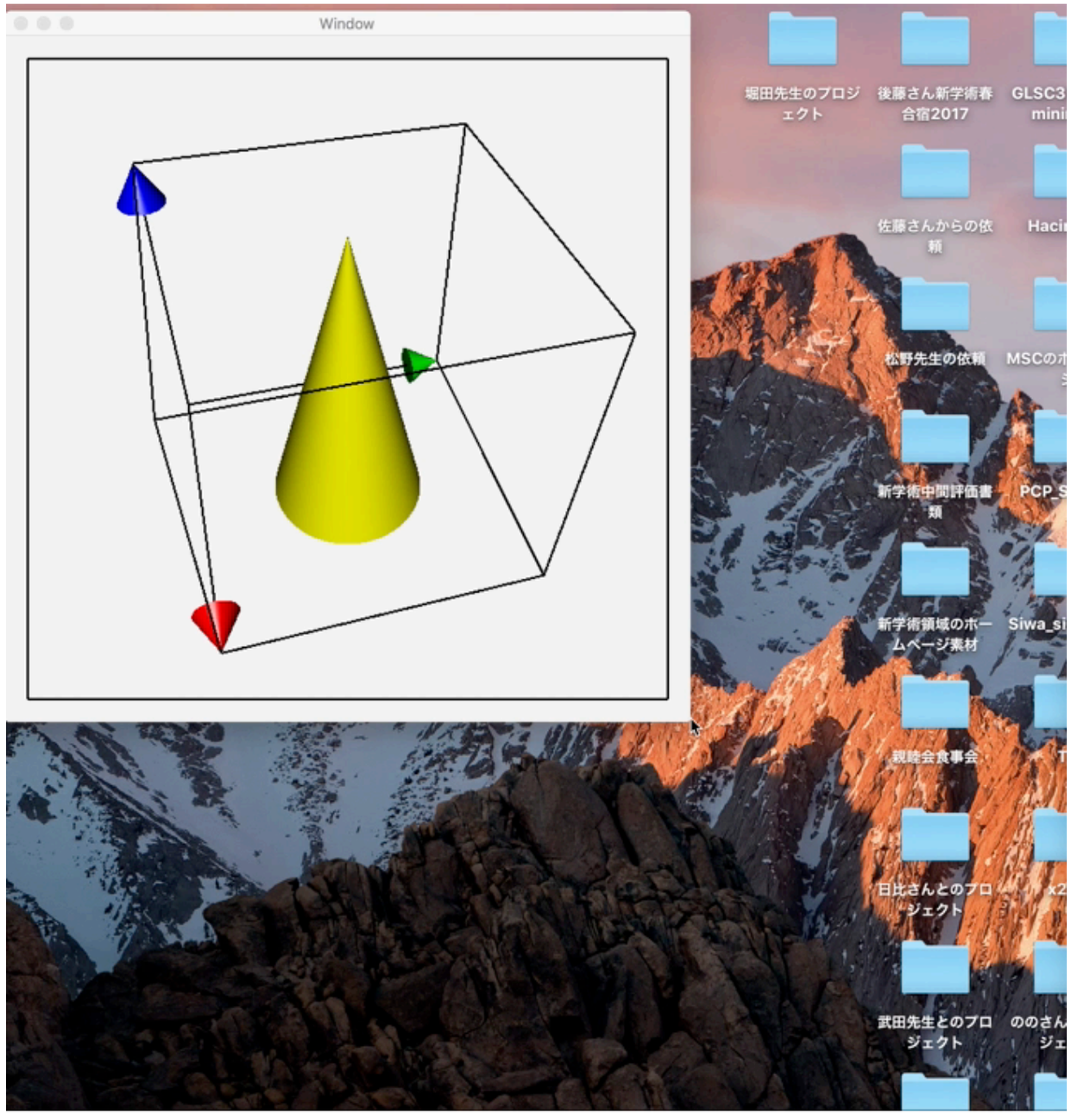
16_NewFunction1.c

GLSC3Dの便利な新機能



描画中にサイズを変更できる

GLSC3Dの便利な新機能



描画中にサイズを変更できる

GLSC3Dの便利な新機能

MacBookProなどの高解像度ディスプレイを搭載するマシンの場合，画面全体が緻密になる.

線描画などをシャープに表示することができる.

```
#include<stdio.h>
#include<glsc3d_3.h>
#define WX      (600)
#define WY      (600)
int main()
{
        g_enable_highdpi();
        g_set_antialiasing(4);
        g_init("Window", WX, WY);
        g_def_scale_3D_fix(0,
                        -1, 1,
                        -1, 1,
                        -1, 1,
                        20.0, 20.0,
                        WX - 40.0, WY - 40.0);

}
```

note:きれいな画像になるが，むやみに使用すると，描画が遅くなるので注意

17_NewFunction2.c

GLSC3Dの使用に関して

１．GLSC3Dのソースを改変し，新しい関数を追加することを認めます．再配布することも認めますが，その場合は質問のメール等が私のもとへ来ないように配慮してください．

２．論文や発表などで，GLSC3Dの可視化を用いた場合は，そのことを明記し，作者にメールをしてください．
これは，GLSC3Dの使用実態を明らかにすることで，将来GLSC3Dをさらに大きなプロジェクトとしてステージアップさせる際に，必要な情報を得るためです．

GLSC2Dユーザー必見
GLSC3Dへの関数の変換に関して

旧 glsc のプログラムを glsc3d に移行するときのプログラム書き換えで、するべきことのリスト。

0. #include <glsc.h> を #include <glsc3d_3.h> に変更。
1. g_init の ウィンドウサイズ指定は int でピクセル単位にする。
2. g_device, g_term は消す。
3. g_def_scale は g_def_scale_2D に変更。std 座標はピクセル単位になっているので変更（3倍ぐらい？）する必要あり。
4. 描画終わりに g_finish を挿入。
5. 全ての色指定は（r, g, b, a）に変更。
6. g_move, g_plot, g_box, g_circle の後ろに _2D をつける。
7. g_polyline, g_polygon, g_data_plot, g_contln の後ろに _2D をつける。
8. g_text は g_text_standard に変更。std座標の変更。
9. g_bird_view, g_hidden は g_bird_view_3D に変更。引数も変更。
10. g_sleep(G_STOP); の G_STOPマクロはないので、-1 などとおく。
11. g_def_scale で座標を決めるのに使った長方形の枠を描くときは、従来は g_box で描いていたが、glsc_3d では g_box_2d ではなく g_boundary を使う。
12. g_sel_scale が呼ばれた時点では g_clipping(0) になっているので、g_boundary で描かれる枠の外にも絵を描きたいときは、g_sel_scale を呼んだ直後に g_clipping(1) としておく。
13. アニメを作るとき、動くオブジェクトを部分的に塗りつぶして、再描画するという方法を取っていた場合、うまくいかなくなっている。 g_cls でクリアして、全体を再描画するように変更する。

文責：小林亮