Array and LAPACK

Stat 580

References

- Part of this slide set is based on *Essential C* by Nick Parlante:

  (for copyright reason, this notice is reproduced here.)

Multivariate normal distribution

- Let $\boldsymbol{\mu} \in \mathbb{R}^d$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$ be nonnegative definite.
- Recall for $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\boldsymbol{A} \in \mathbb{R}^{n \times d}$,

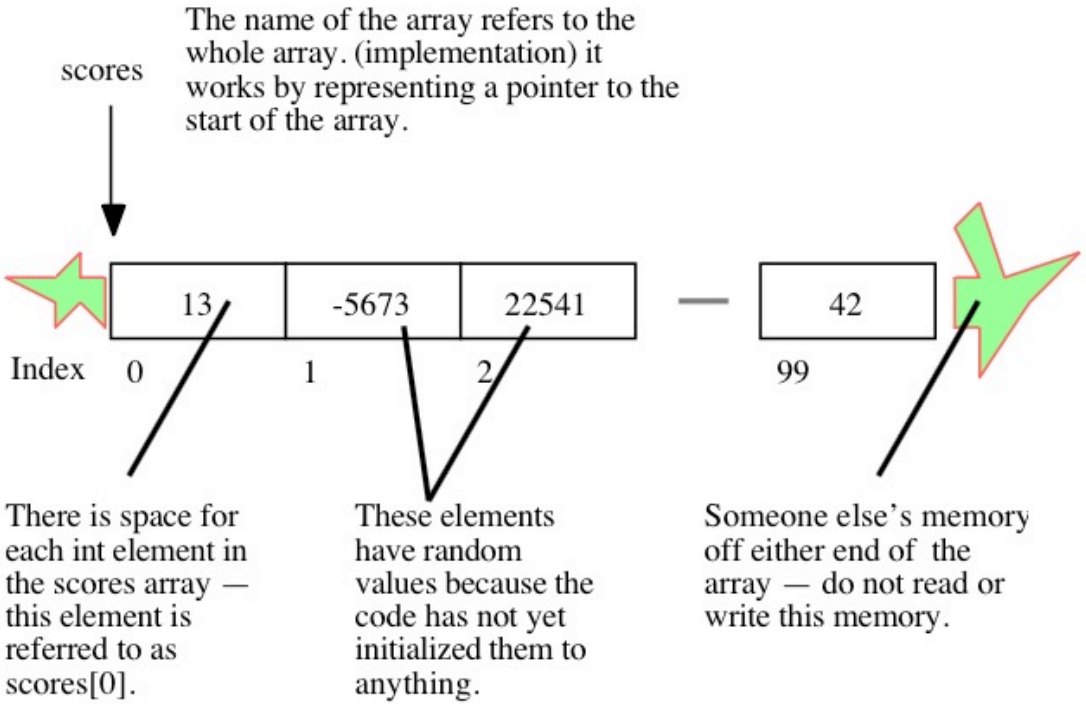
$$\boldsymbol{A}\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{A}\boldsymbol{\mu}, \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^T).$$

- To sample $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$:
  1. first compute the Cholesky decomposition of $\boldsymbol{\Sigma}$: $\boldsymbol{\Sigma} = \boldsymbol{A}\boldsymbol{A}^T$ where $\boldsymbol{A}$ is lower triangular ($\boldsymbol{A}$ is sometimes called the square root of $\boldsymbol{\Sigma}$)
  2. set $\boldsymbol{X} = \boldsymbol{\mu} + \boldsymbol{A}\boldsymbol{Z}$ where coordinates of $\boldsymbol{Z}$ are iid $\mathcal{N}(0, 1)$.
- we need arrays and some linear algebra! How to generate normal random vector in C?

Arrays

The simplest type of array in C is one which is declared and used in one place.

```
int scores[100];  /* declares an array called scores to hold 100 integers */
scores[0]  = 13;  /* set first element; C arrays are indexed from 0! */
scores[99] = 42;  /* set last element */
```



Arrays

- C does not do any run time or compile time bounds checking in arrays.
- At run time the code will just access or mangle whatever memory it happens to hit and crash or misbehave in some unpredictable way thereafter.

Example (other ways of declarations)

```c
int a[2];
int b[2]={1,2}; /* initialization */
int c[]={1,2};  /* initialization */

printf("a: (%d, %d)\n", a[0], a[1]); /* arbitrary values */
printf("b: (%d, %d)\n", b[0], b[1]); /* b: (1, 2) */
printf("c: (%d, %d)\n", c[0], c[1]); /* c: (1, 2) */
```

Multidimensional arrays

- You can declare multidimensional arrays in C
  - e.g., `int b[3][2];`
- The implementation of the array stores all the elements in a single contiguous block of memory.
- In memory, the array is arranged with the elements of the rightmost index next to each other. In other words, `X[1][8]` comes right before `X[1][9]` in memory.

Example

```
int b[3][2];
int c[3][2]={{1,2},{3,4},{5,6}}; /* c[][2], but not c[3][] */
int i, j;


/* print the matrix */
/* run the program a few times, value changes */
printf("b:\n");
for (i=0; i<3; i++) {
  for (j=0; j<2; j++) {
    printf("%d ", b[i][j]);
  }
  printf("\n");
}

/* print the matrix and the entry-wise memory address */
printf("c:\n");
for (i=0; i<3; i++) {
  for (j=0; j<2; j++) {
    printf("%d (%p) ", c[i][j], &(c[i][j]));
  }
  printf("\n");
}
```

Example

```c
double m[3][3] = { {3, 1, 3}, {1, 5, 9}, {2, 6, 5} };
double n[3][3] = { {1, 1, 1}, {2, 5, 1}, {2, 2, 5} };
int i, j, k;
double temp, res[3][3];

/* matrix mulitplication: m %*% n */
for (i=0; i<3; i++){
  for (j=0; j<3; j++){
    temp = 0;
    for (k=0; k<3; k++)
      temp += m[i][k] * n[k][j];
    res[i][j] = temp;
  }
}

/* print the product */
for (i=0; i<3; i++){
  for (j=0; j<3; j++){
    printf("%f ", res[i][j]);
  }
  printf("\n");
}
printf("\n");
```

String

- Strings operate as ordinary arrays of characters
- A C string is just
  - an array of char
  - with a "null" character ('\0') stored at the end to mark the end of the string
- Their maintenance is up to the programmer using the standard facilities available for arrays and pointers
  - regular assignment operator = does not do string copying! (`strcpy()`)

Example

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


int main() {
  char greetings[10]="hello"; /* actual length of hello is 6; 'h', 'e', 'l',*/
                              /* 'l', 'o', '\0' (end-of-string character)   */
  char greetings2[]="hello";
  char name[10];

  printf("%c\n", greetings[3]); /* l */
  printf("%lu\n", strlen(greetings)); /* strlen returns the length of the  */
                                      /* string (does not count '\0') 5     */
  printf("%lu\n", strlen(greetings2)); /* 5 */

  /* name = "Ray" */ /* incorrect */
  strcpy(name, "Ray");
  printf("Nice to meet you, %s.\n", name); /* Nice to meet you, Ray */
  return 0;
}
```

Another example

```
char mystring[1000];   /* 'mystring' is a char array of length 1000 */
int len;
int i, j;
char temp;

strcpy(mystring, "binky");
len = strlen(mystring);









for (i = 0, j = len - 1; i < j; i++, j--) {
  temp = mystring[i];
  mystring[i] = mystring[j];
  mystring[j] = temp;
}
```

Another example

```
char mystring[1000];   /* 'mystring' is a 1000 char array */
int len;
int i, j;
char temp;

strcpy(mystring, "binky");
len = strlen(mystring);


/*
   Reverse the chars in the string:
   i starts at the beginning and goes up
   j starts at the end and goes down
   i/j exchange their chars as they go until they meet
 */
for (i = 0, j = len - 1; i < j; i++, j--) {
  temp = mystring[i];
  mystring[i] = mystring[j];
  mystring[j] = temp;
}
/* at this point 'mystring' should be "yknib" */
```

Calling external library: LAPACK

LAPACK (Linear Algebra PACKage)

- library for numerical linear algebra
- e.g., systems of linear equations, least squares, eigen-decomposition, singular value decomposition, matrix factorization (e.g., LU, QR, Cholesky)
- written in FORTRAN
- See e.g. this for a list of LAPACK functions
- See this for an installation guide of LAPACK.
- In this class, we use linux10.stat.iastate.edu for the compilation

Calling Fortran subroutines from C

Example: dgesv

- Construct a function prototype
  - use pointer arguments
  - add suffix underscore character
  - matching types:

Fortran	C
REAL, REAL*4	float
REAL*8, DOUBLE PRECISION	double
INTEGER	int
COMPLEX, COMPLEX*8	float[2] (an array of two floats)
COMPLEX*16	double[2] (an array of two doubles)

Calling Fortran subroutines from C

Example: dgesv

```
  N       (input) INTEGER

  NRHS    (input) INTEGER

  A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)

  LDA     (input) INTEGER

  IPIV    (output) INTEGER array, dimension (N)

  B       (input/output) DOUBLE PRECISION array, dimension (LDB,NRHS)

  LDB     (input) INTEGER

  INFO    (output) INTEGER
```

```c
void dgesv_(int *N, int *NRHS, double *A, int *LDA, int *IPIV, double *B,
            int *LDB, int *INFO);
```

Calling Fortran subroutines from C

- Array handling:
  - Fortran array indexing starts at 1 (default) (Note: C starts at 0)
    - Not a big problem for passing simple array.
    - Pay attention to this when reading Fortran's documentations
  - Fortran presents higher dimensional arrays "reversely" compared to C
    - Important for dealing with multi-dimensional arrays (e.g. matrix)
    - Need to "transpose" the matrix before passing to Fortran subroutines:
      - e.g. $\boldsymbol{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$. It is organized as $1, 2, 3, 4, 5, 6$ in memory.
      - $\boldsymbol{A}^\intercal = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$. It is organized as $1, 3, 5, 2, 4, 6$ in memory.
      - The $\boldsymbol{A}^\intercal$ does not play any important role. It is used to explain that we pass (the pointer of the first element of) $1, 3, 5, 2, 4, 6$, which is still interpreted as $\boldsymbol{A}$ in Fortran, to Fortran.

Example (dgesv)

```c
#include <stdio.h>

void dgesv_(int *N, int *NRHS, double *A, int *LDA, int *IPIV,
            double *B, int *LDB, int *INFO);

int main(){
  double m[] = {
    3, 1, 3,
    1, 5, 9,
    2, 6, 3
  };
  double x[] = {
    -1, 3, -3
  };

  int ipiv[3];
  int i, j, n1, n2, info;
  double mt[9];

  for (i=0; i<3; i++){
    for (j=0; j<3; j++){
      printf("%5.1f", m[i*3+j]);
    }
    putchar('\n');
  }
```

Example (dgesv)

```
  /* transpose */
  for (i=0; i<3; i++){
    for (j=0; j<3; j++){
      mt[i*3+j] = m[j*3+i];
    }
  }

  n1 = 3;
  n2 = 1;
  dgesv_(&n1, &n2, mt, &n1, ipiv, x, &n1, &info);

  if (info != 0)
    printf("dgesv error %d\n", info);

  for (i=0; i<3; i++)
    printf("%5.1f\n", x[i]);

  return 0;
}
```

On linux10: compile with flags -llapack -lblas -lgfortran

Simulating normal vectors - Cholesky decomposition

See dpotrf

```
#include <stdio.h>

void dpotrf_(char *UPLO, int *N, double *A, int *LDA, int *INFO);

int main(){
  double m[9] = {
    3, 1, 2,
    1, 5, 1,
    2, 1, 3
  };
  char *lower;
  int i, j, n, info;
```

Simulating normal vectors - Cholesky decomposition

```
  /* symmetric: no need to tranpose */

  lower = "L";
  n = 3;
  dpotrf_(lower, &n, m, &n, &info);
  if (info != 0)
    printf("failure with error %d\n", info);

  /* only the lower triangular part are changed in fortran indexing */
  /* i.e. the upper triangular part of the corresponding C array */
  for (i=0; i<3; i++){
    for (j=0; j<3; j++){
      printf("%f ", m[i*3+j]);
    }
    printf("\n");
  }

  return 0;
}
```

On linux10: compile with flags -llapack -lblas -lgfortran

Simulating normal vectors

```
#include <stdio.h>
#include <time.h>
#define MATHLIB_STANDALONE
#include <Rmath.h>

void dpotrf_(char *UPLO, int *N, double *A, int *LDA, int *INFO);
int chol(double *m, int n);

int main(){

  double mu[3] = {1,2,3};
  double A[9] = {
    3, 1, 2,
    1, 5, 1,
    2, 1, 3
  };
  double z[3], out[3];
  int i, j, n=3;
```

Simulating normal vectors

```
  /* Cholesky decomposition */
  chol(A, n);

  /* simulate a standard normal random vector */
  set_seed(time(NULL), 580580); /* set_seed */
  for (i=0; i<n; i++)
    z[i] = rnorm(0.0, 1.0);

  /* transformation */
  for (i=0; i<n; i++){
    out[i] = mu[i];
    for (j=0; j<n; j++)
      out[i] += A[i*n + j] * z[j];
    printf("%f ", out[i]);
  }

  printf("\n");

  return 0;
}
```

Simulating normal vectors

```
/* output as the lower triangular matrix */
int chol(double *m, int n){
  char *lower;
  int i, j, info;

  /* symmetric: no need to tranpose */
  lower = "U";
  dpotrf_(lower, &n, m, &n, &info);
  if (info != 0)
    printf("dpotrf error %d\n", info);

  for (i=0; i<3; i++)
    for (j=(i+1); j<3; j++)
      m[i*n+j] = 0;

  return 0;
}
```

On linux10: compile with flags -llapack -lblas -lgfortran -lRmath -lm

BLAS (Basic Linear Algebra Subprograms)

- Note: we link the BLAS library (-lblas) in the compilation in previous LAPACK examples
- Building blocks for performing basic vector and matrix operations
- See this for the documentation.

Example (dgemv)

See: dgemv

```
#include <stdio.h>

void dgemv_(char *TRANS, int *M, int *N, double *ALPHA, double *A,
            int *LDA, double *X, int *INCX, double *BETA, double *Y,
            int *INCY);

int main(){
  double A[] = {
    3, 1, 3,
    1, 5, 9,
    2, 6, 5
  };

  double x[] = {
    -1, -1, 1
  };

  double y[] = {
    0, 0, 0
  };
```

Example (dgemv)

```
  int i, j;
  char trans='T';
  int m=3, n=3, lda=3, incx=1, incy=1;
  double alpha=1.0, beta=0.0;

  for (i=0; i<3; i++){
    for (j=0; j<3; j++){
      printf("%5.1f", A[i*3+j]);
    }
    putchar('\n');
  }

  dgemv_(&trans, &m, &n, &alpha, A, &lda,
              x, &incx, &beta, y, &incy);

  for (i=0; i<3; i++)
    printf("%5.1f\n", y[i]);

  return 0;
}
```

On linux10: compile with flags -lblas -lgfortran