計算物理学II (第 6 回)

今回の内容

- 前回の復習(変数・代入文・四則演算・ループ・組込関数)
- 配列
  - 変数を複数並べたもの。ベクトルや行列を表現
- if文
  - プログラムは上から順番に一文ずつ実行するが、if文によって特定の条件を満たしたときだけ実行することが可能。
  - do文(for文)と組み合わせて使うことで条件が満たされるまで繰り返す、などの制御も可能。

- 詳細は講義資料の文法のまとめを参照

コーディングの環境について

- ホームディレクトリ以外で作業してください。
  - 新しいファイルができたかどうかがlsで探しにくくないですか。

- emacsは&をつけて実行、書き終わっても終了しない
  - コンパイルが一回で成功することはなかなかありません。
  - 端末とemacs両方を常に使えるようにしておくと効率があがります。
- emacs起動後 **C-x 1**でウィンドウ領域全部を使えます。
- emacsでコードを修正したら保存(**C-x C-s**)
- emacsで新しいファイルを開く場合は**C-x C-f**、その後ファイル名を入力)

コーディングの環境について

- 最近実行したコマンドは”↑”キーを複数回押すとたどれます。
  - コンパイル、実行は通常何度も行いますので毎回gcc/gfortranを入力する必要はありません。

- 端末やemacsは複数起動できます。
  - サンプルを見ながら書く場合はサンプルもemacsで開くとよいです。

- emacs上でTabキーを押すと補完・字下げをしてくれます。
  - Fortran：end do, end programなどの入力時にはendまで入力してTabキー
  - 字下げが正しく行われない場合は手前で文法ミスがある可能性があります。

プログラムの基本構造

Fortran

```
program プログラム名

宣言部 (implicit none宣言、変数宣言)


実行部


end program プログラム名
```

プログラム名は英字から始まる適当な名前

C

```
#include<ヘッダファイル名.h>

int main(){
    宣言部 (変数宣言など)

    実行部 (計算など)

}
```

プログラムはmain関数として実行される。
ヘッダファイルにはプログラム内で
ライブラリ関数などを用いる場合に読み込む。
数値計算で必要となるのは
stdio.h,math.h,stdlib.h,complex.hなど

変数

- 変数：数値・文字などのデータを格納する領域。名前をつけて区別。
- 型があり、FortranやCでは宣言部で全ての型を明示的に指定する

型	Fortran	C	サイズ
整数型	**integer**	**int**	**4バイト（= 32ビット）**
単精度実数型 (物理の数値計算ではあまり使わない)	real, real*4, real(4)	float	4バイト
倍精度実数型	double precision, **real*8**, real(8)	**double**	**8バイト**
複素数型(物理の数値計算ではあまり使わない)	complex, complex*8	float complex (complex.hが必要)	8バイト
倍精度複素数型	complex(kind(0d0)), **complex*16**, double complex	double complex (complex.hが必要)	16バイト
論理型	**logical**	bool (stdbool.hが必要)	trueかfalseか
文字型	**character**	**char**	**1文字1バイト**

太字はよく使うもの。
変数名：英数字とアンダーラインが使える。英字から始まる

四則演算と代入文

- 数学の=(イコール)は左辺と右辺が等しいことを表すが
  プログラムの=は**右辺の計算結果を左辺の変数に代入する**、の意味
- 四則演算は +, -, *, /
- 累乗は Fortranでは** で表す。Cにはない (何度も掛けるか、数学関数のpowを使う)
- **同じ型同士の演算は同じ型になる。**
- 整数と実数の演算→実数、実数と複素数の演算→複素数

Fortran

```
a = 1.0d0
i = i + 5
a = 4**3
a = 5 / 2
```

C

```
a = 1.0;
i = i + 5; または i += 5;
a = 4*4*4; または a = pow(4.0,3.0);
a = 5 / 2;
```

Cは文の終わりに;が必要

倍精度実数型のaに1.0を代入
整数型のiの値を5増やす
aに4の3乗を代入
aは整数型でも実数型でも
2となる(整数と整数の演算は整数)
2.5にしたければ
5.0/2や5.0/2.0、5/2.0などを代入

実行文

- プログラムの実行文は上から順番に一行ずつ実行される(重要)

Fortran

```
integer :: a, b, c

a = 2
b = 5
c = a + b
```

C

```
int a, b, c;

a = 2;
b = 5;
c = a + b;
```

整数変数a,b,cを宣言

変数aに2を代入
変数bに5を代入
変数cにa + bを代入。この時点でaとbに値が入っているので
2+5が計算されて変数cに7が代入される

```
integer :: a, b, c

c = a + b
a = 2
b = 5
```

```
int a, b, c;

c = a + b;
a = 2;
b = 5;
```

順番を変えると結果が変わる。
cにa+bを代入。aとbに値がまだ入っていない。システム依存になるが
おそらくaにもbにも初期値の0が入っているのでcもおそらく0となる
aに2を代入
bに5を代入
cの値はその後変更していないので0のまま。

数学的には同じに見えるがプログラムでは結果は異なる。
プログラムを書くときは**右辺に使う変数に値が入っているかを確認**しながら書く

ループ(doループ/forループ)

- 実行文は上から一行ずつ実行するのが原則だがよく使う例外は2つ(ループ、条件文)
- doループ(Fortran)：do~end doで囲まれた領域をカウンタを変えて繰り返し実行
- forループ(C) **for(式1(実行文); 式2(条件文); 式3(実行文)){繰り返し文}**
  - 式1を実行(初期化)、式2の条件を判定、満たされていれば繰り返し文を実行
  - 式3を実行してから式2の条件判定に戻る

Fortran

```
sum = 0
do i = 1, n
    sum = sum + i
end do
```

C

```
sum = 0;
for(i = 1;i<n ; i++){
    sum = sum + i;
}
```

$$\sum_{i=1}^{n} i$$ を計算していることになっている

ループは和や積の計算、ベクトル、行列などindexを変えて行う計算、
パラメータを変えて同じ計算を繰り返すときなどに使われる

標準入出力

- 標準入力(キーボードからの入力)

Fortran

```
read (*,*) var
read (5,*) var1, var2
```

C

```
scanf(“%d”, &var):
scanf(“%lf %lf”, &var1, &var2);
```

C言語では入出力の型を明示する必要がある
%d：整数型、%lf倍精度実数型
scanfのときには変数名の前に&をつける
(変数のアドレス)

- 標準出力(画面への出力)
- 標準エラー出力(画面への出力)

```
write (*,*) var
write (6,*) var
write (0,*) var
print *, var
```

```
printf(“%d”, var);
printf(”var1 = %f,   var2 = %f¥n”, var1,var2);

fprintf(stderr, “var1 = %f¥n”, var1); //エラー出力
```

C言語の単精度・倍精度実数の出力は%fを使う

Fortran: read, writeの１つ目の引数で**ファイル装置番号**を指定する。*とすると標準入出力になるが、
Fortranでは5が標準入力、6が標準出力、0が標準エラー出力となっている
シェル側でリダイレクトを使うことで標準入出力先をファイルなどに変更することができる
計算した結果はwriteかprintで必ず出力する

組込関数

- 数学関数 (log, exp, sqrt, sin, cos, tanなど)は用意されている
- 角度の単位はラジアン

- C言語で数学関数を使うときの注意：
  - math.hをインクルードする
  - コンパイル時にオプション –lm をつける
  - **gcc ex4.c –lm**

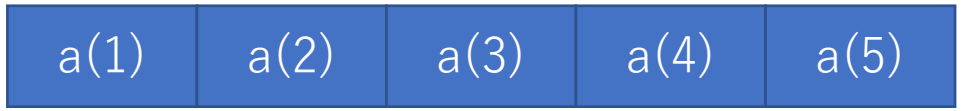
配列

- 配列とは：変数が複数並んだもの(メモリ上でも並んで配置される)。ベクトルや行列が表現できる。

配列の宣言の基本

Fortran

```
double precision :: a(5)   ! aは5つの要素を持つ倍精度実数変数の配列
```



←メモリ上にこのように5つの倍精度実数変数の領域が連続して確保される

C

```
double a[5];    // aは5つの要素を持つ倍精度実数変数の配列
```

←C言語では配列のラベルは0から始まる。
メモリ上にこのように5つの倍精度実数変数の領域が連続して確保される

配列の初期値設定

宣言部で設定

Fortran

```
double precision ::  a(1:5) = 0.0d0
integer :: k(1:5) = (/1, 2, 3, 4, 5/)
```

C

```
double a[5] = {0.,0.,0.,0.,0.};
int k[5] = {1, 2, 3, 4, 5};
```

実行文として設定してもよい(こちらのほうが自由度が高い)。**ループを使う**

```
do i = 1, 5
    a(i) = 0.0d0
end do
```

```
for (i=0;i<5;i++){
    a[i] = 0.0;
}
```

異なる配列の違う添字を持つ要素の代入・演算も可能

```
do i = 1, 5
     a(i) = b(i+3)
end do
```

```
for(i=0;i<5;i++){
     a[i] = b[i+3];
}
```

1次元配列の計算例

ベクトル和

ベクトルの内積

Fortran

```
real*8, dimension(1:5)  :: a, b, c
real*8 :: adotb

! ここでaとbには値を代入しておく(省略)

do i = 1, 5
    c(i) = a(i) + b(i)
end do
do i = 1, 5
    write(*,*) “c(“, i, “)=“,  c(i)
end do

adotb = 0.0d0
do i = 1, 5
    adotb = adotb + a(i)*b(i)
end do
print *, “a dot b = “, adotb
```

C

```
double a[5], b[5], c[5];
double adotb;

// ここでaとbには値を代入しておく(省略)

for( i = 0; i<5; i++){
    c[i] = a[i] + b[i];
}
for( i = 0;i<5;i++){
    printf(“c(%d)=%f¥n”, i,  c[i]);
}

adotb = 0.0;
for( i = 0; i<5; i++){
    adotb += a[i]*b[i];
}
printf(“a dot b = %f¥n“, adotb);
```

1次元配列の組み込み関数(Fortran)

和

```
c = sum(a(1:10))        ! a(1)からa(10)までの和をcに代入
```

最大値・最小値

```
d = maxval(a(1:n))   ! a(1)からa(n)の中の最大値をdに代入
e = minval(a(1:n))   ! a(1)からa(n)の中の最小値をeに代入
```

2つの1次元配列の内積

```
f = dot_product(a(1:n), b(1:n)) ! n要素の配列aとbの内積をfに代入
```

a, bが複素数型の場合は aの複素共役とbの積が計算される

行列

- 2次元配列で行列が表現できる
- 2次元配列の宣言と初期化

```
real*8, dimension(1:5,1:5) :: b
または real*8 :: b(5,5)
または real*8 :: b(1:5,1:5)など

do j = 1, 5
    do i = 1, 5
        b(i,j) = 0.0d0
    end do
end do
```

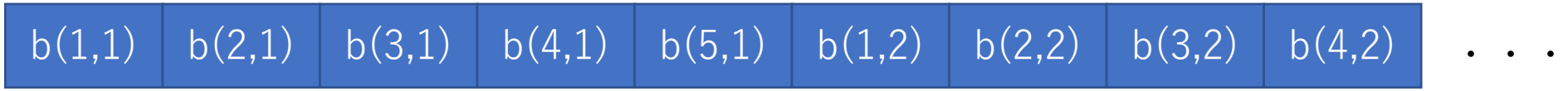
```
double b[5][5];

for(i=0;i<5;i++){
    for(j=0;j<5;j++){
        b[i][j] = 0.0;
    }
}
```

Fortran: b(1:5,1:5) = 0.0d0　!0を代入するだけなら1行でもかける。

2次元配列のメモリ上での配置

Fortranでの多次元配列 (b(5,5)の場合) は



C言語での多次元配列(b[5][5]の場合)は

の順でメモリ上に配置される。
doループ/forループで配列の値を使って計算する場合はメモリ上で連続的にアクセスするほうがよい(キャッシュミスが少なくなる)

例えばFortranではdoループは１つ目のインデックスを先に回すほうがよい

```
do j = 1, 5
    do i = 1, 5
        b(i,j) = 0.0d0
    end do
end do
```

行列の和・値の出力

Fortran

```
real*8, dimension(1:5,1:5) :: a, b, c

! a, bに値をここで代入(省略)

do j = 1, 5
    do i = 1, 5
        c(i,j) = a(i,j) + b(i,j)
    end do
end do

do i = 1, 5
    write(*,*) (c(i,j), j = 1, 5)
end do
```

C

```
double a[5][5], b[5][5], c[5][5];

/* a, bに値をここで代入(省略) */

for(i = 0; i<5; i++){
    for(j = 0; j<5; j++){
        c[i][j] = a[i][j] + b[i][j];
    }
}

for (i = 0; i<5; i++){
    for(j = 0; j<5; j++){
        printf(" %f ", c[i][j]);
    }
    printf("¥n");
}
```

行列の積

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$ 一つの要素を計算するのにループが必要

Fortran

```
real*8, dimension(1:5,1:5) :: amat, bmat, cmat

! amatとbmatの値を代入

do j = 1, 5
   do i = 1, 5
      cmat(i,j) = 0.0d0
   end do
end do

do j = 1, 5
   do i = 1, 5
      do k = 1, 5
         cmat(i,j) = cmat(i,j) + amat(i,k) * bmat(k,j)
      end do
   end do
end do
```

C

```
double amat[5][5], bmat[5][5], cmat[5][5];

/* amatとbmatの値をここで代入 */

for(i = 0; i<5; i++){
   for(j = 0; j<5; j++){
      cmat[i][j] = 0.0;

      for (k = 0; k<5; k++){
         cmat[i][j] += amat[i][k] * bmat[k][j];
      }
   }
}
```

2次元配列の組み込み関数(Fortran)

転置行列

```
bmat(1:n,1:n) = transpose(amat(1:n,1:n))     ! bmatにamatの転置行列を代入
```

行列積

```
cmat(1:n,1:m) = matmul( amat(1:n,1:k), bmat(1:k,1:m))
! (n×k行列のamat とk×m行列のbmatの積をcmatに代入(n×m行列)
```

matmulは経験上計算速度が早くないので大行列の計算では他のライブラリを使うほうがよい

条件文

- 実行文は上から一行ずつ実行するのが原則だがよく使う例外は2つだけ
- if文：条件を論理式で指定し条件を満たすかどうかで処理内容を変える

Fortran

```
integer :: i

read (*,*) i

if ( i<0) i=-i

print *, “i = “, i
```

C

```
int  i;

scanf(”%d”, &i);

if (i<0) i=-i;


printf(“i=%d¥n”,i);
```

条件文

Fortran

```
if (論理式) 実行文

if (論理式) then
    実行文1
    実行文2
end if

if (論理式) then
    実行文1
else
    実行文2
end if

if (論理式1) then
    実行文1
else if (論理式2) then
    実行文2
else
    実行文3
end if
```

C

```
if (論理式) 実行文;

if (論理式){
    実行文1;
    実行文2;
}

if (論理式){
    実行文1;
}else{
    実行文2;
}

if (論理式1){
    実行文1;
}else if (論理式2){
    実行文2;
}else{
    実行文3;
}
```

論理式が真のときのみ実行文が実行される(実行文が1行の場合)

論理式が真のときのみ実行文1と実行文2が実行される。
実行文が複数行ある場合はこのスタイル

論理式が真の場合は実行文1が実行され、
偽の場合は実行文2が実行される

論理式1が真のときは実行文1を実行
論理式1が偽かつ論理式2が真の時は実行文2を実行
論理式1も論理式2も偽の場合は 実行文3を実行

else ifは複数あってもよい。

論理式

Fortran	Fortran	C	
a == b	a .eq. b	a == b	aがbと等しい時に真 (equal)
a /= b	a .ne. b	a != b	aがbと等しくないとき真(not equal)
a >= b	a .ge. b	a >= b	aがb以上の時に真 (greater equal)
a > b	a .gt. b	a > b	aがbより大きい時に真(greater than)
a <= b	a .le. b	a <= b	aがb以下の時に真(less equal)
a < b	a .lt. b	a < b	aがbより小さい時に真(less than)

等しい時は=一つではなく2つなので注意

Fortran 論理演算子	意味	使用例
.not.	(右)以外	.not. a==b
.and.	かつ	a==b .and. b==c
.or.	もしくは	a==b .or. b==c
.eqv.	論理値が等しい	a<0 .eqv b<0 (aとbの符号が同じ)
.neqv.	論理値が異なる	a<0 .neqv. b<0 (aとbの符号が違う)

C 論理演算子	意味	使用例
!	(右)以外	! a==b
&&	かつ	a==b && b==c
||	もしくは	a==b || b==c

プログラムの終了

- プログラムは最終行まで到達すれば終了
- 条件文を使うと途中で終了させることも可能
- Fortran: stop文で終了。 stop “文字列” とすると文字列が標準エラー出力に出力
  - プログラムの一番最後にstopと正常終了のメッセージを書く場合もある
- C: return文でmain関数から抜けてプログラム終了。**異常終了の場合は0でない値を返す。**

Fortran

```
read(*,*) x                                          ! xを標準出力から読み込む
if( x<0.0d0) stop “x must be zero or positive”       ! xが負であればプログラムを終了
                                                     ! xがゼロか正であれば以下の処理が行われる
print *, “sqrt(x) = “, sqrt(x)
```

C

```
double x;
    scanf(“%lf”, &x);                                    // xを標準出力から読み込む
    if( x<0.0){
        fprintf(stderr, “x is negative¥n”);
        return(-1);        // xが負であれば-1を返してプログラムを終了
    }
printf(“sqrt(x) = %f¥n”, sqrt(x));                       // xがゼロか正であれば以下の処理が行われる
```

条件文の例：クロネッカーのデルタ

Fortran

```
integer :: delta, i, j

read (*,*) i, j

if (i == j) then
    delta = 1
else
    delta = 0
end if

print *, delta
```

C

```
int delta, i, j;

scanf(“%d %d”,  &i, &j);

if (i == j){
    delta = 1;
}else{
    delta = 0;
}

printf(“delta=%d¥n”, delta);
```

Fortran

```
integer :: delta, i, j

read(*,*) i,j

delta = 0
if(i==j) delta = 1

print *, delta
```

C

```
int delta, i, j;

scanf(“%d %d”,  &i, &j);

delta = 0;
if(i==j) delta = 1;

printf(“delta = %d¥n”, delta);
```

doループとif文の組み合わせ

無限ループ

```
do
!ここでいろいろ計算
if ( a<0 ) exit
x=x+0.10d0
end do
```

```
for(x=0.1; ; x+=0.1){
   // ここでいろいろ計算
   a = ....;
    if ( a<0 ) break;
}
```

exit/break文はループから抜け、ループ文の直後に移動
条件が満たされない場合は計算が終わらないため実行してみて終わらない場合は**強制終了(C-c)する**

cycle(Fortran), continue(C)：ループの先頭まで戻り、カウンタを次の値に進める

```
do i = 1, n

   a(i) = ....ここで何かを計算する

    if ( a(i)>0 ) cycle
    !a(i)が正なら戻ってカウンタを一つ回す

    ! a(i)が0か負の場合だけこの領域で処理が行われる
end do
```

```
for( i = 0;i<n;i++){

   a[i] = ....; // ここで何かを計算する

    if (a[i]>0) continue; //a[i]>0なら戻ってカウンタを一つ回す

    /* a[i]が0か負の場合だけこの領域で処理が行われる */

}
```

演習

- 自分で書けそうなら資料を見て、難しそうならサンプルのプログラムを写して書いてコンパイル、実行してください。

- 配列：第５回演習問題の(5)-(7)。条件文は第６回の(8)-(11)

- 第１回のレポート締切は11月18日(金)です。