第 5 章

排序、索引与搜索

根据实际应用的需要，一个数组中可能存放有大量的数据，例如从 Excel 电子表格中导入的数据、从工业现场采集到的传感器数据等。这些导入数据的顺序可能是杂乱无章的，经常需要对其进行排序和索引，以便在其中搜索特定的数据。

作为本篇的结束，本章将在前面各章介绍 MATLAB 程序设计的相关概念和基本方法的基础上，以数组元素的排序、索引与搜索为例，介绍它们在 MATLAB 程序中的简单应用。主要知识点如下：

5.1　排序

了解冒泡法和选择法排序的基本原理，掌握实现排序的 MATLAB 程序设计方法，掌握利用 MATLAB 内置函数 sort 和 sortrows 实现排序的方法。

5.2　索引

了解索引的概念，掌握自行编制 MATLAB 程序和利用内置函数实现索引的方法。

5.3　搜索

了解搜索的概念，了解顺序搜索和对分搜索的基本原理，掌握自行编制 MATLAB 程序和利用内置函数 find 实现搜索的基本方法。

5.1　排序

所谓**排序**(Sort)就是将一组数据按照指定的顺序进行排列。基本的排列顺序包括递增和递减，即分别按照从小到大和从大到小的顺序对数据进行重新排列。

5.1.1　排序的基本方法

比较经典的排序算法有**冒泡法**和**选择法**。冒泡法排序是依次比较相邻的两个数，将小数放在前面，大数放在后面。选择法排序的基本原理是从第一个数开始，将其与后面的各数据进行比较，找出最小或者最大的数，将该数与第一个数交换位置。再从第二个数开始，将其与后面的各数进行比较，最小或者最大值作为第二个数并与第二个数据交换位置。以此类推。

选择法排序是对冒泡法排序的改进，其基本原理是：每轮比较并不马上交换位置，而是找到本轮比较中的最小值或最大值，记下该数据的位置，在本轮比较结束后，再交换位置。

这里以冒泡法为例，介绍排序的基本原理及程序实现方法。

实例 5-1　冒泡法排序。

编制 MATLAB 函数实现冒泡法排序。程序代码如下：

```
%文件名：ex5_1.m
clc
clear
A = [17 80 32 53 17 61 27 66 69 75];      %主程序
Y = sortBub(A);
fprintf("排序前 排序后\n")
for i = 1:length(A)
   fprintf("%5d %5d\n",A(i),Y(i))
end
% ============================================================
function X = sortBub(A)                   %冒泡法排序函数
    N = size(A,2);
    for i = 1:N
        for j = 1:N-1
          if A(j) > A(j+1)
             a = A(j);
             A(j) = A(j+1);
             A(j+1) = a;
          end
        end
    end
    X = A;
end
```

在上述主程序中，首先给定整数向量 $\boldsymbol{A}$，之后调用 sortBub 函数实现冒泡法排序。在 sortBub 函数中，利用两层 for 循环实现冒泡法排序。在内层循环中，每次循环将向量 $\boldsymbol{A}$ 中相邻的两个元素进行比较。如果前面一个数大于后面一个数，则将两个数交换。这样重复到循环结束，实现向量 $\boldsymbol{A}$ 中数据的排序，最后将结果保存到向量 $\boldsymbol{X}$ 中返回。

程序运行结果如下：

```
排序前   排序后
   17     17
   80     17
   32     27
   53     32
   17     53
   61     61
   27     66
   66     69
```

```
    69     75
    75     80
```

5.1.2 排序内置函数

根据排序的基本原理，MATLAB 中提供了几个内置函数，实现数组元素的排序。其中 **sort** 函数实现普通的排序，**sortrows** 函数实现对二维数组按行排序。这些函数都可以在 MATLAB 的命令行窗口直接用脚本命令调用。

1. sort 函数

sort 函数用于实现数组中各数据的排序，其基本调用格式为：

```
B = sort(A,dim,direction)
```

其中，参数 dim 指定排序的维数，direction 指定排序的顺序，可以为 **'ascend'** 或 **'descend'**，分别表示递增或递减排序，默认为递增排序。如果没有参数 dim 和 direction，则默认是对数组 A 中的每一列进行递增排序。

例如：

```
>> A = [1  3  -1;0  -2,4]
A =
     1     3    -1
     0    -2     4
>> sort(A)                              % 默认按列递增排序
ans =
     0    -2    -1
     1     3     4
>> sort(A,1)                            % 按列递增排序
ans =
     0    -2    -1
     1     3     4
>> sort(A,'descend')                    % 按列递减排序
ans =
     1     3     4
     0    -2    -1
>> sort(A,2)                            % 按行递增排序
ans =
    -1     1     3
    -2     0     4
```

2. sortrows 函数

sortrows 函数可以实现对二维数组按行排序，其典型调用格式为：

```
B = sortrows(A,col,direction)
```

其中，参数 direction 可以是字符向量 **'ascend'** 或 **'descend'**；参数 col 指定基于第 col 列对数组中的各行进行递增或递减排序。如果没有后面两个入口参数，则默认对数组 A 中的各行按

第一列进行递增排序。

例如，如下命令及其执行结果为：

```
>> A = [1  3  -1;0  -2,4]
A =
     1     3    -1
     0    -2     4
>> sortrows(A,3,'descend')
ans =
     0    -2     4
     1     3    -1
```

对数组 A 的各行，按照第 3 列进行递减排列，也就是第 3 列数据最大的行排序到最前面，第 3 列数据最小的行排序到最后面。

参数 col 和 direction 也可以是长度相同的行向量，其中每个元素对应排序的一列。例如，命令

```
>> sortrows(A,[3,5],{'ascend','descend'})
```

基于第 3 列按升序对数组 A 的行进行排序，然后基于第 5 列按降序排序。

如下命令：

```
>> A = [26  99  91  3;...
        30  74  88  43;...
        62  35  82  32;...
        30  59  27  17;...
        83  11  60  18];
>> sortrows(A,[1,2],{'descend','ascend'})
```

首先产生一个 4×4 数组 A，再调用 **sortrows** 函数对其先按第一列递减排序。如果第一列数据相等，再按第二列递减排序。排序结果如下：

```
ans =
    83    11    60    18
    62    35    82    32
    30    59    27    17
    30    74    88    43
    26    99    91     3
```

5.2　索引

索引(Index)实际上也是一种排序。进行索引排序时，不仅返回排序后的数组，同时也返回一个新的索引数组，指示排序后各数据在原数组中的位置下标。

上述 **sort** 和 **sortrows** 函数都提供了另外一种调用格式，可以很方便地实现索引排序。归纳起来，这种形式的入口参数完全相同，只是在调用时，需要另外给定索引数组出口参数，

索引数组中的每个元素代表递增或者递减排序后原数据的位置。

例如，对前面得到的 4×4 数组 A，执行如下命令：

```
>> [B,I] = sortrows(A,'descend')
```

得到的结果为：

```
B =
    83    11    60    18
    62    35    82    32
    30    74    88    43
    30    59    27    17
    26    99    91     3
I =
     5
     3
     2
     4
     1
```

其中，B 为对数组 A 按第 1 行进行降序排列(递减排序)后得到的数组，列向量 ***I*** 中各元素分别指示排序后 B 中各行在排序前数组 A 中所在的行号，向量的长度等于原数组 A 的行数。

如果执行如下命令：

```
>> [C,I1] = sort(A,'descend')
```

该命令对 A 中的各列分别进行递减排序，得到如下结果：

```
C =
    83    99    91    43
    62    74    88    32
    30    59    82    18
    30    35    60    17
    26    11    27     3
I1 =
     5     1     1     2
     3     2     2     3
     2     4     3     5
     4     3     5     4
     1     5     4     1
```

由此可见，采用 **sort** 函数进行索引排序，得到的索引数组维数与原数组相同，索引数组中的各元素表示排序后的数组 C 中各元素在原数组 A 中的索引下标。

同步练习

5-1　修改实例 5-1 中的程序，实现递减排序。

5-2　编制函数实现选择法递减排序。

5-3　修改同步练习 5-2 的程序，实现索引排序功能。

5.3　搜索

搜索(Search)指的是在给定的一组数据(例如数组)中查找满足指定条件的数据。在高级语言程序中,常用的搜索算法有顺序搜索、对分搜索等。在 MATLAB 中,专门提供了 **find** 函数实现搜索。

5.3.1　搜索的基本方法

这里主要介绍顺序搜索和对分搜索的基本方法及实现程序。所谓**顺序搜索**,就是从数组的第一个数据开始,逐一判断各元素是否为指定的数据(关键字),或者是否满足指定的条件。为实现**对分搜索**,首先必须对需要搜索的向量数据进行排序。之后,将向量中处于中间位置的元素与关键字进行比较。如果中间位置上的元素不等于关键字,则进一步确定查找范围应该在向量的前半部分还是后半部分,并在新确定的范围内,继续按上述方法进行查找,直到获得最终结果。

实例 5-2　**顺序搜索**。

顺序查找一个给定向量中是否含有指定的关键字,并返回该关键字在向量中的序号。程序代码如下:

```
%文件名:ex5_2.m
clear
clc
A = [7  4  4  10  1  9  10  8  1]      %任一给定待搜索的向量
key = 1;                                %指定搜索的关键字
index = search(A,key);                  %搜索
fprintf("搜索关键字为%d,",key);
fprintf("所在下标位置为:");
fprintf("%2d",index);
fprintf("\n");
%%=========================================================
function index = search(A,key)          %顺序搜索函数
    len = length(A);
    index = -1;                         %返回结果初始化为-1
    j = 1;
    for i = 1:len                       %顺序搜索
       if A(i) == key                   %如果找到关键字,
          index(j) = i;                 %则将索引保存到向量 index 中
          j = j+1;                      %索引向量 index 的下标加 1
       end
    end
end
```

程序运行结果如下:

```
A = 7  4  4  10  1  9  10  8  1  3
搜索关键字为 1,所在下标位置为: 5  9
```

由于向量 ***A*** 中有两个待搜索的关键字“1”,因此搜索结果返回一个长度为 2 的向量 index,其中的两个元素分别指示关键字在向量 ***A*** 中第 5 和第 9 位置。根据 search 函数的定义,如果向量 ***A*** 中没有关键字,返回 index 将是一个标量－1。

实例 5-3　对分搜索。

利用对分搜索查找一个给定向量中是否含有指定的关键字,并返回该关键字在向量中首次搜索到的序号。程序代码如下:

```
%文件名: ex5_3.m
clear
clc
A = [5  10  2  9  7  4  2  5  5  2];        %定义待搜索向量
key = 5;                                    %指定搜索关键字
[B,index] = binsearch(A,key);               %搜索
fprintf("排序前: ");                        %显示结果
fprintf(" %3d",A);fprintf("\n");
fprintf("排序后: ")
fprintf(" %3d",B);fprintf("\n");
fprintf("搜索关键字为 %d,所在下标位置为: %d\n",key,index);
% % ======================================================
function [B,index] = binsearch(A,key)       %对分搜索函数
   if ~issorted(A)                          %如果 A 未经排序
      B = sort(A);                          %则首先排序
      end
      low = 1; high = length(A);            %设置首尾指针
      index = 0;                            %关键字位置指针,初始化为 0
      while low <= high && index == 0
          mid = floor((low+high)/2);        %确定中间数位置
          if B(mid) == key                  %如果中间数等于关键字
              index = mid;                  %则表示搜索到,保存位置
          elseif key<B(mid)                 %否则,重新设置首尾指针,继续搜索
              high = mid-1;
          else
              low = mid+1;
          end
      end
end
```

在上述对分搜索函数中,首先将被搜索的向量进行递增排序。其中,内置函数 **issorted** 返回一个逻辑型变量,表示向量 ***A*** 是否已经进行了排序。之后,根据对分搜索原理实现搜索。在搜索过程的每次循环中,利用 low 和 high 确定搜索的对分区间。如果当前对分区间中间的数等于关键字,则表示搜索成功,将其位置保存到 index 后退出循环;否则,如果对

分区间中间的数不等于关键字，则根据关键字与中间数的相对大小，修改下一次循环搜索的区间，即重新设置 low 或 high 指针，继续下一次循环。

程序运行结果如下：

```
排序前:   5 10  2  9  7  4  2  5  5  2
排序后:   2  2  2  4  5  5  5  7  9 10
搜索关键字为 5,所在下标位置为: 5
```

在上述结果中，$\boldsymbol{A}$ 是经递增排序后的向量。index＝5 表示在排序后的向量中，待搜索的关键字“5”是第 5 个元素。

5.3.2　搜索内置函数

在 MATLAB 中，提供了 **find** 函数，根据该函数的功能，结合条件表达式可以用于实现指定关键字或者满足指定条件的多个数据的搜索。

1. 非零数据的搜索

函数 **find** 的基本功能是查找非零元素的索引和值，其基本调用格式为：

```
k = find(X,n,dir)
```

其中，参数 X 为输入向量或者数组；参数 n 为要查找的非零元素的个数；参数 dir 可以为 **'first'** 或 **'last'**，表示查找与 X 中非零元素对应的最前面或者最后面 n 个索引，默认值为 **'first'**。

如果未给定参数 n 和 dir，则返回 X 中所有非零元素所在的下标位置向量。如果给定了参数 n，则返回向量 k 的长度小于或等于 n，k 中的每个元素表示在 X 中前面或者后面 n 个非零元素所在位置下标。

例如，如下命令

```
>> X = [1  0  2  0  -1  1  0  -2  4];
>> k1 = find(X)
>> k2 = find(X,3,'last')
>> k3 = find(X,10,'first')
```

执行结果为：

```
k1 = 1  3  5  6  8  9
k2 = 6  8  9
k3 = 1  3  5  6  8  9
```

再如，如下命令及其执行结果为：

```
>>  A = [1  0  2;0  1  -1]
A =   1  0   2
      0  1  -1
>> find(A,3)
ans = 1  4  5
```

在本例中，A 是一个二维数组，搜索过程是按列进行的，因此返回结果中的“4”表示数组 A 中的第 4 个元素，即第二行第二列元素。

2. 给定条件搜索

要利用 **find** 函数查找符合给定条件的数组元素，可以利用关系表达式构造逻辑型数组，再作为 **find** 函数的第一个参数 X。

例如，如下命令及其执行结果为：

```
>> X = [1  10  9  2  4  6];
>> find(X<5,3)
ans = 1  2  4
```

其中，关系表达式 $X<5$ 的返回结果为逻辑型向量[0 1 1 0 0 1]，向量中每个 1 表示在原向量 X 中对应位置上的元素小于 5。因此将 $X<5$ 作为 **find** 函数的第一个参数，即可搜索向量 X 中所有小于 5 的元素。返回结果为一个列向量，其中的每个元素表示数组 X 中前面 3 个小于 5 的元素在数组中的下标，即元素 $X(1)$、$X(2)$和 $X(4)$都小于 5。

再如，如下命令：

```
>> X = [1  -1  0  2  -1  5  1  -1  0  -1];
>> find(X == -1,3,'last')
```

从尾到头反序搜索向量 X，返回 X 中最后 3 个－1 元素所在的索引位置，得到如下结果：

```
ans = 5  8  10
```

3. 搜索数据的返回

find 函数还有另外一种调用格式，不仅可以返回数据所在数组中的各维下标，同时还可以返回满足条件的数据。在这种调用格式中，需要给定 2 个或者 3 个返回参数。如果只给定 2 个返回参数，则返回数组 X 中每个非零元素的行和列下标。如果给定了第 3 个返回参数，则同时返回 X 中非零元素的向量。

例如，如下命令及其执行结果为：

```
>> X = [1  -1  0  2; -1  5  1  -1]
X =
     1    -1     0     2
    -1     5     1    -1
>> [r,c,v] = find(X,2)
r = 2  1
c = 1  2
v = 1  -1
```

在上述命令中，搜索矩阵 X 中前面两个非零元素。在返回的列向量 r 和 c 中，$r(1)=2$，$c(1)=1$，表示为元素 $X(2,1)$；$r(2)=1$，$c(2)=2$，表示为元素 $X(1,2)$。在原数组 X 中，这两个元素是非零元素。由向量 v 可知，这两个元素分别为 1 和－1。

需要注意的是，对于给定条件搜索，**find** 函数的第一个参数“$X==-1$”是一个逻辑或

者关系表达式，该表达式判断指定向量或者数组 X 中的每个元素是否满足给定条件，返回维数与 X 相同、但各元素都为 0 或 1 的数组。此时，返回的第三个参数 v 是一个全 1 列向量，而不是原数组 X 中的元素。

例如，如果将上述搜索命令修改为：

```
>> [r,c,v] = find(X==== -1,2)
```

则表示搜索数组 X 中前面两个－1 元素，此时返回向量 v 是一个逻辑型数组，结果为：

```
v =
  2×1 logical 数组
   1
   1
```

同步练习

5-4　在实例 5-2 中，采用的是从头到尾顺序搜索。修改程序实现从尾到头反序搜索，返回找到关键字一共执行的搜索次数。不用考虑没有关键字的情况。

5-5　在实例 5-3 中，对分搜索函数 binsearch 返回的结果表示关键字在排序后的向量中的位置。修改程序，使得返回结果表示关键字在原向量中的位置(提示：利用索引排序实现)。

5-6　统计一个数组中正数、负数和 0 的个数，要求利用 find 函数实现。

5-7　将一个正整数向量中所有的奇数和偶数分别保存到另一个向量中。