元胞数组

讲义和代码可以看配套的课程的第一个视频下载

- 配套课程b站：《MATLAB教程新手入门篇（数学建模清风主讲，适合零基础同学观看）》
- 数学建模清风老师版权所有

MATLAB中的数据类型

- 数据类型是编程语言的核心概念之一，它决定了变量所能存储的数据种类，以及能够对这些数据执行的操作。
- whos函数能显示工作区各变量的详细信息，包括变量的名称、大小、占用的内存大小和数据的属性。
- 如果我们只想查看变量的类型，我们可以使用class函数
- MATLAB中也提供了一系列判断变量是否属于某个数据类型的函数，属于就返回逻辑值1，不属于就返回逻辑值0。
- 例如islogical函数可以判断输入变量是否为逻辑类型

创建元胞数组

- 元胞数组可用于保存不同大小、不同类型的数据。
- 创建元胞数组（cell array）则需要使用英文输入模式下的大括号{}（又称花括号）。

```
c1 = {1:3, 'abcd';
   char('ab','cd','e') 50;
   [5 6 7; 8 9 10] [60;70]}
```

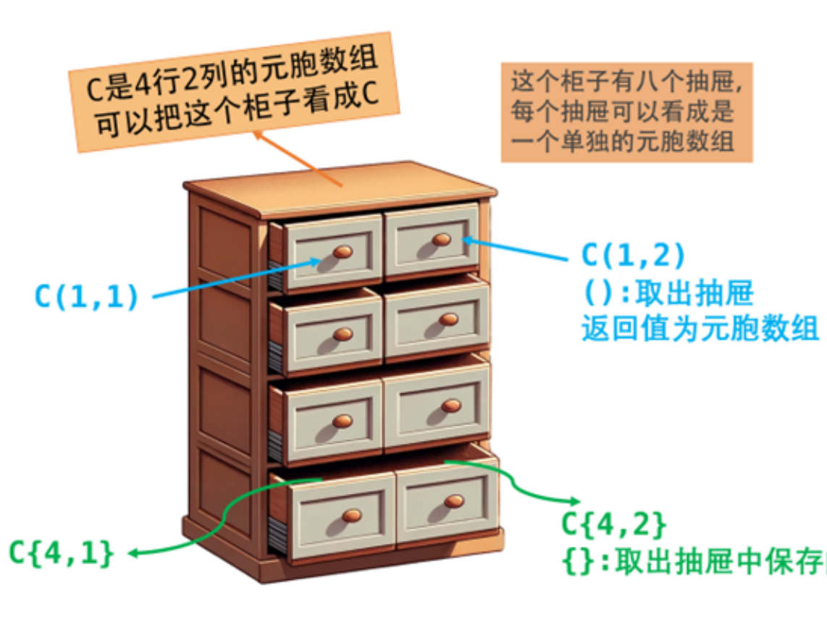
```
3×2 cell 数组
    {[    1 2 3]}    {'abcd'     }
    {3×2 char  }    {[       50]}
    {2×3 double}    {2×1 double}
```

  - MATLAB自动将每个数据保存到了独立的元胞中
- MATLAB将元胞数组的每个元素视为一个独立的元胞，这样就可以灵活地进行修改，而不会影响数组的整体结构。
- 可以使用cell函数来创建一个指定大小且数据全为空矩阵的元胞数组
- 元胞数组中保存的数据也可以是元胞数组，这被称为嵌套的元胞数组

```
c3 = {  {1:3, 'abcd'};
        [3,4;5 6]  }
```

引用元胞数组

示意图：



```
C = {1:3, 'abcd';
     true, 50;
     [5 6 7; 8 9 10], [60;70];
     char('ab','cd','e'), 5+2i}
```

```
4×2 cell 数组
    {[ 1 2 3]}    {'abcd'        }
    {[     1]}    {[          50]}
    {2×3 double}  {2×1 double    }
    {3×2 char }   {[5.0000 + 2.0000i]}
```

使用小括号()引用

小括号()引用返回的是对应位置的元胞数组，而不是元胞数组中存储的数据。

```
% 第 1 行第 1 列位置的元素
c11 = C(1,1)
```
```
1×1 cell 数组
    {[1 2 3]}
```
```
% 第 4 行第 1 列位置的元素
c41 = C(4,1)
```
```
1×1 cell 数组
    {3×2 char}
```
```
% 使用 class 查看 c11 的数据类型
class(c11)
```
```
    'cell'
```
```
% 查看 c11 的大小
size(c11)
```
```
     1     1
```
```
% 偶数行元素
C(2:2:end,:)
```
```
2×2 cell 数组
    {[     1]}    {[              50]}
    {3×2 char}    {[5.0000 + 2.0000i]}
```
```
% 超过索引边界的情况
C(1,3)
```
```
报错：位置 2 处的索引超出数组边界。索引不能超过 2。
```
```
% 线性索引为 5 的元素
C(5)
```
```
1×1 cell 数组
    {'abcd'}
```
```
% 按照线性索引的顺序重新排列元胞数组
C(:)
```
```
8×1 cell 数组
    {[          1 2 3]}
    {[              1]}
    {2×3 double       }
    {3×2 char         }
    {'abcd'           }
    {[             50]}
    {2×1 double       }
    {[5.0000 + 2.0000i]}
```

使用大括号{}引用

使用大括号{}引用元胞数组时，我们可以直接得到对应位置的元胞数组中的数据。

```
% 第 1 行第 1 列元胞中的数据
c_11 = C{1,1}
```
```
     1     2     3
```
```
% 第 4 行第 1 列元胞中的数据
c_41 = C{4,1}
```
```
3×2 char 数组
    'ab'
    'cd'
    'e '
```
```
% 使用 class 查看 c_11 的数据类型
class(c_11)  % 别把这个小括号改成大括号了
```
```
    'double'
```
```
% 查看 c_11 的大小
size(c_11)   % 别把这个小括号改成大括号了
```
```
     1     3
```

链式索引（chained indexing）

- 允许我们在单个表达式中执行多个索引操作

```
cc = {[1 2 3;4 5 6],'abc'}
```
```
1×2 cell 数组
    {2×3 double}    {'abc'}
```
```
% 第一个数据是一个 2 行 3 列的数值矩阵
% 引用这个矩阵中第 1 行第 3 列的元素
cc{1}(1,3)
```
```
     3
```
```
% 第二个数据是三个元素的字符向量
% 引用第 3 个字符
cc{2}(3)
```
```
    'c'
```

- 不允许在小括号()的后面跟上大括号{}进行索引，这样的索引方式会导致错误。

套用 cat、char等函数拼接元胞数组元素

```
student = {'李华',[65 78 90];
           '清风',[95 99 93];
           '张无忌', [79 88 64];
           '苏大强', [54 96 33]}
```

student = 4×2 cell

	1	2
1	'李华'	[65,78,90]
2	'清风'	[95,99,93]
3	'张无忌'	[79,88,64]
4	'苏大强'	[54,96,33]

```
name = char(student{:,1})
```
```
name = 4×3 char 数组
    '李华 '
    '清风 '
    '张无忌'
    '苏大强'
```
```
score = cat(1,student{:,2})
```
```
score = 4×3
    65    78    90
    95    99    93
    79    88    64
    54    96    33
```

显示元胞数组的数据

```
cc = {'xyz',{1:3, 'abcd'};
      [3,4;5 6],3+2i};
disp(cc)
```
```
    {'xyz'     }    {1×2 cell         }
    {2×2 double}    {[3.0000 + 2.0000i]}
```
```
celldisp(cc)
% 大家可以试试 cellplot(cc)
% 我个人觉得效果不是很好看
```
```
cc{1,1} =
    xyz
cc{2,1} =
     3     4
     5     6
cc{1,2}{1} =
     1     2     3
cc{1,2}{2} =
    abcd
cc{2,2} =
     3.0000 + 2.0000i
```

从上面例子可以看出，celldisp 函数会按照线性索引的顺序逐个展开元胞数组。使用 celldisp 函数能使我们更详细地查看元胞数组的内容，进而为后续的数据分析工作提供方便。

拼接元胞数组

- MATLAB中提供了两种不同的方法来拼接元胞数组：使用中括号[]和使用大括号{}
- 使用中括号[]对元胞数组进行拼接时，实际上是对元胞数组中的数据进行拼接，这种操作方式和对数值矩阵的拼接方式完全相同。

```
C1 = {1, 2};
C2 = {'A', 'B'};
C3 = {10, 20};
[C1 C2 C3] % 横向拼接，也能用逗号隔开
% 等价于使用下面两个函数
% horzcat(C1,C2,C3)
% cat(2,C1,C2,C3)
```
```
1×6 cell 数组
  {[1]} {[2]} {'A'} {'B'} {[10]} {[20]}
```
```
[C1;C2;C3] % 纵向拼接，也能用回车键隔开
% 等价于使用下面两个函数
% vertcat(C1,C2,C3)
% cat(1,C1,C2,C3)
```
```
3×2 cell 数组
    {[ 1]}    {[ 2]}
    {'A' }    {'B' }
    {[10]}    {[20]}
```

- 使用大括号{}拼接对参与拼接的原始元胞数组的大小没有要求。实际上，这里大括号的作用就相当于创建了一个嵌套的元胞数组。

```
C4 = {'abc',1};
C5 = {'def',2,3;
   'xyz',4,5};

{C4,C5}
```

ans = 1×2 cell

	1	2
1	1×2 cell	2×3 cell

```
{C4;C5}
```

ans = 2×1 cell

	1
1	1×2 cell
2	2×3 cell

修改元胞数组

（1）使用小括号()修改元胞数组的"抽屉"

（1）使用小括号()修改元胞数组的"抽屉"

当我们使用小括号()来引用元胞数组时，我们实际上是在操作整个"抽屉"。这意味着我们可以更换"抽屉"，但是替换时必须使用另一个"抽屉"，即更换为另一个元胞数组。例如：

```
C = {'apple', 'banana';
     'pear', 'cherry'};
C(2,1) = {'watermelon'}
```
```
2×2 cell 数组
    {'apple'     }    {'banana'}
    {'watermelon'}    {'cherry'}
```

上面的代码将第 2 行第 1 列的"抽屉"{'pear'}替换成了{'watermelon'}。注意，我们使用了一个元胞数组来进行替换，这样保持了"抽屉"的一致性，如果你忘记了加大括号{}就会报错：

```
% 如果赋值的等号右侧不是元胞数组
C(1,3) = 'watermelon'
```
```
报错：无法从 char 转换为 cell。
```

此外，如果我们修改时指定的索引范围超出了元胞数组的大小，那么 MATLAB 会自动对元胞数组进行扩充，没有被指定的位置对应的数据会被填充为空向量[]：

```
C = {'apple', 'banana'};
C(2,1) = {'watermelon'}
```
```
2×2 cell 数组
    {'apple'     }    {'banana'  }
    {'watermelon'}    {0×0 double}
```

此外，对元胞数组使用小括号()修改时，赋值等号的左右两侧元胞数组的大小不必完全一样，只需要大小兼容即可：

```
C = {'apple', 'banana';
     'pear', 'cherry'};
C(:,1) = {[1,2,3]}
```
```
2×2 cell 数组
    {[1 2 3]}    {'banana'}
    {[1 2 3]}    {'cherry'}
```
```
% 左侧大小是 2×1，右侧大小是 1×2
C(:,1) = {'a','b'}
```
```
2×2 cell 数组
    {'a'}    {'banana'}
    {'b'}    {'cherry'}
```
```
% 大小不兼容就会报错
C(:,1) = {'a','b','c'}
```
```
报错：无法执行赋值，因为左侧的大小为 2×1，右侧的大小为 1×3。
```

（2）使用大括号{}修改"抽屉"中的数据

（2）使用大括号{}修改"抽屉"中的数据

当我们使用大括号{}时，我们是在修改"抽屉"中的具体物品，即元胞数组中保存的数据。

```
C = {'apple', 'banana';
     'pear', 'cherry'};
C{2,1} = 'watermelon'
```
```
2×2 cell 数组
    {'apple'     }    {'banana'}
    {'watermelon'}    {'cherry'}
```

如果我们将赋值等号右侧的元素加上大括号，MATLAB 也不会报错。它会认为我们要将这个数据替换成一个新的元胞数组，此时返回的结果就是一个嵌套的元胞数组。

```
C = {'apple', 'banana';
     'pear', 'cherry'};
C{2,1} = {'watermelon'}
```
```
2×2 cell 数组
    {'apple' }    {'banana'}
    {1×1 cell}    {'cherry'}
```

从上面的例子可以看出：使用大括号{}对数据进行修改时，对新数据的类型并没有要求，这也正是元胞数组储存数据的优势所在。我们再来举个例子：

```
C = {'apple', 'banana';
     'pear', 'cherry'};
C{2,1} = 1:3
```
```
2×2 cell 数组
    {'apple'}    {'banana'}
    {[1 2 3]}    {'cherry'}
```

注意，当我们使用大括号{}修改数据时，MATLAB不支持使用简单的赋值语句对两个或两个以上的数据进行修改

```
C = {'apple', 'banana';
     'pear', 'cherry'};
C{:,1} = 'xyz'
```
```
报错：不支持使用简单赋值语句为 2 个元素赋值。请考虑使用以逗号分隔的列表赋值。
```

逗号分隔的列表

逗号分隔的列表可以理解为一系列使用逗号分隔的数字、表达式或者变量。

deal函数：将输入变量分发给输出变量

```
% 将 0 赋值给 a,b,c 三个变量
[a,b,c] = deal(0)
```
```
a =
     0
b =
     0
c =
     0
```
```
% 为 x、y、z 三个变量分别赋值
[x,y,z] = deal('Mate',60,'pro')
```
```
x =
    'Mate'
y =
    60
z =
    'pro'
```
```
C = {'apple', 'banana';
     'pear', 'cherry'};
[C{:,1}] = deal('xyz')
```
```
2×2 cell 数组
    {'xyz'}    {'banana'}
    {'xyz'}    {'cherry'}
```

删除元胞数组

（1）使用小括号()删除元胞数组的"抽屉"

借助小括号()，我们可以将整个"抽屉"从数组中移除。例如，如果我们想删除元胞数组C的第二行元素，可以简单地将其赋值为空向量[]来实现删除：

```
C = {'apple', 'banana';
     'pear', 'cherry'};
C(2,:) = [] % 使用小括号()
```
```
1×2 cell 数组
    {'apple'}    {'banana'}
```

（2）使用大括号{}删除"抽屉"中的数据

使用大括号{}，我们可以删除"抽屉"中保存的具体数据，但这并不会移除"抽屉"本身，仅仅是将"抽屉"内的数据清空。例如我们想删除C中第二行第一列元胞中的数据：

```
C = {'apple', 'banana';
     'pear', 'cherry'};
C{2,1} = []   % 使用大括号{}
```
```
2×2 cell 数组
    {'apple'   }    {'banana'}
    {0×0 double}    {'cherry'}
```

对元胞数组进行运算

和普通的数值数组不同，绝大多数的运算方法都不适用于元胞数组

```
C = {1,2,3};
C + 1
```
```
报错：'cell' 类型的操作数不支持运算符 '+'。
```
```
C == 1
```
```
报错：'cell' 类型的操作数不支持运算符 '=='。
```
```
sin(C)
```
```
报错：函数 'sin' 的输入或输出的数目或类型不正确。
```
```
sum(C)
```
```
报错：数据类型无效。第一个参数必须为数值或逻辑值。
```

返回数组大小的三个函数：size、numel和length对元胞数组仍然有效：

```
C  = {65,'A',true;
      [1 2 3],'xyz','a'}
size(C)  % 返回行数和列数
```
```
     2     3
```
```
[r_num,c_num] = size(C)
```
```
r_num =
     2
c_num =
     3
```
```
numel(C)  % 返回元素总数
```
```
     6
```
```
length(C)  % 返回行列数中较大值
```
```
     3
```

还有几个函数比较特殊，那就是第三章集合运算中介绍的六个函数：unique(返回数组的唯一值)、ismember(判断一个数组的元素是否在另一个数组内)、intersect(交集)、union(并集)、setdiff(差集)和setxor(对称差集)。它们只能用于字符向量元胞数组，即元胞数组中的数据全为字符向量时才可以使用。

```
C = {1,2,3};
unique(C)
```
```
报错：元胞数组输入必须为字符向量元胞数组。
```
```
ismember(C,{2,4,5})
```
```
报错：类 cell 的输入 A 和类 cell 的输入 B 必须为字符向量元胞数组，除非其中某个输入为字符向量。
```
```
C1 = {'bc','aca','bc','ab'};
a = unique(C1)
```
```
a  = 1×3 cell 数组
    {'ab'}    {'aca'}    {'bc'}
```
```
C1 = {'bc','aca','bc','ab'};
[~,b,c] = unique(C1)
% 第一个返回值就是 C1 中的唯一值，即上一行代码算出来的 a。我们这里用~代替，表示我们不需要这个返回结果。
% b 就是 a 中每个元素在 C1 中的索引
% c 就是 C1 中每个元素在 a 中的索引
```
```
b =
     4
     2
     1
c =
     3
     2
     3
     1
```
```
% 按照原来的顺序输出唯一值
unique(C1,'stable')
```
```
1×3 cell 数组
    {'bc'}    {'aca'}    {'ab'}
```
```
C2 = 'bc';
ismember(C1,C2)
```
```
1×4 logical 数组
    1   0   1   0
```
```
C3 =  {'aca','BC','Ab'};
ismember(C3,C1)
```
```
1×3 logical 数组
    1   0   0
```

元胞数组和其他数据类型的转换

num2cell函数

num2cell函数将数组转换为元胞数组，转换后的元胞数组中的数据大小相同

- 用法一：C = num2cell(A) 通过将 A 的每个元素放置于 C 的一个单独元胞中，来将数组 A 转换为元胞数组 C。

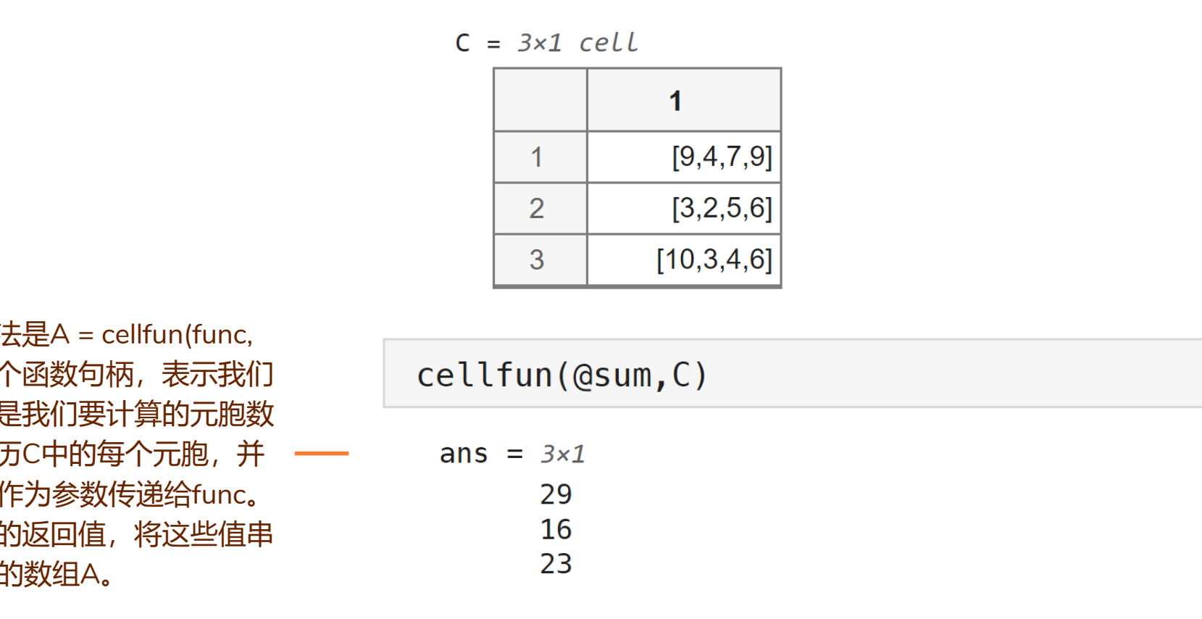
```
a = reshape(1:12,3,4)
```
```
     1     4     7    10
     2     5     8    11
     3     6     9    12
```
```
% 将数值矩阵 a 转换为元胞数组
num2cell(a)
```
```
3×4 cell 数组
    {[1]}    {[4]}    {[7]}    {[10]}
    {[2]}    {[5]}    {[8]}    {[11]}
    {[3]}    {[6]}    {[9]}    {[12]}
```
```
% 生成一个字符矩阵，'A'的 ASCII 码为 65
b = char(a + 64)
```
```
3×4 char 数组
    'ADGJ'
    'BEHK'
    'CFIL'
```
```
% 将字符矩阵 b 转换为元胞数组
num2cell(b)
```
```
3×4 cell 数组
    {'A'}    {'D'}    {'G'}    {'J'}
    {'B'}    {'E'}    {'H'}    {'K'}
    {'C'}    {'F'}    {'I'}    {'L'}
```

- 用法二：C = num2cell(A,dim) 将 A 的内容划分成 C 中单独的元胞，其中 dim 表示维度。dim等于1表示沿着行方向划分，dim等于2表示沿着列方向划分。

```
% a 同上
c1 = num2cell(a,1)
% 沿着行方向进行划分，得到的元胞数组 c1 中的数据就是 a 的每列元素
```
```
c1 = 1×4 cell 数组
  列 1 至 3
   {3×1 double} {3×1 double} {3×1 double}
  列 4
   {3×1 double}
```
```
% 显示元胞数组 c1 的数据
celldisp(c1)
```
```
c1{1} =    c1{2} =     c1{3} =     c1{4} =
   1            4           7          10
   2            5           8          11
   3            6           9          12
```
```
c2 = num2cell(a,2)
% 沿着列方向进行划分，得到的元胞数组 c2 中的数据就是 a 的每行元素
```
```
c2 = 3×1 cell 数组
    {[1 4 7 10]}
    {[2 5 8 11]}
    {[3 6 9 12]}
```

mat2cell函数

mat2cell函数是num2cell函数的进阶版，它允许我们将一个数组分割为多个大小不同的子块，并将这些子块存储在元胞数组中。



```
A = randi(10,6,5)
```
```
    10     2     5    10     3
     1    10     3     7     3
     3    10     4     9     2
     8     4    10    10     6
     7     1    10     1     3
     6    10     1    10     9
```
```
r = [3 2 1];
c = [1 4];
cell_A = mat2cell(A, r, c)
```
```
3×2 cell 数组
    {3×1 double}    {3×4 double }
    {2×1 double}    {2×4 double }
    {[       6]}    {[10 1 10 9]}
```

cell2mat函数

cell2mat函数是mat2cell函数的逆操作，它主要用于将元胞数组转换为普通的数组。

A = cell2mat(C)将元胞数组转换为普通数组。元胞数组的元素必须全都包括相同的数据类型，并且生成的数组也是该数据类型。

```
C = {5,     [3 7 4];
    [6; 9], [8 7 8; 4 9 7]}
```
```
2×2 cell 数组
    {[        5]}    {[   3 7 4]}
    {2×1 double}    {2×3 double}
```
```
cell2mat(C)
```
```
     5     3     7     4
     6     8     7     8
     9     4     7     9
```

对每个元胞中保存的数据应用函数

cellfun函数

cellfun函数的基本用法是A = cellfun(func, C)。这里的func是一个函数句柄，表示我们希望应用的函数， C是我们要计算的元胞数组。cellfun函数会遍历C中的每个元胞，并将每个元胞中的数据作为参数传递给func。然后，它会收集func的返回值，将这些值串联起来，形成一个新的数组A。

```
A = randi(10,3,4);
C = num2cell(A,2)
```

C = 3×1 cell

	1
1	[9,4,7,9]
2	[3,2,5,6]
3	[10,3,4,6]

```
cellfun(@sum,C)
```
```
ans = 3×1
    29
    16
    23
```
```
cellfun(@sort,C,'UniformOutput',false)
```

ans = 3×1 cell

	1
1	[4,7,9,9]
2	[2,3,5,6]
3	[3,4,6,10]

拓展：cellfun函数的运行效率高于for循环吗？

- 不一定，和具体的任务有关
- 通常，当数据量不大时，两种方法都不会花费太长时间，此时使用cellfun函数会让代码更加简单，也更为灵活。

isequal函数

isequal函数的基本语法为：tf = isequal(A,B)，如果A和B等效，则 tf等于逻辑值 1 (true)；否则tf等于逻辑值 0 (false)。这里所说的“等效”要比“完全相同”包含的情况更多，例如字符 'A' 的ASCII码为65，那么MATLAB会认为 'A' 和65等效；再比如逻辑值1(true)和数值1也是等效的。