第2回レポート

- 反応拡散モデル (Wiener and Rosenblueth model) による状態の変化を表示するプログラム

- 提出締め切り: ７月 ８日(金)
- 修正締め切り: ７月１３日(水)

- レポート回答: ７月１４日(木)

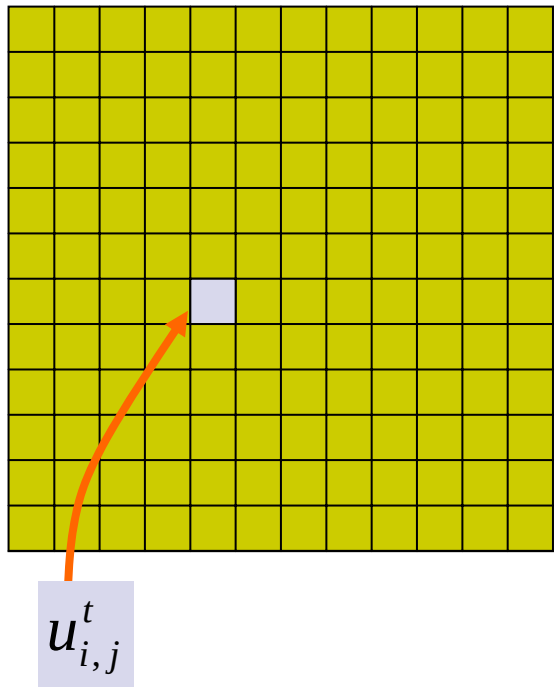
- 試験: ７月２１日(木) ２限目
  場所: 本館 １０号

Wiener and Rosenblueth model

- (0~N, 0~N) の2次元格子に並んだセルの状態の変化に関するモデル.

- 時刻 t における(i, j) 番目のセルの状態を $u_{i,j}^{t}$ とする.
  - $u_{i,j}^{t}$ は -1, 0, 1 のいずれかの値を取る.

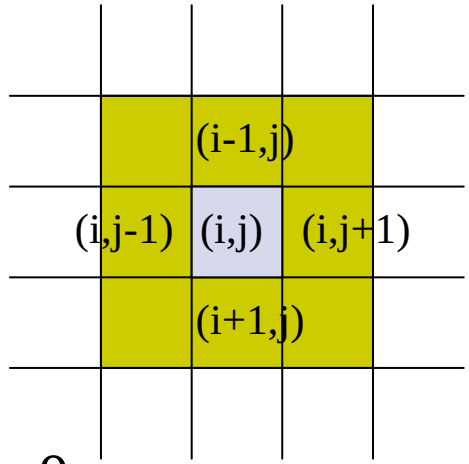
N=11 の場合



状態の変化に関する法則

- 次の時間 t+1 の状態は以下で決まる.

$$u_{i,j}^{t+1} := E(u_{i,j}^{t}) + D(u_{i,j}^{t}, u_{i-1,j}^{t}, u_{i,j-1}^{t}, u_{i+1,j}^{t}, u_{i,j+1}^{t})$$



- ここで

$$E(0) = 0, E(-1) = 0, E(1) = -1$$

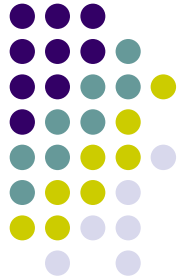
$$D = \begin{cases} \max\{0, u_{i-1,j}^{t}, u_{i,j-1}^{t}, u_{i+1,j}^{t}, u_{i,j+1}^{t}\}, & if \quad u_{i,j}^{t} = 0 \\ 0, & if \quad u_{i,j}^{t} \neq 0 \end{cases}$$

日本語で書くと...

- 平面に格子状に要素（セル）が並んでおり、それぞれ以下の状態のいずれかであるとする。
  - １：　発火（励起）状態
  - －１：　不応状態
  - ０：　鎮静状態

- 各要素は時間ステップごとに以下の法則で状態を変化させるとする。
  - 発火状態の場合：　不応状態となる
  - 不応状態の場合：　鎮静状態となる
  - 鎮静状態の場合：
    上下左右に発火状態の要素があれば発火状態となる。
    上下左右に発火状態の要素が無ければそのまま。

以下の条件で状態を変化させた時の様子を表示するプログラム

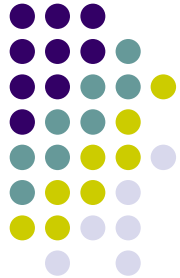


- 格子の形状：（0～N，0～N）
  - Nは実行時に入力する
  - 状態が変化するのは(1～N－1, 1～N－1)の範囲のみとする

- 初期状態：以下の要素をのぞいて全て鎮静状態（＝0）
  (N/2, N/2～N-1)の要素：発火状態（＝1）
  (N/2 + 1, N/2～N-1)の要素：不応状態（＝－1）

- 状態を変化させるステップ数：50

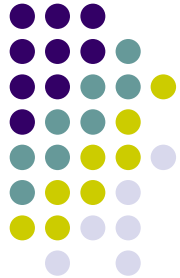
プログラムの条件



- ステップ毎に状態を表示する。
  - 次ページに紹介するサブルーチンを利用してよい。
  - 状態の変化を見やすくするため、
    ステップ毎に read(*,*) を使って進行を中断する。

- 全要素の状態を更新するサブルーチンを作成し、利用する。

状態を表示するサブルーチンの例



```
subroutine visualize(N,A)
  implicit none
  integer, intent(IN) :: N
  integer,dimension(0:N,0:N),intent(IN) :: A
  integer :: i,j

  do i=0,N
    do j=0,N
      if(A(i,j)== 1) write(*, '(A$)') "*"
      if(A(i,j)== 0) write(*, '(A$)') " "
      if(A(i,j)==-1) write(*, '(A$)') "+"
    end do
    write(*,*)
  end do
end subroutine
```

実行例

```
Enter size
20
initial
```

入力

```
**********
++++++++++
```

改行

```
1 -th
```

```
**********
*++++++++++
```

改行

```
2 -th
```

```
**********
 *++++++++++
  *+
   *
```

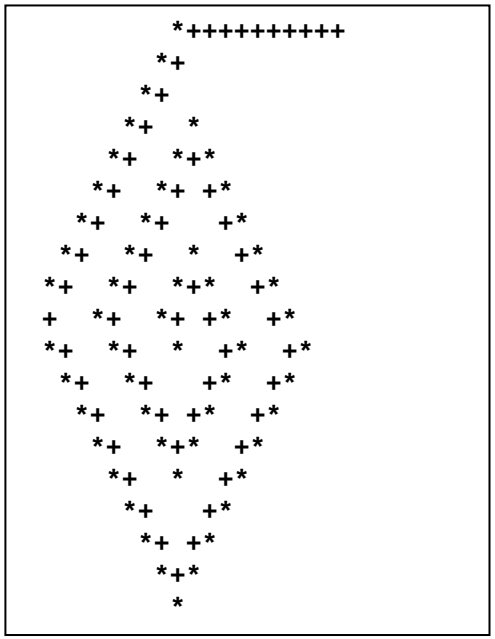
改行

```
3 -th
```

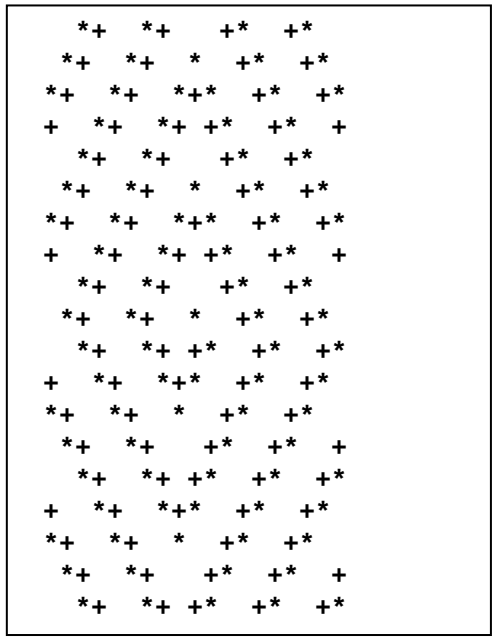
```
**********
 *++++++++++
  *+
   *+
    *+*
     *
```

N=20 の時の状態

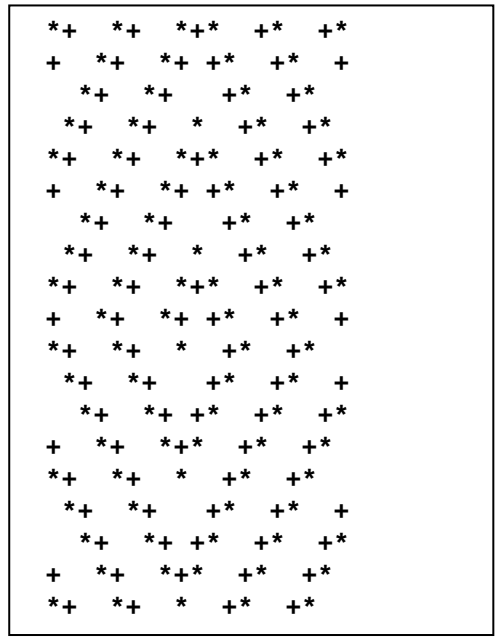
10ステップ目



20ステップ目



50ステップ目

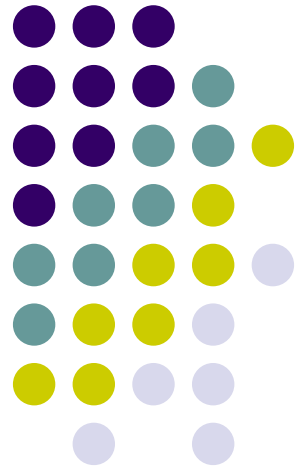


時間に余裕があれば

- 初期状態を変えてみる
  例)

  以下の要素以外は全て鎮静状態
  （N/2, N/2）： 発火状態
  （N/2 - 1, N/2）： 不応状態

第二回レポート　解答



今回の問題のミソ

- 単純に配列の各要素の状態を更新していくと正しい結果が得られない.

間違いの例）

```
do i=1, n-1
   do j=1, n-1
      if(a(i, j) == 1) then
         a(i, j) = -1
      else if(a(i, j) == -1) then
         a(i, j) = 0
      else if(a(i-1,j)==1 .or. a(i,j-1)==1 .or.    &
              a(i+1,j)==1 .or. a(i,j+1)==1) then
         a(i, j) = 1
      end if
   end do
end do
```

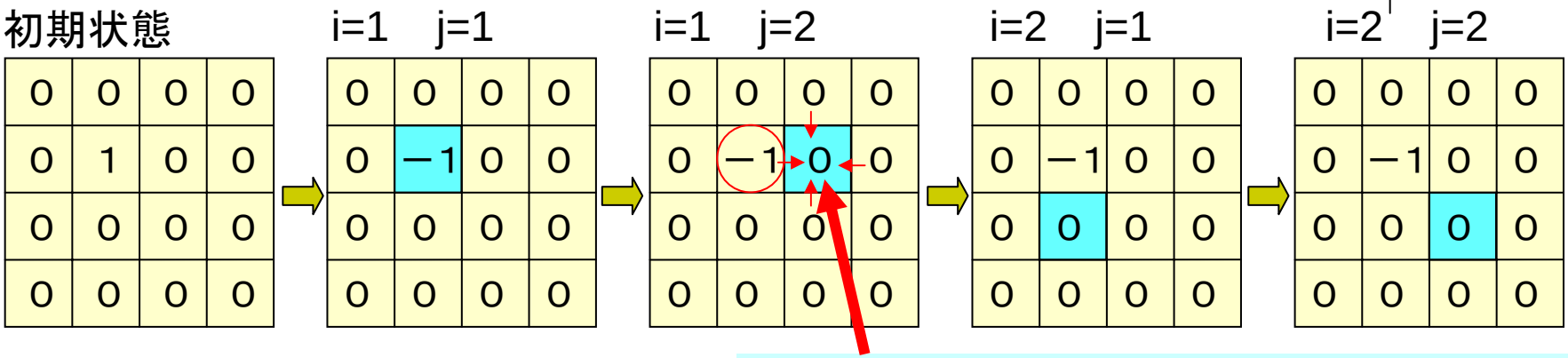
状態 1 →状態 -1

状態 -1 →状態 0

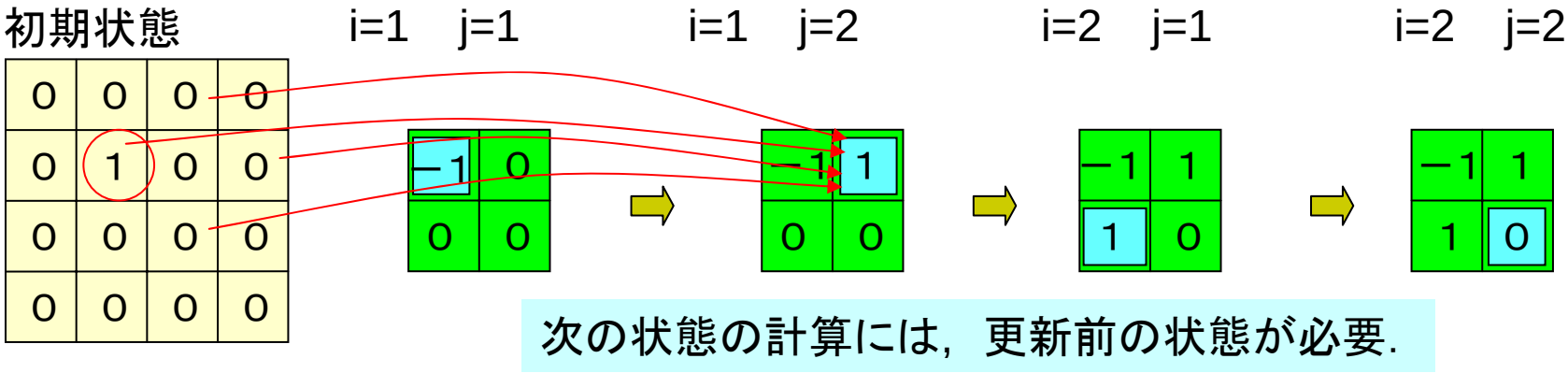
状態 0 かつ，四方に状態 1 の要素がある
→ 状態 1

何故か？

- 動作を追ってみると...



- 正しい動作



解決策の例1

- まず，状態 1 の要素と隣接している要素を調べておく.
  - 調べ終われば，配列を直接書き換え可能.
  - 状態 1 の要素と隣接している要素
    ＝ 四方の要素の最大値
    max（a(i-1,j), a(i,j-1), a(i+1,j), a(i,j+1)）
    が1である要素

第二回レポートの解答例1 (1/3)

```
program report2_1
  implicit none
  integer, parameter :: step = 50
  integer, dimension(:,:), allocatable :: a
  integer :: n,i

  write(*,*)  "Enter Size"
  read(*,*) n

  allocate(a(0:n, 0:n))
  a = 0
  a(n/2, n/2:n-1)= 1
  a(n/2+1, n/2:n-1)=-1

  write(*,*) 'Initial'
  call visualize(n, a)

  do i=1,step
     call change(n, a)
     read(*,*)
     write(*,*) i,'-th step'
     call visualize(n, a)
  end do
end program report2_1
```

aの内容を初期化

状態の更新

第二回レポートの解答例1 (2/3)

```
subroutine change(n, a)
  implicit none
  integer, intent(IN) :: n
  integer, dimension(0:n,0:n),intent(INOUT) :: a
  integer, dimension(n-1, n-1) :: actives
  integer :: i,j
  intrinsic max

  do i=1, n-1
     do j=1, n-1
        actives(i, j) = max(a(i-1, j), a(i, j-1), &
                            a(i+1, j), a(i, j+1))
     end do
  end do

  do i=1, n-1
     do j=1, n-1
        if(a(i, j) == 1) then
           a(i, j) = -1
        else if(a(i, j) == -1) then
           a(i, j) = 0
        else if(actives(i, j)  == 1) then
           a(i, j) = 1
```

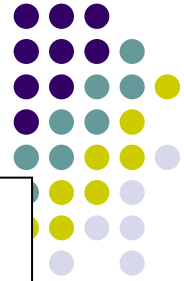
四方の要素の最大値を計算して配列 actives に格納しておく

状態 1 →状態 -1

状態 -1 →状態 0

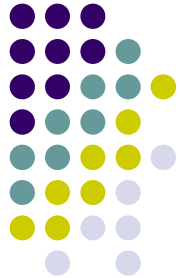
状態 0 で，四方に状態 1 の要素がある→状態 1

第二回レポートの解答例1 (3/3)



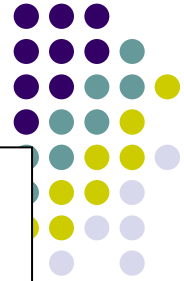
```
        end if
      end do
    end do
end subroutine change

subroutine visualize(n, a)
  implicit none
  integer, intent(IN) :: n
  integer, dimension(0:n, 0:n), intent(INOUT) :: a
  integer :: i,j

  do i=0, n
     do j=0, n
        if(a(i, j) == 1)  write(*, '(A$)') "*"
        if(a(i, j) == 0)  write(*, '(A$)') " "
        if(a(i, j) == -1) write(*, '(A$)') "+"
     end do
     write(*,*)
  end do
end subroutine visualize
```

解決策の例2



- 次の状態を一時的に別の配列に保存しておき，全要素の計算が終わったらコピーする．

第二回レポートの解答例2（一部）



```
subroutine change(n, a)
  implicit none
  integer, intent(IN) :: n
  integer, dimension(0:n,0:n),intent(INOUT) :: a
  integer, dimension(n-1, n-1) :: next
  integer :: i,j
  intrinsic max

  do i=1, n-1
     do j=1, n-1
        if(a(i, j) == 1) then
           next(i, j) = -1
        else if(a(i, j) == -1) then
           next(i, j) = 0
        else
           next(i, j) = max(0, a(i-1, j), a(i, j-1), &
                            a(i+1, j), a(i, j+1))
        end if
     end do
  end do

  a(1:n-1, 1:n-1) = next(1:n-1, 1:n-1)

end subroutine change
```

次の状態を一時的に格納する配列

状態 1 →状態 -1

状態 -1 →状態 1

状態 0 →状態 1 もしくは状態 0

新しい状態に更新

解答例1と2の比較

- 関数 max の呼び出し回数
  - 解答例1: $(N-1)\times(N-1)\times$ステップ数
    - 全要素について max を呼び出し
  - 解答例2: $\sum_{i=1}^{\text{ステップ数}}$ステップ$i$で状態が0である要素の数
    - 状態が0の場合のみ，max を呼び出し
- 実際には，それほど大きな差ではない．
  - 条件が複雑になった場合，影響が大きくなる可能性あり

面白かった解答1

- 数式に忠実なプログラム

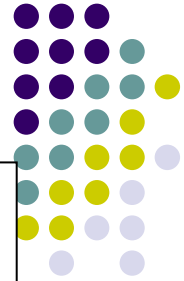
$$u_{i,j}^{t+1} := E(u_{i,j}^{t}) + D(u_{i,j}^{t}, u_{i-1,j}^{t}, u_{i,j-1}^{t}, u_{i+1,j}^{t}, u_{i,j+1}^{t})$$

$$E(0) = 0, E(-1) = 0, E(1) = -1$$

$$D = \begin{cases} \max\{0, u_{i-1,j}^{t}, u_{i,j-1}^{t}, u_{i+1,j}^{t}, u_{i,j+1}^{t}\}, \\ 0, \end{cases}$$

  - 全ステップの状態を記憶する
  - 関数Eと関数Dを定義し, 各ステップの計算に利用.

解答例3 (1/4)

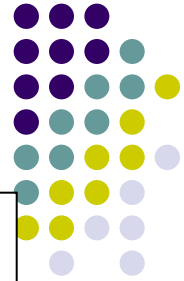


```
program WRmodel
  implicit none
  integer ::n,t,x,y
  integer,dimension(:,:,:),allocatable :: u
  integer,external :: E,D
  intrinsic dble

  write(*,*) 'Number:'
  read(*,*) n

  allocate(u(0:50,0:n,0:n))

  ! 初期状態の入力
  do x=0,n
     do y=0,n
        if(x==n/2.and.y>=n/2.and.y<=n-1) then
           u(0,x,y)=1
        else if(x==n/2+1.and.y>=n/2.and.y<=n-1) then
           u(0,x,y)=-1
        else
           u(0,x,y)=0
        end if
     end do
  end do
  write(*,*) 'initial'
  call visualize(0,n,u)
  read(*,*)
```

解答例3 (2/4)

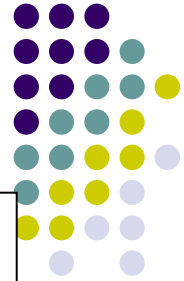
```
  !main program
  do t=0,49
     do x=1,n-1
        do y=1,n-1
           u(t+1,x,y)=E(t,x,y,n,u)+D(t,x,y,n,u)
        end do
     end do
     write(*,*) t+1,'-th'
     call visualize(t+1,n,u)
     read(*,*)
  end do

  stop
end program WRmodel

function E(T,i,j,N,A)
  implicit none
  integer,dimension(0:50,0:N,0:N) :: A
  integer :: E,T,N,i,j

  if(A(T,i,j)==0)   E=0
  if(A(T,i,j)==-1)  E=0
  if(A(T,i,j)==1)   E=-1

end function E
```

解答例3 (3/4)



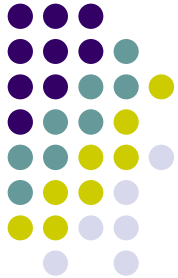
```
function D(T,i,j,N,A)
  implicit none
  integer :: D,N,T,i,j
  integer,dimension(0:50,0:N,0:N) :: A

  if(A(T,i,j)==0) then
     D=max(0,A(T,i-1,j),A(T,i,j-1),A(T,i+1,j),A(T,i,j+1))
  else
     D=0
  end if

end function D
```

解答例3 (4/4)



```
subroutine visualize(T,N,A)
  implicit none
  integer,intent(IN) :: N,T
  integer,dimension(0:50,0:N,0:N),intent(IN) :: A
  integer :: i,j

  do i=0,N
     do j=0,N
        if(A(T,i,j)==1) write(*,'(A$)') "*"
        if(A(T,i,j)==0) write(*,'(A$)') " "
        if(A(T,i,j)==-1) write(*,'(A$)') "+"
     end do
     write(*,*)
  end do
end subroutine visualize
```

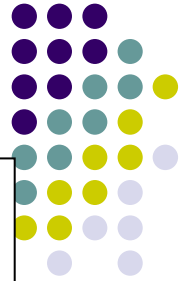
面白かった解答2



- 一つの配列だけで更新するプログラム
  - 既に更新した a(i-1, j) と a(i, j-1) を参照する際, 符号を反転させる.
    - 状態が 1だった要素は, 必ず -1に変化している
    - 状態が 0だった要素は, 0か 1に変化している
    - 状態が -1だった要素は, 必ず 0に変化している.

符号を反転して 1 になるのは, 更新前の状態が 1だった要素だけ

解答例4 (一部)



```
subroutine change(N,A)
  implicit none
  integer, intent(IN) :: N
  integer,dimension(0:N,0:N),intent(INOUT) :: A
  integer :: b, c, d, e, f, i, j
  intrinsic max

  do i=1,N-1
     do j=1,N-1
        if ( A(i,j)==0 ) then
           b = -A(i-1, j)
           c = -A(i, j-1)
           d = A(i+1, j)
           e = A(i, j+1)
           f = max(0, b, c, d, e)
           A(i,j) = f
        else if ( A(i,j)==-1 ) then
           A(i,j) = 0
        else if ( A(i,j)==1 ) then
           A(i,j) = -1
        end if
     end do
  end do
end subroutine change
```

細かい注意点

- 変数（配列）の初期値について
  - 変数に最初に値を代入するまで，初期値は不定．
    ＝　何が入っているか分からない．

例）

```
program test
  implicit none
  integer :: i

  write(*, *) i

stop
end program
```

何が表示されるか分からない

初期値の設定

- 特に配列の初期値
  - 必ず全要素に初期値を設定しておく

```
program report2_1
  implicit none
  integer, parameter :: step = 50
  integer, dimension(:,:), allocatable :: a
  integer :: n,i

  write(*,*)  "Enter Size"
  read(*,*) n

  allocate(a(0:n, 0:n))
  a = 0
  a(n/2, n/2:n-1)= 1
  a(n/2+1, n/2:n-1)=-1
```

aの内容を初期化

必ず初期値を代入しておく

```
program report2_1
  implicit none
  integer, parameter :: step = 50
  integer, dimension(:,:), allocatable :: a
  integer :: n,i

  write(*,*)  "Enter Size"
  read(*,*) n

  allocate(a(0:n, 0:n))
  a = 0
  a(n/2, n/2:n-1)= 1
  a(n/2+1, n/2:n-1)=-1
```