Intermediate programming

— Pointer to a matrix —

Waseda Univ.

Today's topic

- Representation of a matrix and vector by using arrays
- Calculation of a matrix and vector
- Arrays and pointers
- Dynamic allocation of matrixes and vectors
- Matrix vector multiplication with a pointer

Matrix and vector representations using an array

Vectors and matrixes can be expressed using an array.

- Vector: Using one-dimensional array

```
double x[N];    /*Allocate N elements x[0]~x[N-1]*/
```

$$x[N] = \left(x[0], \quad x[1], \quad \cdots \quad , x[N-1]\right)$$

- Matrix : Using two-dimensional array

```
double a[N][N];/*Allocate N*N elements a[0][0]~a[N-1][N-1]*/
```

$$a[N][N] = \begin{pmatrix} a[0][0] & a[0][1] & \cdots & a[0][N-1] \\ a[1][0] & a[1][1] & \cdots & a[1][N-1] \\ \vdots & \vdots & \ddots & \vdots \\ a[N-1][0] & a[N-1][1] & \cdots & a[N-1][N-1] \end{pmatrix}$$

Calculation of matrix and vector

Matrix vector multiplication

The program to compute matrix vector multiplication $\vec{y} = A\vec{x}$, where A is a $N \times N$ matrix and vector $\vec{x}$ is a N vector.

- Representing a matrix-vector multiplication for each element of $\vec{y} = A\vec{x}$:

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{pmatrix} = \begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N-1,0} & a_{N-1,1} & \cdots & a_{N-1,N-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{pmatrix}$$

- It is possible to write a certain element $y_i (i = 0, \ldots, N-1)$ of vector y as

$$y_i = a_{i,0}x_0 + a_{i,1}x_1 + \ldots + a_{i,N-1}x_{N-1} = \sum_{k=0}^{N-1} a_{i,k}x_k$$

Calculation of matrix and vector

- For obtaining the element of vector $\vec{y}$,

```
y[i]=0.0;
for(k=0; k<N; k++){
    y[i] += a[i][k]*x[k];
}
```

- For obtaining $\vec{y} = A\vec{x}$,

```
for(i=0; i<N; i++){
  y[i]=0.0;
  for(k=0; k<N; k++){
      y[i] += a[i][k]*x[k];
  }
}
```

- Don't forget to initialize vector $\vec{y}$ like $y[i] = 0.0$.

Calculation of matrix

Matrix multiplication

The program to compute matrix multiplication $Y = AB$, where A is $N \times L$ matrix A and B is $L \times M$ matrix.

- Representing a matrix multiplication for each element:

$$\begin{pmatrix} y_{0,0} & \cdots & y_{0,M-1} \\ \vdots & \ddots & \vdots \\ y_{N-1,0} & \cdots & y_{N-1,M-1} \end{pmatrix} = \begin{pmatrix} a_{0,0} & \cdots & a_{0,L-1} \\ \vdots & \ddots & \vdots \\ a_{N-1,0} & \cdots & a_{N-1,L-1} \end{pmatrix} \begin{pmatrix} b_{0,0} & \cdots & b_{0,M-1} \\ \vdots & \ddots & \vdots \\ b_{L-1,0} & \cdots & b_{L-1,M-1} \end{pmatrix}$$

- It is possible to write a certain element $y_{i,j} (i = 0, \ldots, N-1; j = 0, \ldots, M-1)$ of matrix Y as

$$y_{i,j} = a_{i,0}b_{0,j} + a_{i,1}b_{1,j} + \ldots + a_{i,L-1}b_{L-1,j} = \sum_{k=0}^{L-1} a_{i,k}b_{k,j}$$

Calculation of matrix

- For computing a certain element of matrix Y ($y[i][j]$), multiply $a_{i,k}b_{k,j}$, then add:

```
y[i][j]=0.0;
for(k=0; k<L; k++){
  y[i][j] += a[i][k]*b[k][j];
}
```

- For obtaining matrix AB,

```
for(i=0; i<N; i++){
  for(j=0; j<M; j++){
    y[i][j]=0.0;
    for(k=0; k<L; k++){
      y[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

- Similarly, you should initialize all elements of Y as 0.0.

Example 1

Example 1

Write the program to compute $\vec{y} = A\vec{x}$.

$$A = \begin{pmatrix} 3.0 & 2.0 & 5.0 \\ 1.0 & 4.0 & 3.0 \\ 0.0 & 1.0 & 6.0 \end{pmatrix}, \quad \vec{x} = \begin{pmatrix} -3.0 \\ 2.0 \\ -1.0 \end{pmatrix}$$

Answer

```
#include <stdio.h>
#define N 3
int main(void) {
  int i, j;
  double A[N][N] = {{3.0, 2.0,5.0},{1.0, 4.0, 3.0},{0.0, 1.0, 6.0}};
  double x[N]= {-3.0, 2.0, -1.0};
  double y[N];  /* Result of calculation */
  printf("A = \n"); /* Display of matrix A*/
  for (i = 0; i < N; i++) { /* i-th row */
    for (j = 0; j < N; j++) { /* j-th column */
      printf("%6.1f\t", A[i][j]);
    }
    printf("\n"); /* Line break for display the next row*/
  }
  printf("\n");
  printf("x=\n");
  for (i = 0; i < N; j++) printf("%6.1f\n", x[i]);
  printf("\n");
  /* Computing y = Ax*/
  for (i = 0; i < N; i++) {
    y[i] = 0; /* Initialization */
    for (j = 0; j < N; j++) {
      y[i] += A[i][j] * x[j];
    }
  }
  printf("\n");
  printf("y = \n"); /* Display of y */
  for (i = 0; i < N; j++) printf("%6.1f\n", y[i]);
  printf("\n");
  return 0;
}
```

About example 1

About example1

Write the program to compute $\vec{y} = A\vec{x}$.

$$A = \begin{pmatrix} 3.0 & 2.0 & 5.0 \\ 1.0 & 4.0 & 3.0 \\ 0.0 & 1.0 & 6.0 \end{pmatrix}, \quad \vec{x} = \begin{pmatrix} -3.0 \\ 2.0 \\ -1.0 \end{pmatrix}$$

- If we use the array for matrix and vector, we need to modify the program whenever the problem size is changed.
  $\Rightarrow$Dynamic allocation using malloc function
  $\Rightarrow$ Handling vectors and matrices using pointer.
- We are going to study the relationship between arrays and pointers.

Array and pointer

For the representation of vectors, we use the array.

double x[N]; Allocating n-th double data space in the memory continuously.

Address		0xbfff660	0xbfff668	0xbfff676	$\cdots$		
Contents	$\cdots$	x[0]	x[1]	x[2]	$\cdots$	x[N-1]	$\cdots$

- Using an address operator (&) (ampersand operator), we can refer to the address of each element:

  `&x[0], &x[1], &x[2], ···, &x[N-1]`

- (For double data type, 8 bytes are required to store each element)

Array and pointer

- The address of element $x[i]$ is referred by using $\&x[i]$.

Address		0xbfff660 =&x[0]	0xbfff668 =&x[1]	0xbfff676 =&x[2]	$\cdots$	&x[N-1]	
Contents	$\cdots$	x[0]	x[1]	x[2]	$\cdots$	x[N-1]	$\cdots$

- The name of array x can be used with a pointer as the first address of array.
- Using pointer operation, we can refer to an address of each element $x + i$

Address		0xbfff660 =x	0xbfff668 =x+1	0xbfff676 =x+2	$\cdots$	x+(N-1)	
Contens	$\cdots$	x[0]	x[1]	x[2]	$\cdots$	x[N-1]	$\cdots$

Address

$$x + i \quad = \quad \&x[i] \qquad (i = 0, 1, \cdots, N-1)$$

Array and pointer

- For pointers using the dereference operator, we can refer to the value that is pointed to by the address.
- For pointer $x + i$, we can can refer to the value using $*(x + i)$.

Address		x ↓	x+1 ↓	x+2 ↓		x+(N-1) ↓	
Contents	$\cdots$	x[0]	x[1]	x[2]	$\cdots$	x[N-1]	$\cdots$
Reference of value		↑ *x	↑ *(x+1)	↑ *(x+2)		↑ *(x+(N-1))	

How to refer

$$x[i] \quad \leftrightarrow \quad *(x+i)(i = 0, 1, 2, \cdots, N-1)$$

Passing the array using its address

- To pass the array, the receive side of the pointer is used.

```
/*Function to initialize an array*/
void setZero( int *x, int size){
    int i;
    for(i=0; i<size; i++)
        x[i]=0;
}
int main(void){
    int x[10];
    setZero(x,10);          /*Note that not &x*/
```

- In function setZero, $x[i] = 0$ may be used as $*(x + i) = 0$.

Pointer to a matrix

- We can express matrices on the program using a 2D-array.
- For example, for $N \times N$ matrices, when we declare the array as double a[N][N], it is stored in the memory.

Matrix A:

A[0][0]	A[0][1]	A[0][2]	$\cdots$	A[0][N-1]
A[1][0]	A[1][1]	A[1][2]	$\cdots$	A[1][N-1]
A[2][0]	A[2][1]	A[2][2]	$\cdots$	A[2][N-1]
$\vdots$	$\vdots$	$\vdots$		$\vdots$
A[N-1][0]	A[N-1][1]	A[N-1][2]	$\cdots$	A[N-1][N-1]

Pointer to a matrix

Declaring a matrix using array:

double A[N][N];

In this case, the data space for $N \times N$ elements of A is stored continuously on the memory like the right side.
Here, array name A is a pointer that is the first address of array

$$A[0], A[1], \cdots, A[N-1]$$

$(A = \&A[0])$

Each $A[i](i = 0, 1, \cdots, N-1)$ is a pointer that points the first elements of the i-th row.

Pointer	Contents
A[0]	A[0][0]
	A[0][1]
	⋮
	A[0][N-1]
A[1]	A[1][0]
	A[1][1]
	⋮
	A[1][N-1]
	⋮
A[N-1]	A[N-1][0]
	⋮
	A[N-1][N-1]
	⋮

Pointer to a matrix

double A[N][N];

For referring the element ($A[i][j]$) in the i-th row and the j-th column of matrix A, we can write it as

$$*(A[i]+j)$$

$$A[i][j] \Leftrightarrow *(A[i]+j)$$

	Contents
*(A[0]) →	A[0][0]
*(A[0]+1) →	A[0][1]
	⋮
*(A[0]+N-1) →	A[0][N-1]
	⋮
*(A[i]+0) →	A[i][0]
*(A[i]+1) →	A[i][1]
	⋮
*(A[i]+j) →	A[i][j]
	⋮

Pointer to a matrix

For

double A[N][N];

we introduce a pointer value ai

double $*$ ai;

Using $A[0]$ that points a pointer to the first element of $A(A[0][0])$, it stores

$ai = A[0];$

we can refer to the elements in the 0th row using

$*ai, *(ai+1), \cdots, *(ai+N-1)$

In this case:

$ai[j] \Leftrightarrow *(ai+j)$

	Contents	Pointer
*ai →	A[0][0]	ai
*(ai+1) →	A[0][1]	ai+1
	⋮	
*(ai+N-1) →	A[0][N-1]	ai+N-1
	⋮	
*(a[i]+0) →	A[i][0]	
*(a[i]+1) →	A[i][1]	
	⋮	
*(a[i]+j) →	A[i][j]	
	⋮	

Pointer to a matrix

For moving ai to the next row, to be able to write it as

$$ai+ = N;$$

ai always means

"A pointer to the first address $A[i][0]$ in the i-th row".

We can refer to the elements in the i-th row by

$$ai[0], ai[1], \cdots, ai[N-1]$$

	Contents	Pointer
	A[0][0]	ai
	A[0][1]	ai+1
	⋮	
	A[0][N-1]	ai+N-1
ai[0] →	A[1][0]	ai
ai[1] →	A[1][1]	ai+1
	⋮	
ai[N-1] →	A[1][N-1]	ai+N-1
ai[0] →	A[2][0]	ai
	⋮	

Pointer to a matrix

For moving ai to the next row, to be able to write it as

$$ai+ = N;$$

ai always means

"A pointer to the first address $A[i][0]$ in the i-th row".

We can refer to the elements in the i-th row by

$$ai[0], ai[1], \cdots, ai[N-1]$$

	Contents	Pointer
	A[0][0]	ai
	A[0][1]	ai+1
	⋮	
	A[0][N-1]	ai+N-1
ai[0] →	A[1][0]	ai
ai[1] →	A[1][1]	ai+1
	⋮	
ai[N-1] →	A[1][N-1]	ai+N-1
ai[0] →	A[2][0]	ai
	⋮	

Pointer to a matrix

For moving ai to the next row, to be able to write it as

$$ai+ = N;$$

ai always means

"A pointer to the first address $A[i][0]$ in the i-th row".

We can refer to the elements in the i-th row by

$$ai[0], ai[1], \cdots, ai[N-1]$$

	Contents	Pointer
	A[0][0]	ai
	A[0][1]	ai+1
	⋮	
	A[0][N-1]	ai+N-1
ai[0] →	A[1][0]	ai
ai[1] →	A[1][1]	ai+1
	⋮	
ai[N-1] →	A[1][N-1]	ai+N-1
ai[0] →	A[2][0]	ai
	⋮	

Example 2

Example 2

Rewrite the program to compute $\vec{y} = A\vec{x}$ using a pointer expression.

$$A = \begin{pmatrix} 3.0 & 2.0 & 5.0 \\ 1.0 & 4.0 & 3.0 \\ 0.0 & 1.0 & 6.0 \end{pmatrix}, \quad \vec{x} = \begin{pmatrix} -3.0 \\ 2.0 \\ -1.0 \end{pmatrix}$$

- Like the following, rewrite the computational part using pointer expression

2 dimensional array:

```
for(i=0;i<N;i++){
    y[i] = 0.0;
    for(j=0;j<N;j++){
        y[i] += A[i][j]*x[j];
    }
}
```

Pointer expression:

```
ai=A[0];
for(i=0;i<N;i++){
    computational part
}
```

Hint

- Declaring arrays as in the past and writing the calculation part using a pointer expression:

```
double x[N];
double A[N][N];
double *ai;
```

- Each element of vector x

```
x[i]     (i=0,1,2,⋯,N-1)
```

- Each element of matrix A: ai is the pointer to the first address in the i-th row:

```
ai[0], ai[1], ⋯,ai[j],⋯,ai[N-1]
```

- For moving ai to the next row (the i+1'th row),

```
ai +=N;
```

Answer

```
#include <stdio.h>

#define N 3
void PrintVector(double *y, int n){
int i;
for(i=0;i<n;i++){
printf("%6.1f\n",*(y+i));
}
}
int main(void) {
  int i, j;
  double *ai;
  double A[N][N] =
   {{3.0, 2.0,5.0},{1.0, 4.0, 3.0},{0.0, 1.0, 6.0}};
  double x[N]= {-3.0, 2.0, -1.0};
  double y[N];
  printf("A = \n");
  ai = A[0];
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      printf("%6.1f", *(ai+j));
    }
    ai+=N;
    printf("\n");
  }
  printf("\n");
  printf("x=\n");
  PrintVector(x,N);

  ai = A[0];
  for (i = 0; i < N; i++) {
     *(y+i) = 0;
    for (j = 0; j < N; j++) {
    *(y+i) += *(ai+j) * *(x+j);
     }
     ai+=N;
  }
  printf("\n");
  printf("y = \n");
  PrintVector(y,2);
  printf("\n");

  return 0;
}
```

Calculation of matrix and vector

In the previous program, we declare arrays as

```
double A[3][3], x[3];
```

with the size of these. However,

- depending on the size of declared array,
- it needs to modify the program whenever the size or its contents are changed.

Therefore, we shall change a program that does not depend on it.

- For changing the contents of matrix and vector, here, we will input elements of matrix and vector by keyboard using scanf( ) function.
- For changing the size of problems,
  we will use malloc function for allocating the memory dynamically. ⇒ When we run the program, we can allocate the memory that is needed for problems.

Calculation of matrix and vector

In the previous program, we declare arrays as

```
double A[3][3], x[3];
```

with the size of these. However,

- depending on the size of declared array,
- it needs to modify the program whenever the size or its contents are changed.

Therefore, we shall change a program that does not depend on it.

- For changing the contents of matrix and vector, here, we will input elements of matrix and vector by keyboard using scanf( ) function.
- For changing the size of problems,
  we will use malloc function for allocating the memory dynamically. $\Rightarrow$ When we run the program, we can allocate the memory that is needed for problems.

Dynamic allocation

Vector

```
double *x;
x= (double *)malloc(N*sizeof(double));
```

$\Rightarrow$ We can handle the pointer x similar to the array that have n-th elements.

- x+i: pointer to the i-th element
- *(x+i): element of the i-th (similar to $x[i]$)

Example

```
scanf("%lf",&x[i]);  /*Input*/
printf("%.2f\n",x[i]); /*Display*/
```

Dynamic allocation

Matrix

```
double *A, *ai;
A = (double *)malloc(N*N*sizeof(double));
```

$\Rightarrow$ We can handle the pointer A similar to the array that have $N \times N$ elements.
Introducing to a pointer value ai, let it be $ai = A$;, it is possible to refer to the each element as $ai[j]$ in the 0th row, and for moving ai to the next row, we add like

$$ai+ = N;$$

ai always works as a first pointer in the i-th row.

Example 3

Example 3

Write the program to compute matrix vector multiplication $\vec{y} = A\vec{x}$.

- Use malloc function for allocating a matrix A and vector $\vec{x}$.
- Input the elements of these by keyboard.

$$A = \begin{pmatrix} 3.0 & 2.0 & 5.0 \\ 1.0 & 4.0 & 3.0 \\ 0.0 & 1.0 & 6.0 \end{pmatrix}, \quad \vec{x} = \begin{pmatrix} -3.0 \\ 2.0 \\ -1.0 \end{pmatrix}$$

Hint

- Don't forget to include stdlib.h.
- Release the allocated memory using free function.
- Example for vector

```
int N;
double *x;
printf("Input N:");
scanf("%d",&N);
x= (double *) malloc(N*sizeof(double));
if(x==NULL){
    printf("Can't allocate memory. \n");
    exit(1);
}

/*Input the elements by keyboard*/
for(i=0; i<N;i++){
   printf("x[%d]=",i);
   scanf("%lf",&x[i]);
}
...
free(x);
```

Hint

- Display of a vector

```
void PrintVector(double *y, int n){
   int i;
   for(i=0;i<n;i++){
      printf("%6.1f\n",*(y+i));
   }
}
```

- Display of a matrix

```
void PrintMatrix(double *A, int n){
   int i,j;
   double *ai;
   ai= A;
   for( i = 0 ; i < n ; i++){
     for( j = 0 ; j < n ; j++){
       printf("%6.1f", *(ai+j));
     }
     ai+=n;
     printf("\n");
   }
   printf("\n");
}
```

- Computational part

```
  ai = A;
  for (i = 0; i < N; i++) {
     *(y+i) = 0; /* Initialization */
     for (j = 0; j < N; j++) {
       *(y+i) += *(ai+j) * *(x+j);
     }
     ai+=N;
  }
```

```c
#include <stdio.h>
#include <stdlib.h>
void PrintVector(double *y, int n){/*Print function of a vector*/}
void PrintMatrix(double *A, int n){/*Print function of a matrix*/}
int main(void) {
  int i, j, N;
  double *ai;
  double *A, *x, *y;
  printf("N=");
  scanf("%d",&N);

  /*Allocating memory*/

  /*行列・ベクトルの入力*/
  ai = A;
  for( i = 0 ; i < N ; i++ ){
     for( j = 0 ; j < N ; j++ ){
        printf("A[%d][%d] = ", i , j ); scanf("%lf", ai+j );
     }
     ai+=N;
  }
  for( i = 0 ; i < N ; i++ ){
     printf("x[%d] = ", i ); scanf("%lf", x+i);
  }
  printf("A = \n"); PrintMatrix(A,N);
  printf("x = \n"); PrintVector(x,N);

  /* Computation */

  printf("y = \n");
  PrintVector(y,N);

  /*free function*/
  return 0;
}
```

Today's summary

- Representation of a matrix and vector by using arrays
- Calculation of a matrix and vector
- Arrays and pointers
- Dynamic allocation of matrixes and vectors
- Matrix vector multiplication with a pointer