Fortran ハンドブック　補足説明

–サブルーチンの引数とメモリアドレス–

2018 年 6 月 12 日　田口俊弘

1.　はじめに

本書の p.150 に示した 2 次の正方行列の固有値と固有ベクトルの解答例が以下のようになっていることについて疑問を持たれる方が多いようなので，少し詳しく説明したいと思います．

```
program eigenvalue_2
   implicit none
   real a(2,2),x(2,2),eigen(2)
   real u,v,d
   real, parameter :: eps = 1e-14
   ..........
   call check_eigenvalue(a,eigen(1),x(1,1),2)
   call check_eigenvalue(a,eigen(2),x(1,2),2)
end program eigenvalue_2

subroutine check_eigenvalue(a,eigen,x,n)
   implicit none
   real a(n,n),eigen,x(n),xe,err
   integer n,i,j
   ..........
```

ここで問題になるのは，サブルーチンをコールする側の第 3 引数が

```
   call check_eigenvalue(a,eigen(1),x(1,1),2)
                                    ^^^^^^
   call check_eigenvalue(a,eigen(2),x(1,2),2)
                                    ^^^^^^
```

のように，2 次元配列で指定されているのに，サブルーチン側の宣言では，

```
subroutine check_eigenvalue(a,eigen,x,n)
   implicit none
   real a(n,n),eigen,x(n),xe,err
                      ^^^^
```

のように，1 次元配列になっているのが理解しにくいということにあるようです．これには，コンピュータのしくみを少し理解してもらわないといけないので，そのあたりから説明したいと思います．

2.　メモリとメモリアドレス

プログラムにおける「変数」というのは，コンピュータでいえば,「メモリ」という記憶装置の中の一つに対応しています．例えば，

```
   real x,y
```

のように宣言した場合には，x や y という実数型変数を使うことができて，プログラム中で

```
x = 10
y = x**2
```

のように使うことが可能ですが，コンピュータの動作として「y=x**2」という式を考えると，

メモリ x に入っている数値を２乗する　→　その結果をメモリ y に書き込む

となります．このとき，コンピュータがメモリを指定するのは，変数「x」や「y」という文字ではなく，あらかじめそれらに対応するように決めた数値「メモリアドレス」です.「メモリ番地」あるいは単に「番地」と呼ぶこともあります．よって，もう少し正確に表現すると，

x に対応する番地で指定したメモリに入っている数値を読み出して２乗する
→　その結果を y に対応する番地で指定したメモリに書き込む

となります．ここまでは，単一変数の場合ですが，配列の場合も同じです．配列というのは，アドレスが連続したメモリで実現されています．例えば,「real a(10),b(3,2)」のように宣言した場合の配列は，下図のようなイメージです．

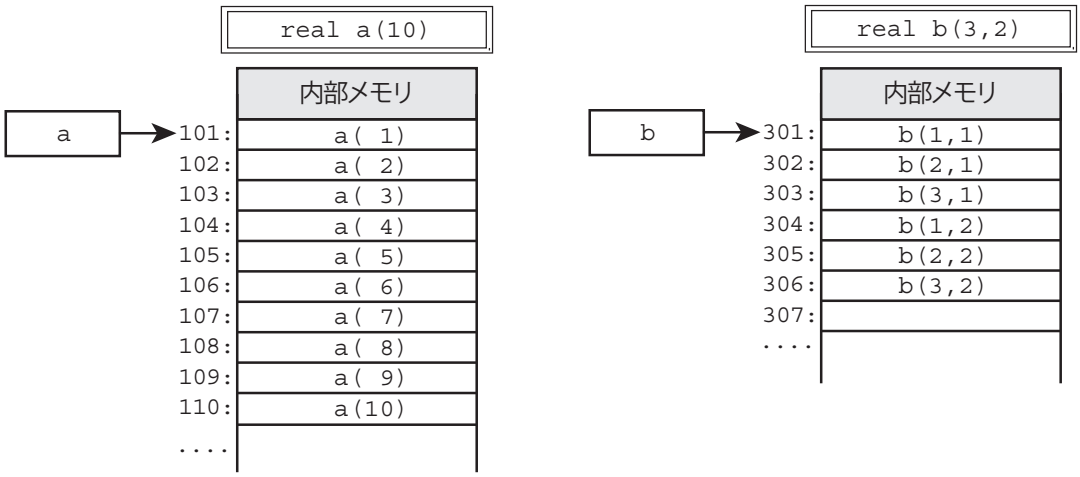

図 1.　1 次元や 2 次元配列のメモリアドレス

この図の左側は 1 次元配列で，この例では配列の先頭要素 a(1) のアドレスを 101 としていますが，そうすると，a(2) のアドレスは 102，a(3) のアドレスは，103，... のように続いていることになります．

これに対し，右側の 2 次元配列もやはり 1 次元的に並んでいます．このとき，Fortran では左の第 1 添字が先に進んでいることに注意して下さい．この例では配列の先頭要素 b(1,1) のアドレスを 301 としていますが，そうすると，b(2,1) のアドレスは 302，b(3,1) のアドレスは 303 となります．b(3,1) になると，第 1 添字が最大値の 3 に到達するので，次の要素は，b(1,2) になり，このアドレスが 304 になります．

以上のことから，配列の指定した要素に値を代入する式，

```
a(5) = 10
b(1,2) = 3
```

は，次のような動作になります．

配列の先頭番地 a(1) から数えて 5 番目の番地で指定したメモリに値（10）を書き込む
配列の先頭番地 b(1,1) から数えて 4 番目の番地で指定したメモリに値（3）を書き込む

この「メモリアドレスで変数を指定する」という動作がサブルーチンの引数指定にとって重要なのです.

3. サブルーチンの引数とメモリアドレス

さて，本書 p.38 にも書いてありますが，Fortran では，変数をサブルーチンまたは関数の引数に与えたときは，その変数に書き込まれた値を与えるのではなく，変数のメモリアドレスを与えます．これは，サブルーチン側で引数に値を代入したときに，コール側の変数に代入される「戻り値（p.37)」を可能にするためです.

例えば，下図のようなメインプログラム（左）と，その中でコールしているサブルーチン sub（右）を見てみましょう。

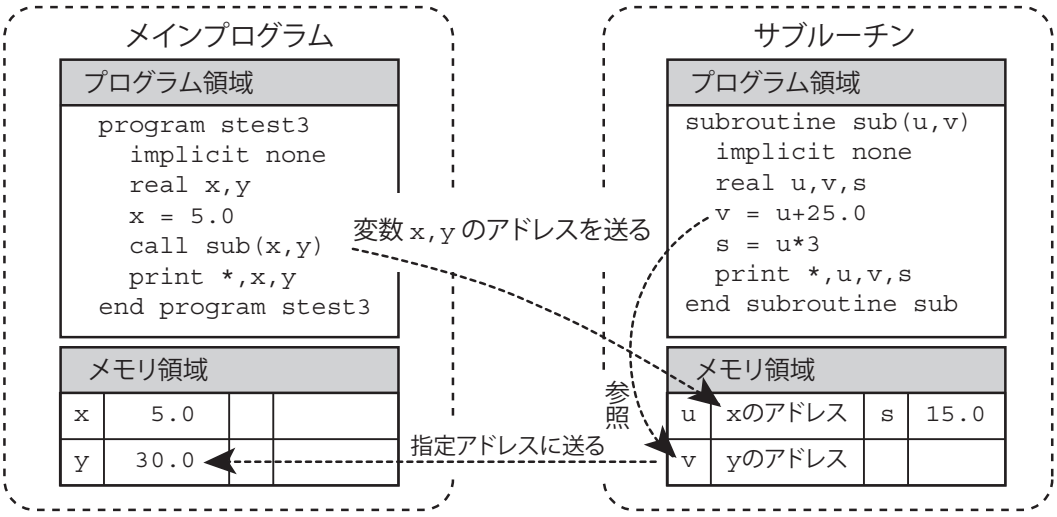

図 2. サブルーチンの引数の受け渡し

通常，プログラム中で宣言された変数は，そのプログラムに付随したメモリ領域に所属しています．このため，メインプログラムには，メインプログラムのメモリ領域があり，サブルーチンにはサブルーチンのメモリ領域があって，図のようにそれぞれは独立しています．よって，メインプログラムで宣言されている x や y はサブルーチンからは見えません．そこで，引数にこれらを与えると，サブルーチンのメモリ領域の対応する変数のメモリには引数の中に入っている値ではなく，引数のメモリアドレスが与えられます．サブルーチンでは，このアドレス情報を使ってコール側変数の読み書きを行うしくみになっているので，サブルーチン中の計算で得られた結果をメインプログラムの変数に代入することが可能になります．これが戻り値です．これに対し，サブルーチンの引数ではない変数 s は，通常の変数なので，代入すると値がそのまま入ります．すなわち，サブルーチンにおける引数変数とそれ以外の変数は，見た目は同じなのですが，意味は全く違うのです.

さて，逆に言えば，呼び出されたサブルーチンには，そのアドレス情報しか送られてこないので，そのメモリアドレスをどう解釈するかはサブルーチン側で決めることができます．というか，サブルーチン側で決めるしかないのです.

図 1 に示したように，1 次元配列 a(10) は,

```
a(1), a(2), a(3), ..., a(9), a(10)
```

という 10 個の並んだメモリであり，2 次元配列 b(3,2) は,

```
    b(1,1), b(2,1), b(3,1), b(1,2), b(2,2), b(3,2)
```

という 6 個の並んだメモリのことです．そこで，上のサブルーチン sub に，1 変数 y の代わりに 1 次元配列 a(10) を使って，

```
   call sub(x,a)
```

のように配列名でコールすると，サブルーチンの対応する引数 v にはその先頭アドレス（a(1) のアドレス）が与えられます．2 次元配列 b(3,2) を使って，

```
   call sub(x,b)
```

とコールしても，サブルーチンの対応する引数 v に与えられるのは，やはり先頭アドレス（b(1,1) のアドレス）です．よって，これらは，

```
   call sub(x,a(1))
   call sub(x,b(1,1))
```

と書いても全く同じ意味になります．

さて，図 2 のサブルーチンは使用するのが一つの変数だけだったのですが，これを連続したメモリ領域として利用する，つまり「配列」として利用したいときには，サブルーチン側で配列宣言をする必要があります．例えば，図 2 のサブルーチンの宣言を

```
subroutine sub(u,v)
   real u,v(10)         !...........(1)
```

と修正すれば，第 2 引数は配列として計算に使用することができます．ただし，「情報は先頭アドレスだけ」というのは変わりません．このため，この例では v(10) と宣言していますが，10 にはあまり意味はなく，1 でも良いし，1000 でも同じです．このため，p.39 に書いてあるように，*と書くこともできるし，整合配列にすることもできます．ただし，2 次元配列として利用する場合，

```
subroutine sub(u,v)
   real u,v(3,2)        !...........(2)
```

と書けますが，このとき，左の第 1 添字 3 には意味があります．これは，上記のように先頭アドレスからの順番を決めるときに右の添字を増加するタイミング情報が必要だからです．このため，第 1 添字は*にすることができません（p.39）．

さて，情報をもらったアドレスを先頭だと解釈するのですから，引数を 1 次元配列と解釈するサブルーチン (1) の場合，

```
   call sub(x,a(3))
```

のように，コールすれば，

```
    a(3) → v(1), a(4) → v(2), a(5) → v(3), ....
```

のようにずれた対応になります．2 次元配列を与えても同じで，

```
   call sub(x,b(3,1))
```

というコールでは，

```
    b(3,1) → v(1), b(1,2) → v(2), b(2,2) → v(3), b(3,2) → v(4)
```

という対応になります．

ここまで来ると，本書 3.1 の解答例の問題がおわかりいただけるのではないでしょうか．解答例では，2 次元配列 x(2,2) を使っているので，メモリの並びは，

```
    x(1,1), x(2,1), x(1,2), x(2,2)
```

です．よって，(x(1,1),x(2,1)) のペアに固有ベクトルの成分を代入したければ，その先頭アドレス，x(1,1) を引数に与え，(x(1,2),x(2,2)) のペアに固有ベクトルの成分を代入したければ，その先頭アドレス，x(1,2) を引数に与えれば良いことになります．x(1,1) のところは，x だけでも良いのですが，形をそろえるために x(1,1) にしています．この形なら do 文で書き直すこともできます．

このあたり，Fortran のわかりにくい部分ではありますが，引数が先頭メモリアドレスの情報のみ渡している，ということが理解できれば，それほど不思議ではないと思います[1]．

なお，最近は第 7 章に書いてある配列計算式が使えるようになったので，

```
  call check_eigenvalue(a,eigen(1),x(:,1),2)
  call check_eigenvalue(a,eigen(2),x(:,2),2)
```

のように，部分配列で記述した方が 1 次元配列を与えている感じがするので，良いかもしれません．部分配列を使う場合には，元の 2 次元配列の並びを考慮する必要がないので，

```
  call check_eigenvalue(a,eigen(1),x(1,:),2)
  call check_eigenvalue(a,eigen(2),x(2,:),2)
```

などと書くこともできます．この場合は，1 行目は (x(1,1),x(1,2)) のペアに固有ベクトルの成分を代入し，2 行目は (x(2,1),x(2,2)) のペアに固有ベクトル成分を代入します．なお，部分配列を使う方法は便利なのですが，アドレスを渡す前に，配列の 1 部を取り出して，別の配列に代入してからサブルーチンに渡す，というような中間処理を伴う可能性があるので，若干時間がかかったり余分なメモリが必要になることがあります．

このように，Fortran におけるサブルーチンの引数というのは，先頭アドレス情報しか伝わらないので，対応を間違うと思わぬエラーを引き起こす可能性があります．そこで，本書では説明しませんでしたが，引数をもう少し厳密にチェックする方法も用意されています．この機能を使えば，配列 a で指定するのと，配列要素 a(1) で指定するのは意味が異なることになるので，より安全なプログラムにすることができます．そのような機能を使ってみたい場合には色々調べてもらえればと思います．

[1] 実のところ，私も学生のときに配列名 a を与えるのと，a(1) を与えるのは意味が違うと思ってたことがあります．