第七章 稀疏矩阵 单元阵列 结构

在本章中我们要学习三种数据类型：稀疏矩阵，单元阵列和结构。稀疏矩阵是矩阵的一种特殊形式，在这个矩阵中只对非零元素分配内存。单元阵列也是一种矩阵，它的每一个元素可以是 **MATLAB** 任何一种数据类型。它们广泛应用于 **MATLAB** 用户图象界面（GUI）函数。最后，结构提供了一种在单个变量中存储了不同类型的数据的方法，在这个变量中的每一个数据项目都有一个独立的名字。

7.1 稀疏矩阵

我们在第二章中已经学过了普通的 **MATLAB** 数组。当一个普通的数组被声明后，**MATLAB** 将会为每一个数组元素分配内存。例如函数 a = eye(10)要创建了 100 个元素，按 10×10 的结构分配，对角线上的元素均为 1，其余的元素为 0。注意这些数组其中的 90 个元素为 0。这个包含有一百个元素的矩阵，只有 10 个元素包含非零值。这是稀疏矩阵或稀疏数组的一个例子。稀疏矩阵是指一个很大的矩阵，且大多数的元素为 0。

```
>> a=2*eye(10)
a =
     2     0     0     0     0     0     0     0     0     0
     0     2     0     0     0     0     0     0     0     0
     0     0     2     0     0     0     0     0     0     0
     0     0     0     2     0     0     0     0     0     0
     0     0     0     0     2     0     0     0     0     0
     0     0     0     0     0     2     0     0     0     0
     0     0     0     0     0     0     2     0     0     0
     0     0     0     0     0     0     0     2     0     0
     0     0     0     0     0     0     0     0     2     0
     0     0     0     0     0     0     0     0     0     2
```

现在假如我们要创建一个 10×10 的矩阵，定义如下

```
b =
     1     0     0     0     0     0     0     0     0     0
     0     2     0     0     0     0     0     0     0     0
     0     0     2     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     5     0     0     0     0     0
     0     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     0     0     1
```

若 a，b 两矩阵相乘得到的结果为

```
>> c = a * b
c =
     2     0     0     0     0     0     0     0     0     0
     0     4     0     0     0     0     0     0     0     0
     0     0     4     0     0     0     0     0     0     0
     0     0     0     2     0     0     0     0     0     0
     0     0     0     0    10     0     0     0     0     0
     0     0     0     0     0     2     0     0     0     0
     0     0     0     0     0     0     2     0     0     0
     0     0     0     0     0     0     0     2     0     0
```

```
     0     0     0     0     0     0     0     0     2     0
     0     0     0     0     0     0     0     0     0     2
```

这两个稀疏矩阵相乘需要 1900 次相加和相乘，但是在大多数时侯相加和相乘的结果为 0，所以我们做了许多的无用功。这个问题会随着矩阵大小的增大而变得非常的严重。例如，假设我们要产生两个 200×200 的稀疏矩阵，如下所示

```
a = 5 * eye(200);
b = 3 * eye(200);
```

每一个矩阵有 20000 个元素，其中 19800 个元素是 0。进一步说，对这两个矩阵相乘需要 7980000 次加法和乘法。

我们可以看出对大规模稀疏矩阵进行存储和运算（其中大部分为 0）是对内存空间和 cpu 资源的极大浪费。不巧的是，在现实中的许多问题都需要稀疏矩阵，我们需要一些有效的方示来解决这个问题。

大规模供电系统是现实世界中涉及到稀疏矩阵一个极好的例子。大规模供电系统可以包括上千条或更多的母线，用来产生，传输，分配电能到子电站。如果我们想知道这个系统的电压，电流和功率，我们必须首先知道每一条母线的电压。如果这个系统含有一千个母线，这就要用到含有 1000 个未知数的联立方程组，包括一个方程，也就是说我们要创建含有 1000000 个元素的矩阵。解出这个矩阵，需要上百万次的浮点运算。

但是，在这个系统中，一条母线平均只它的三条母线相连，而在这个矩阵中每一行其他的 996 个元素将为 0，所以在这个矩阵的加法和乘法运算中将会产生 0。如果在求解的过程中这些 0 可以忽略，那么这个电力系统的电压和电流计算将变得简单而高效。

7.1.1 sparse 数据类型

在 **MATLAB** 中有一个专门的数据类型，用来对稀疏进行运算。sparse 数据类型不同于 doulbe 数据，它在内存中只存储非零元素。实际上，sparse 数据类型只存储每一个非零元素的三个值：元素值，元素的行号和列号。尽管非零元素这三个值必须存储在这内存，但相对于存储稀疏矩阵的所有元素来说要简单高效得多。

我们用 10×10 的方阵来说明稀疏矩阵的应用。

```
>> a = eye(10)

a =

     1     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     1     0     0     0     0     0
     0     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     0     0     1
```

如果这个矩阵被转化为稀疏矩阵，它的结果是

```
>> as = sparse(a)
as =
   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
   (5,5)        1
   (6,6)        1
   (7,7)        1
```

```
    (8,8)        1
    (9,9)        1
   (10,10)       1
```

注意在稀疏矩阵存储的是行列地址和这一点所对应的非零数据值。只要一个矩阵的大部分数都是 0，这种方法用来存储数据就是高效的，但是如果非零元素很多的话，那么用占用更多的空间，因为稀疏矩阵需要存储蓄地址。函数 issparse 通常用作检测一个矩阵是否为稀疏矩阵。如果这个矩阵是稀疏的，那么这个函数将会返回 1。

稀疏矩阵的优点可以通过下面的描述体现出来，考虑一个 1000×1000 的矩阵平均每一行只有 4 个非零元素。如果这个矩阵以全矩阵的形式储存，那么它要战胜 8000000 个字节。从另一方面说，如果它转化为一个稀疏矩阵，那么内存的使用将会迅速下降。

7.1.1.1 产生稀疏矩阵

MATLAB 可以通过 sparse 函数把一个全矩阵转化为一个稀疏矩阵，也可以用 **MATLAB** 函数 speye，sprand 和 sprandn 直接产生稀疏矩阵，它们对应的全矩阵为 eye，rand，和 randn。例如，表达式 a = speye(4)将产生一个 4×4 的稀疏矩阵。

```
>> a = speye(4)
a =
    (1,1)        1
    (2,2)        1
    (3,3)        1
    (4,4)        1
```

表达式 b = full(a)把稀疏矩阵转化相应的全矩阵。

```
>> b = full(a)
b =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

7.1.1.2 稀疏矩阵的运算

如果一个矩阵是稀疏的，那么单个元素可以通过简单的赋值语句添加或删除，例如下面的语句产生一个 4×4 的稀疏矩阵，然后把其他的非零元素加入其中。

```
>> a = speye(4)
a =
    (1,1)        1
    (2,2)        1
    (3,3)        1
    (4,4)        1
>> a(2,1) = -2
a =
    (1,1)        1
    (2,1)       -2
    (2,2)        1
    (3,3)        1
    (4,4)        1
```

MATLAB 允许全矩阵与稀疏的混合运算。它们产生的结果可以是全矩阵也可以是稀疏矩阵，这取决于那种结果更高效。更重要的是，任何的适用全矩阵算法同样地也适合稀疏矩阵。

表 7.1 列出的是一些普通的稀疏矩阵。

表 7.1 普通的 **MATLAB** 稀疏矩阵函数

类别	函数	描述
创建一个稀疏矩阵	speye	创建一个单位稀疏矩阵
	sprand	创建一个稀疏矩阵，元素是符合平均分布的随机数
	sprandn	创建一个稀疏矩阵，元素是普通的随机数
全矩阵和稀疏矩阵的转换函数	sparse	把一个全矩阵转化为一个稀疏矩阵
	full	把一个稀疏矩阵转化为全矩阵
	find	找出矩阵中非零元素和它对应的上下标
对稀疏矩阵进行操作的函数	nnz	非零元素的个数
	nonzeros	返回一个向量，其中的元素为矩阵中非零元素
	spones	用 1 代替矩阵中的非零元素
	spalloc	一个稀疏矩阵所占的内存空间
	issparse	如果是稀疏矩阵就返回 1
	spfun	给矩阵中的非零元素提供函数
	spy	用图象显示稀疏矩阵

例 7.1

用稀疏矩阵解决联立方程组

为了解说明稀疏矩阵在 **MATLAB** 中应用，我们将用全矩阵和稀疏矩阵来解决下面的联立方程组。

$1.0x_1 + 0.0x_2 + 1.0x_3 + 0.0x_4 + 0.0x_5 + 2.0x_6 + 0.0x_7 - 1.0x_8 = 3.0$
$0.0x_1 + 1.0x_2 + 0.0x_3 + 0.4x_4 + 0.0x_5 + 0.0x_6 + 0.0x_7 + 0.0x_8 = 2.0$
$0.5x_1 + 0.0x_2 + 2.0x_3 + 0.0x_4 + 0.0x_5 + 0.0x_6 - 1.0x_7 + 0.0x_8 = -1.5$
$0.0x_1 + 0.0x_2 + 0.0x_3 + 2.0x_4 + 0.0x_5 + 1.0x_6 + 0.0x_7 + 0.0x_8 = 1.0$
$0.0x_1 + 0.0x_2 + 1.0x_3 + 1.0x_4 + 1.0x_5 + 0.0x_6 + 0.0x_7 + 0.0x_8 = -2.0$
$0.0x_1 + 0.0x_2 + 0.0x_3 + 1.0x_4 + 0.0x_5 + 1.0x_6 + 0.0x_7 + 0.0x_8 = 1.0$
$0.5x_1 + 0.0x_2 + 0.0x_3 + 0.0x_4 + 0.0x_5 + 0.0x_6 + 1.0x_7 + 0.0x_8 = 1.0$
$0.0x_1 + 1.0x_2 + 0.0x_3 + 0.0x_4 + 0.0x_5 + 0.0x_6 + 0.0x_7 + 1.0x_8 = 1.0$

答案

为了解决这一问题，我们将创建一个方程系数的全矩阵，并用 sparse 函数把他转化为稀疏矩阵。我们用两种方法解这个方程组，比较它们的结果和所需的内存。

代码如下：

```
% Script file: simul.m
%
% Purpose:
% This program solves a system of 8 linear equations in 8
% unknowns (a*x = b), using both full and sparse matrices.
%
% Record of revisions:
% Date          Programmer          Description of change
% ====          ========            ===============
% 10/14/98    S. J. Chapman         Original code
%
% Define variables:
% a                         --Coefficients of x (full matrix)
% as                        --Coefficients of x (sparse matrix)
% b                         --Constant coefficients (full matrix)
% bs                        --Constant coefficients (sparse matrix)
% x                         --Solution (full matrix)
```

```
% xs                          --Solution (sparse matrix)
% Define coefficients of the equation a*x = b for
% the full matrix solution.
a = [
1.0 0.0 1.0 0.0 0.0 2.0 0.0 -1.0; ...
0.0 1.0 0.0 0.4 0.0 0.0 0.0 0.0; ...
0.5 0.0 2.0 0.0 0.0 0.0 -1.0 0.0; ...
0.0 0.0 0.0 2.0 0.0 1.0 0.0 0.0; ...
0.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0; ...
0.0 0.0 0.0 1.0 0.0 1.0 0.0 0.0; ...
0.5 0.0 0.0 0.0 0.0 0.0 1.0 0.0; ...
0.0 1.0 0.0 0.0 0.0 0.0 0.0 1.0
];
b = [ 3.0 2.0 -1.5 1.0 -2.0 1.0 1.0 1.0]';
% Define coefficients of the equation a*x = b for
% the sparse matrix solution.
as = sparse(a);
bs = sparse(b);
% Solve the system both ways
disp ('Full matrix solution:');
x = a\b
disp ('Sparse matrix solution:');
xs = as\bs
% Show workspace
disp('Workspace contents after the solutions:')
whos
```

运行这个程序，结果如下

```
>> simul
Full matrix solution:
x =
    0.5000
    2.0000
   -0.5000
   -0.0000
   -1.5000
    1.0000
    0.7500
   -1.0000
Sparse matrix solution:
xs =
   (1,1)        0.5000
   (2,1)        2.0000
   (3,1)       -0.5000
   (5,1)       -1.5000
   (6,1)        1.0000
   (7,1)        0.7500
   (8,1)       -1.0000
Workspace contents after the solutions:
  Name         Size                    Bytes  Class
  a            8x8                       512  double array
  as           8x8                       276  double array (sparse)
  b            8x1                        64  double array
  bs           8x1                       104  double array (sparse)
  x            8x1                        64  double array
  xs           8x1                        92  double array (sparse)
Grand total is 115 elements using 1112 bytes
```

两种算法得到了相同的答案。注意用稀疏矩阵产生的结果不包含 x_4，因为它的值为 0。注意 b 的稀疏形式占的内存空间比全矩阵形式还要大。这种情况是因为稀疏矩阵除了元素值之外必须存储它的行号和列号，所以当一个矩阵的大部分元素都是非零元素，用稀疏矩阵将降低运算效率。

7.2 单元阵列(cell array)

单元阵列是 **MATLAB** 中特殊一种数组，它的元素被称为单元(cells)，它可以存储其它类型的 **MATLAB** 数组。例如，一个单元阵列的一个单元可能包含一个实数数组或字符型数组，还可能是复数组(图 7.1 所示)。

在一个编程项目中，一个单元阵列的每一个元素都是一个指针，指向其他的数据结构，而这些数据结构可以是不同的数据类型。单元阵列为选择问题信息提供极好的方示，因为所有信息都聚集在一起，并可以通边一单个名字访问。单元阵列用大括号{}替代小括号来选择和显示单元的内容。这个不同是由于单元的内容用数据结构代替了内容。假设一单元阵列如图 7.2 所示。元素 a(1，1)是数据结构 3×3 的数字数组。a(1，1)的含义为显示这个单元的内容，它是一个数据结构。

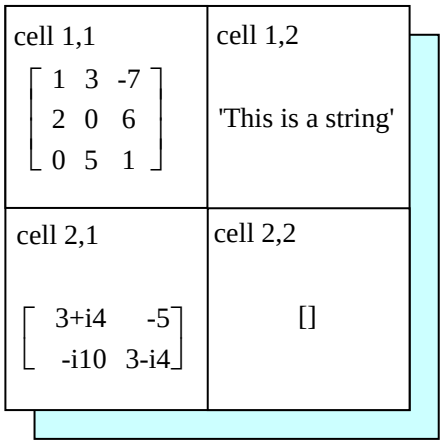

图 7.1 一个单元阵列的一个单元可能包含一个实数数组或字符型数组，还可能是复数组

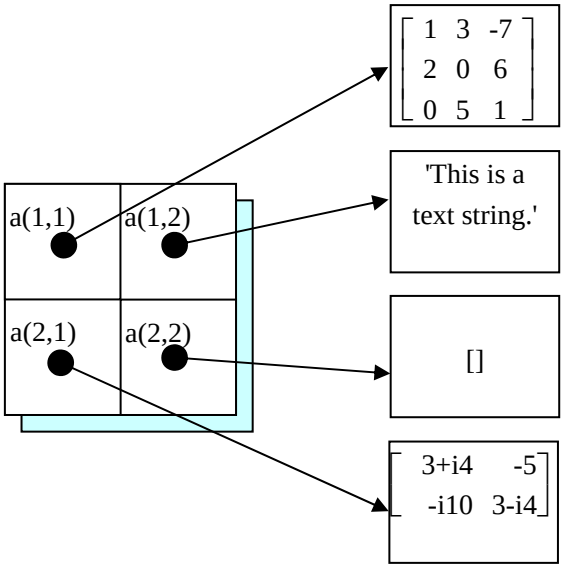

图 7.2 单元阵列中的每一个元素都是指向其他数据结构的指针，指向的数据结构可能都不相同

```
>> a(1,1)
ans =
```

```
    [3x3 double]
```

相对地，a{1,1}的含义为显示这个数据结构的内容。

```
>> a{1,1}
ans =
     1     3    -7
     2     0     6
     0     5     1
```

总起来说，标识 a{1,1}反映的是数据结构 a(1,1)内容，而标识 a(1,1)是一个数据结构。

好的编程习惯

当你访问一单元阵列时，不要把()与{}混淆。它们完全不同的运算。

7.2.1 创建单元阵列

创建单元阵列有两种方法

- 用赋值语句
- 用函数 cell 创建

最简单的创建单元阵列的方法是直接把数据结构赋值于独立的单元，一次赋一个单元。但用 cell 函数创建将会更加地高效，所以我们用 cell 创建大的单元数组。

7.2.1.1 用赋值语句创建单元阵列

你可以用赋值语句把值赋于单元阵列的一个单元，一次赋一个单元。这里有两种赋值的方法，即内容索引(content indexing)和单元索引(cell indexing)。

内容索引要用到大括号{}，还有它们的下标，以及单元的内容。例如下面的语句创建了一个 2×2 的单元阵列，如图 7.2 所示。

```
a{1,1} = [1 3 -7; 2 0 6; 0 5 1];
a{1,2} = 'This is a text string.';
a{2,1} = [3+4*i -5; -10*i 3-4*i];
a{2,2} = [];
```

索引的这种类型定义了包含在一个单元中的数据结构的内容。

单元索引把存储于单元中的数据用大括号括起来，单元的下标用普通下标标记法。例如下面的语句将创建一个 2×2 的单元阵列，如图 7.2 所示。

```
a(1,1) ={[1 3 -7; 2 0 6;0 5 1]};
a(1,2) = {'This is a text string.'};
a(2,1) = {[3+4*i -5; -10*i 3-4*i]};
a(2,2) = {[]};
```

索引的这种类型创建了包含有指定值的一个数据结构，并把这个数据结构赋于一个单元。

这两种形式是完全等价的，你可以在你的程序任选其一。

常见编程错误

不要创建一个与已存在的数字数组重名的元阵列。如果得名了，**MATLAB** 会认为你把单元阵列的内容赋值给一个普通的数组，这将会产生一个错误信息。在创建单元阵列之前，确保同名的数字数字数组已经被删除。

7.2.1.2 用 cell 函数创建单元阵列

函数 cell 允许用户创建空单元阵列，并指定阵列的大小。例如，下面的语句创建一个 2×2 的空单元阵列。

```
a = cell(2, 2)
```

一旦单元阵列被创立，你就可以用赋值语句对单元阵列进行赋值。

7.2.2 单元创建者——大括号({})的应用

如果在单个大括号中列出所有单元的内容，那么就定义了许多的单元，在一行中的独立单元用逗号分开，行与行之间用分号隔开。例如下面的语句创建一个 2×3 单元阵列。

```
b = {[1 2], 17, [2;4]; 3-4*i, 'Hello', eye(3)}
```

7.2.3 查看单元阵列的内容

MATLAB 可以把单元阵列每一个元素的数据结构缩合在一行中显示出来。如果全部的数据结构没有被显示出来，那么显示就是一个总结。例如，单元阵列 a 和 b 显示如下

```
>> a
a =
    [3x3 double]    [1x22 char]
    [2x2 double]                []
>> b
b =
            [1x2 double]    [    17]    [2x1 double]
    [3.0000- 4.0000i]    'Hello'    [3x3 double]
```

注意 **MATLAB** 显示的只是数据结构，包括中括号和省略号，而不包含数据结构的内容。

如果你想要知道看到单元阵列的所有内容，要用到 celldisp 函数。这个函数显示的是每一个单元中的数据结构的内容。

```
>> celldisp(a)
a{1,1} =
     1     3    -7
     2     0     6
     0     5     1
a{2,1} =
   3.0000 + 4.0000i   -5.0000
             0 -10.0000i    3.0000 - 4.0000i
a{1,2} =
This is a text string.
a{2,2} =
     []
```

如果要用高质量的图象显示数据结构的内容，要用到函数 cellplot。例如，函数 cellplot(b) 产生了一个图象，如图 7.3 所示。

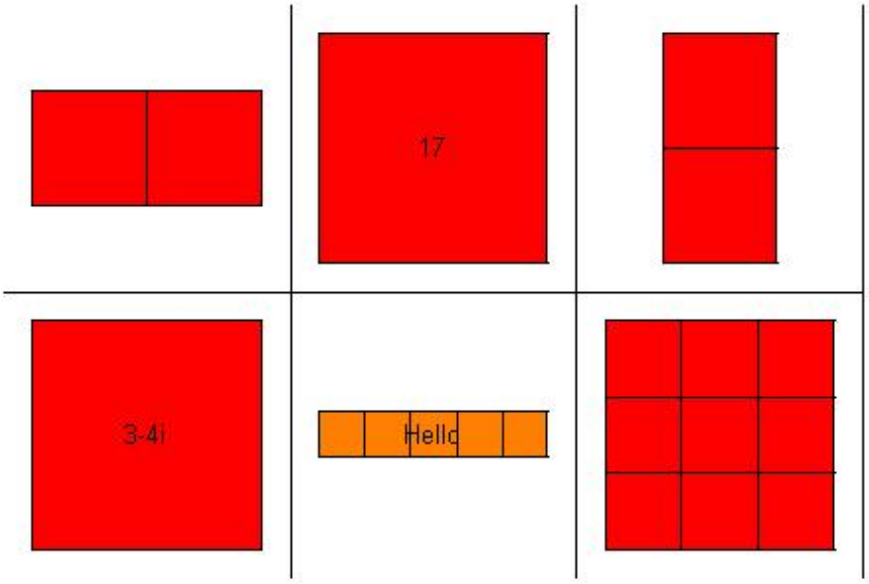

图 7.3 用函数 cellplot 显示单元阵列 b 数据结构的内容

7.2.4 对单元阵列进行扩展

一个值赋值于一个单元阵列中的元素，如果这个元素现在不存在，那么这个元素就会被自动的建立，其他所需的元素也会被自动建立。例如，假设定义了一个 2×2 单元阵列，如图 7.1 所示。如果我们执行下面的语句

a{3, 3} = 5

单元阵列将会自动扩展为 3×3 单元阵列，如图 7.4 所示。

cell 1,1 $\begin{bmatrix} 1 & 3 & -7 \\ 2 & 0 & 5 \\ 0 & 5 & 1 \end{bmatrix}$	cell 1,2 'This is a text string.'	cell 1,3 []
cell 2,1 $\begin{bmatrix} 3+i4 & -5 \\ -i10 & 3-i4 \end{bmatrix}$	cell 2,2 []	cell 2,3 []
cell 3,1 []	cell 3,2 []	cell 3,3 [5]

图 7.4 把一个值赋值于 a(3,3)产生的结果。注意其他的空元素也是自动创建的。

7.2.5 删除阵列中的元素

如果要删除阵列中的所有元素，我们要用 clear 命令。如果要删除单元阵列中的部分元素，我们把空值赋值于这一部分元素。例如，假设 a 的定义如下

```
>> a
a =
    [3x3 double]    [1x22 char]      []
    [2x2 double]          []         []
              []          []       [5]
```

我们可以用下面的语句删除第三行

```
>> a(3,:)=[]
a =
    [3x3 double]    [1x22 char]      []
    [2x2 double]          []         []
```

7.2.6 单元阵列数据的应用

在一个单元阵列中，数据结构中数据可以随时用内容索引或单元索引调用。

例如假设单元阵列 c 的定义如下

```
c = {[1 2; 3 4],'dogs';'cats',i}
```

存储于 c(1,1)的内容可由下面的语句调用

```
>> c{1,1}
ans =
     1     2
     3     4
```

同样 c(2,1)中的元素可由下面的元素调用

```
>> c{2,1}
ans =
cats
```

一个单元内容的子集可由两套下标得到。例如，假设我们要得到单元 c(1,1)中的元素(1,2)。为了达到此目的，我们可以用表达式 c{1,1}(1,2)，它代表单元 c(1,1)中的元素(1,2)。

```
>> c{1,1}(1,2)
ans =
     2
```

7.2.7 字符串单元阵列

在一个单元阵列中存储一批字符串与在标准的字符数组中存储相比是非常方便的，因为在单元阵列中每一个字符串的长度可以是不相同的，而在标准字符数组的每一行的长度都必须相等。这就意味着在单元阵列中的字符串没的必要增加多余的空格。许多的 **MATLAB** 用户图形界面函数均使用单元阵列，正是基于这个原因，我们将在第十章看到。

字符串单元阵列可以由两种方法创建。我们可以用方括号把独立的字符串插入到单元阵列，我们也可以函数 cellstr 把一个二维字符数组转化为相应的字符串单元阵列。

下面的例子用第一种方法创建了一个字符串单元阵列，并显示出这个阵列的结果。注意下面的每一个字符串具有不同的长度。

```
>> cellstring{1} = 'Stephen J. Chapman';
>> cellstring{2} = 'Male';
>> cellstring{3} = 'SSN 999-99-9999';
>> cellstring
```

```
cellstring =
    'Stephen J. Chapman'    'Male'    'SSN 999-99-9999'
```

我们可以利用函数 cellstr 把一个二维字符数据转化为相应的字符串单元阵列。考虑下面的字符数组。

```
>> data = ['Line 1              ';'Additional Line']
data =
Line 1
Additional Line
```

相应的字符串单元阵列为

```
>> c = cellstr(data)
c =
    'Line 1'
    'Additional Line'
```

我们还可以用 char 函数它转化回去

```
>> newdata = char(c)
newdata =
Line 1
Additional Line
```

7.2.8 单元阵列的重要性

单元阵列是非常灵活的，因为各种类型的大量数据可以存储在每一个单元中。所以，它经常当作中间 **MATLAB** 数据结构来用。我们必须理解它，因为在第十章中 **MATLAB** 图形用户界面要用到它的许多特性。

还有，单元阵列的灵活性可能使它们具有函数普通特性，这个函数是指带有输入参数和输出参数的变量个数的函数。一种特别的输入参数 varargin 可以在自定义函数中得到，这种函数支持输入参数的变量的个数。这个参数显在输入参数列表的最后一项，它返回一个单元阵列，所以一个输入实参可以包括任意数目的实参。每一个实参都变成了由 varagin 返回的单元阵列元素。如果它被应用，varagin 必须是函数中的最后一个输入参数。

例如，假设我们要编写一个函数，它可能需要任意个数的输入参数。这个函数执行如下所示

```
function test1(varargin)
disp(['There are ' int2str(nargin) ' arguments.']);
disp('The input arguments are:');
disp(varargin);
```

我们用不同的数目参数来执行这个函数，结果如下

```
>> test1
There are 0 arguments.
The input arguments are:
>> test1(6)
There are 1 arguments.
The input arguments are:
    [6]
>> test1(1,'test 1',[1 2,3 4])
There are 3 arguments.
The input arguments are:
    [1]    'test 1'    [1x4 double]
```

正如我们所看到的，参数变成了函数中的单元阵列元素。

下面是一个简单函数例子，这个函数拥有不同的参数数目。函数 plotline 任意数目的 1×2 行向量，每一个向量包含一个点(x,y)。函数把这些点连成线。注意这个函数也接受直线类型字符串，并把这些字符串转递给 plot 的函数。

```
function plotline(varargin)
```

```
%PLOTLINE Plot points specified by [x,y] pairs.
% Function PLOTLINE accepts an arbitrary number of
% [x,y] points and plots a line connecting them.
% In addition, it can accept a line specification
% string, and pass that string on to function plot.
% Define variables:
% ii                --Index variable
% jj                --Index variable
% linespec          --String defining plot characteristics
% msg               --Error message
% varargin          --Cell array containing input arguments
% x                 --x values to plot
% y                 --y values to plot
% Record of revisions:
% Date          Programmer      Description of change
% ====          ==========      =====================
% 10/20/98 S. J. Chapman        Original code
% Check for a legal number of input arguments.
% We need at least 2 points to plot a line...
msg = nargchk(2,Inf,nargin);
error(msg);
% Initialize values
jj = 0;
linespec = '';
% Get the x and y values, making sure to save the line
% specification string, if one exists.
for ii = 1:nargin
    % Is this argument an [x,y] pair or the line
    % specification?
    if ischar(varargin{ii})
        % Save line specification
        linespec = varargin{ii};
    else
        % This is an [x,y] pair. Recover the values.
        jj = jj + 1;
        x(jj) = varargin{ii}(1);
        y(jj) = varargin{ii}(2);
    end
end
% Plot function.
if isempty(linespec)
    plot(x,y);
else
    plot(x,y,linespec);
end
```

我们用下面的参数调用这个函数，产生的图象如图 7.5 所示。用相同的数目的参数调用函数，看它产生的结果为什么？

也有专门的输出参数，vargout，它支持不同数目的输出参数。这个参数显示在输出参数列表的最后一项。它返回一个单无阵列，所示单个输出实参支持任意数目的实参。每一个实参都是这个单无阵列的元素，存储在 varargout。如果它被应用，varargout 必须是输出参数列表中最后一项，在其它输入参数之后。存储在 varargout 中的变量数由函数 nargout 确定，这个函数用指定于任何一个已知函数的输出实参。例如，我们要编写一函数，它返回任意数目的随机数。我们的函数可以用函数 nargout 指定输出函数的数目，并把这些数目存储在单元阵列 varargout 中。

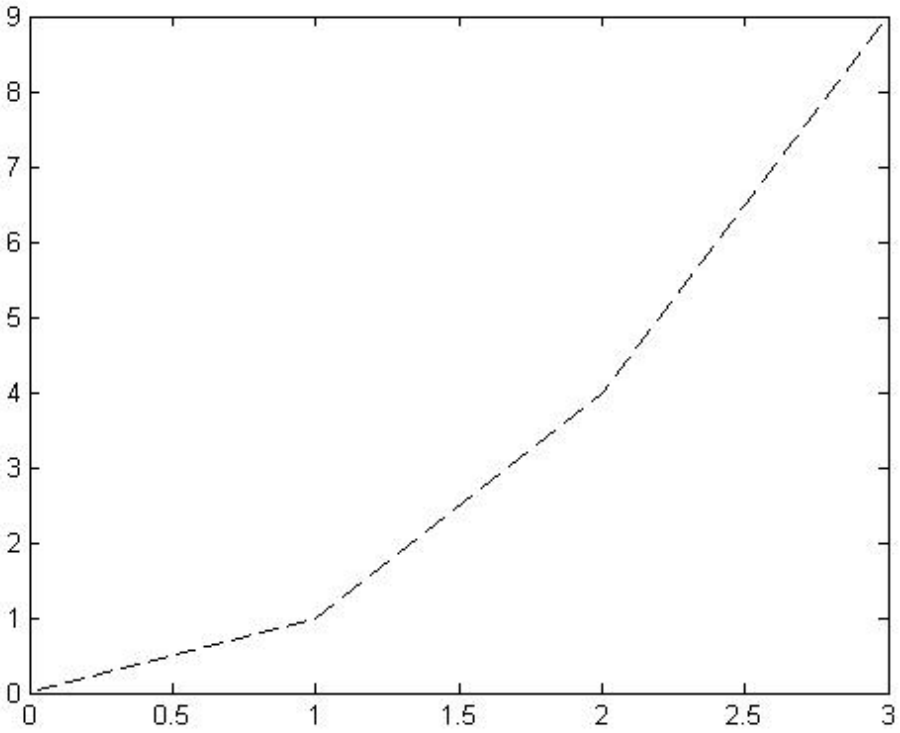

图 7.5 函数 plotline 产生的图象

```
function [nvals, varargout] = test2(mult)
%      nvals is the number of random values returned
%      varargout contains the random values returned
nvals = nargout - 1;
for ii = 1:nargout-1
       varargout{ii} =randn * mult;
end
```

当这个函数被执行时，产生的结果如下

```
>> test2(4)
ans =
       -1
>> [a b c d] = test2(4)
a =
        3
b =
    -1.7303
c =
    -6.6623
d =
       0.5013
```

好的编程习惯

应用单元阵列 varargin 和 varargout 创建函数，这个函数支持不同数目的输入或输出参数。

7.2.9 单元阵列函数总结

支持单元阵列的一些普通函数总结在表 7.2 中。

表 7.2 普通的单元阵列函数

函数	描述
cell	对单元阵列进行预定义
celldisp	显示出单元阵列的内容
cellplot	画出单元阵列的结构图
cellstr	把二维字符数组转化为相应的字符串单元阵列
char	把字符串单元阵列转化相应的字符数组

7.3 结构数组

一个数组是一个数据类型，这种数组类型有一个名字，但是在这个数组中的单个元素只能通过已知的数字进行访问。数组 arr 中的第五个元素可由 arr(5)访问。注意在这个数组中的所有元素都必须是同一类型(数字或字符)。一个单元阵列也是一种数据类型，也有一个名字，单个元素也只能通过已知的数字进行访问。但是这个单元阵列中元素的数据类型可以是不同的。相对地，一个**结构**也是一种数据类型，它的每一个元素都有一个名字。我们称结构中的元素为**域**。单个的域可以通过结构名和域名来访问，用句号隔开。

7.3.1 创建结构

创建结构有两种方法

- 用赋值语句创建
- 用函数 struct 函数进行创建

7.3.1.1 用赋值语句创建函数

你可以用赋值语句一次创建一个结构域。每一次把数据赋值于一个域，这个域就会被自动创建。例如用下面的语句创建如图 7.6 所示的结构。

```
>> student.name = 'John Doe';
>> student.addr1 = '123 Main Street';
>> student.city = 'Anytown';
>> student.zip = '71211'
student =
        name: 'John Doe'
       addr1: '123 Main Street'
        city: 'Anytown'
         zip: '71211'
```

第二个 student 可以通过在结构名前加上一个下标的方式加入到这个结构中。

```
>> student(2).name = 'Jane Q. Public'
student =
1x2 struct array with fields:
      name
      addr1
      city
      zip
```

Student 现在是一个 1×2 数据。注意当一个结构数据超一个元素，只有域名，而没有它的内容。在命令窗口中独立键入每一个元素，它的内容就会被列出。

```
>> student(1)
ans =
        name: 'John Doe'
       addr1: '123 Main Street'
        city: 'Anytown'
         zip: '71211'
>> student(2)
ans =
        name: 'Jane Q. Public'
       addr1: []
        city: []
         zip: []
```

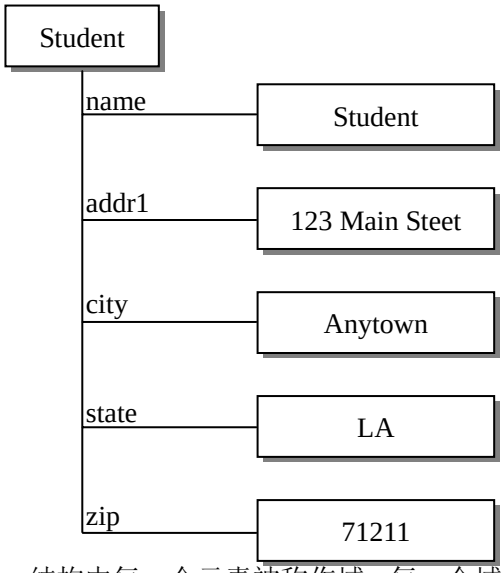

图 7.6 一个简单的例子。结构中每一个元素被称作域，每一个域都可以通过它的名字来访问。

注意无论结构的元素什么时间被定义，结构中的所有域都会被创建，不管它是否被初始化，没有被初始化的域将包含一个空数组，在后面我们可以用赋值语句来初始化这个域。

我们可以用 fieldnames 函数随时对结构中的域名进行恢复。这个函数可以返回一系列字符单元阵列中的域名，这个函数对结构数组的运算是非常有用的。

7.3.1.2 用 struct 函数创建结构

函数 struct 允许用户预分配一个结构数据。它的基本形式如下

structure_array = struct(fields)

其中 fields 是填补结构域名的字符串数组或单元阵列。用上面的语法，函数 struct 用空矩阵初始化数组中的每一个域。

当域被定义时，我们就可以对它进行初始化，形式如下

str_array = struct('field1', var1, 'field2', val2, ...)

其中参数为域名和它们的值。

7.3.2 增加域到结构

如果一个新的域名在结构数组中的任意一个元素中被创建，那么这个域将会增加到数组的所有元素中去。例如，假设我们把一些成绩添加到 jane public 的记录中。

```
>> student(2).exams = [90 82 88]
```

```
student =
1x2 struct array with fields:
    name
    addr1
    city
    zip
    exams
```

如下所示，在数组的每一个记录中都有一个 exams 域。这个将会在 student(2)中进行初始化，其他同学的数组为空，除非用赋值语句给他赋值。

```
>> student(1)
ans =
     name: 'John Doe'
    addr1: '123 Main Street'
     city: 'Anytown'
      zip: '71211'
    exams: []
>> student(2)
ans =
     name: 'Jane Q. Public'
    addr1: []
     city: []
      zip: []
    exams: [90 82 88]
```

7.3.3 删除结构中的域

我们可以用 rmfield 函数删除结构数据中的域。这个函数的形式如下

```
struct2 = rmfield(struct_array, 'field')
```

其中 struct_array 是一个结构数组，field 是要去除的域，stuct2 是得到的新结构的名字。例如从结构 student 中去除域名 zip，代码如下

```
>> stu2 = rmfield(student, 'zip')

stu2 =

1x2 struct array with fields:
    name
    addr1
    city
    exams
```

7.3.4 结构数组中数组的应用

现在我们假设结构 sdudent 中已经有三个学生，所有数据如图 7.7 所示。我们如何应用结构数组中的数据呢？

我们可以在句号和域名后加数组元素名访问任意数组元素的信息。

```
>> student(2).addr1
ans =
P.O.Box 17
>> student(3).exams
ans =
    65    84    81
```

任何带一个域的独立条目可以在域名后加下标的方式进行访问，例如，第三个学生的第

二个科目的成绩为

```
>> student(3).exams(2)
ans =
     84
```

结构数组中的域能作为支持这种数组类型任意函数的参数。例如，计算 student(2)的数据平均值，我们使用这个函数

```
>> mean(student(2).exams)
ans =
   86.6667
```

不幸是的，我们不能同时从多个元素中得到一个域的值。例如，语句

student.name 是无意义的，并导致错误。如果我们想得到所有学生的名字，我们必须用到一个 for 循环。

```
for ii = 1:length(student)
    disp(student(ii).name);
end
```

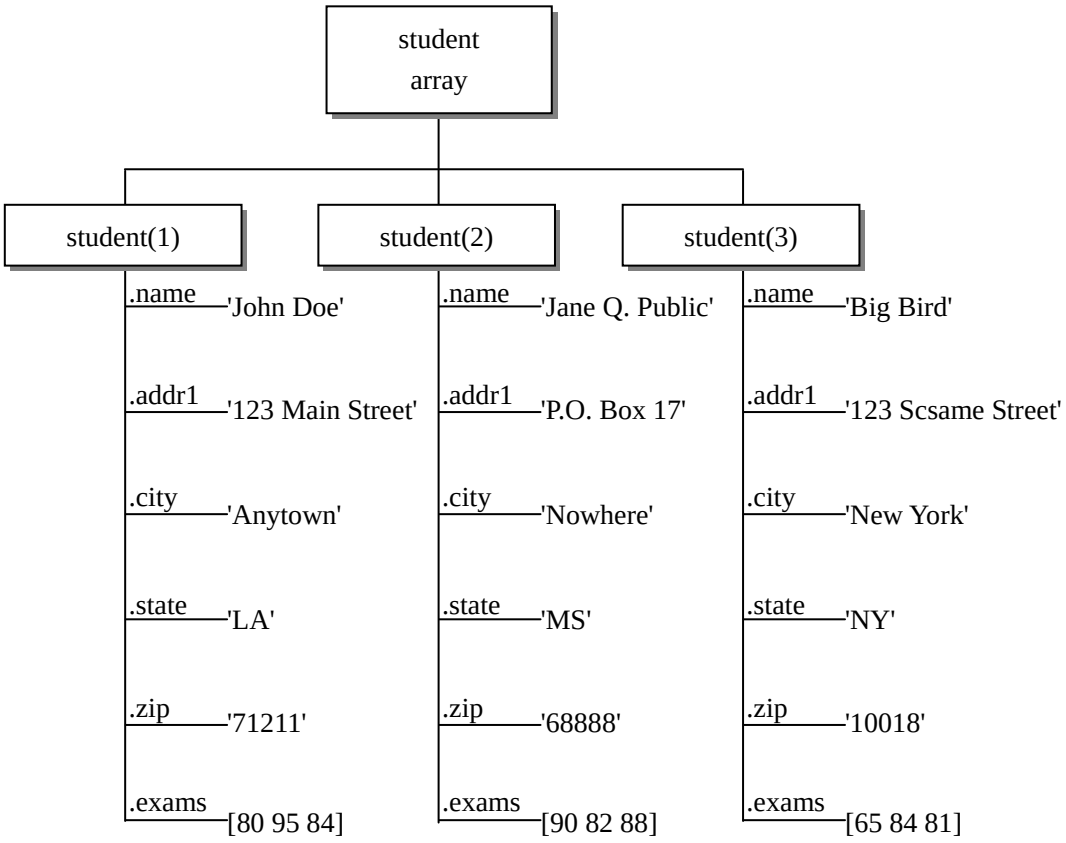

图 7.7 用三个元素组成的 student 数组和它所有的域

相似地，如果我们想得到所有学生的平均成绩，我们不能用函数 mean(student.exams)，我们必须独立的访问学生的每一个成绩，并算出总成绩。

```
exam_list = [];
for ii = 1:length(student)
    exam_list = [exam_list student(ii).exams];
end
mean(exam_list)
```

7.3.5 函数 getfield 和函数 setfield

MATLAB 中有两个函数用来创建结构，这比用自定义函数创建要简单的多。函数 getfield 得到当前存储在域中的值，函数 setfield 在域中插入一新值。

getfield 函数的结构是

f = getfield(array,{array_index},'field',{field_index})

field_index 和 array_index 是可选择性，array_index 用于创建 1×1 结构数组。调用这个函数的语句为

f = array(array_index).field(field_index);

当编写程序时，尽管程序不知道结构数组中的域名，这个语句也可以应用。例如，假设我们需要编写一个程序，读取一未知数组并对它进行操作。函数可以通过调用 fieldnames 命令来确定一个结构数据的域名，然后通过函数 getfield 读取数据。为了读取第二个学生的 zip 码，函数为

```
>> zip = getfield(student,{2},'zip')
zip =
68888
```

相似地，我们可以用 setfield 函数修改结构的值。形式如下

f = setfield(array,{array_index},'field',{field_index},value)

f 是输出结构数组，field_index 和 array_index 都是可选择性参数，array_index 用于创建 1×1 结构数组。调用这个函数的语句为

array(array_index).field(field_index) = value;

7.3.6 对结构数组应用 size 函数

当 size 函数应用于结构数组时，它将返回这个结构数组的大小。当 size 函数应用于结构数组中的一个元素的域时它将返回这个域的大小。例如

```
>> size(student)
ans =
     1     3
>> size(student(1).name)
ans =
     1     8
```

7.3.7 结构的嵌套

结构数组的每一个域可是任意数据类型，包括单元阵列或结构数组。例如，下面的语句定义了一个新结构数组。它作为 student 的一个域，用来存储每一个学生所在班级的信息。

```
>> student(1).class(1).name = 'COSC 2021';
>> student(1).class(2).name = 'PHYS 1001';
>> student(1).class(1).instructor = 'Mr. Jones';
>> student(1).class(2).instructor = 'Mrs. Smith';
```

在这个语句被执行后，student(1)将由以下数据组成。注意访问嵌套结构中的数据方法。

```
>> student(1)
ans =
        name: 'John Doe'
      addr1: '123 Main Street'
        city: 'Anytown'
      state: 'LA'
         zip: '71211'
      exams: [80 95 84]
      class: [1x2 struct]
>> student(1).class
```

```
ans =
1x2 struct array with fields:
    name
    instructor
>> student(1).class(1)
ans =
            name: 'COSC 2021'
    instructor: 'Mr. Jones'
>> student(1).class(2)
ans =
            name: 'PHYS 1001'
    instructor: 'Mrs. Smith'
>> student(1).class(2).name
ans =
PHYS 1001
```

7.3.8 struct 函数总结

支持 struct 的普通函数总结在表 7.3 中。

表 7.3 支持 struct 的函数

函数	描述
fieldnames	在一个字符串单元阵列中返回域名
getfield	从一个域中得到当前的值
rmfield	从一个结构中删除一个域
setfield	在一个域中设置一个新值
struct	预定义一个结构

测试 7.1

本测试提供了一个快速的检查方式，看你是否掌握了本章的基本内容。如果你对本测试有疑问，你可以重读本章，问你的老师，或和同学们一起讨论。在附录 B 中可以找到本测试的答案。

1. 什么是稀疏矩阵？它和全矩阵有何区别？两者之间如何相互转化？
2. 什么是单元阵列？它和普通的数组有何区别？
3. 内容检索和单元检索有何区别？
4. 什么是结构？它与稀疏矩阵，单元阵列有什么区别？
5. varargin 的功能是什么？它是如何工作的？
6. 下面给出了数组 a 的定义，下面的语句将会产生怎样的结果？

```
a{1,1} = [1 2 3; 4 5 6; 7 8 9];
a(1,2) = {'Comment line'};
a{2,1} = j;
a{2,2} = a{1,1} -a{1,1}(2,2);
```

(a) a(1,1)　(b) a{1,1}　(c) 2*a(1,1)　(d) 2*a{1,1}
(e) a{2,2}　(f) a(2,3) = {[-17;17]}　(g) a{2,2}(2,2)

7. 下面给出了结构 b 的定义，下面的语句将会产生怎样的结果？

```
b(1).a = -2*eye(3);
b(1).b = 'Element 1';
b(1).c = [1 2 3];
```

```
b(2).a = (b(1).c' [-1; -2; -3] b(1).c');
b(2).b = 'Element 2';
b(2).c = [1 0 -1];
```

(a) b(1).a - b(2).a　　(b) strncmp(b(1).b, b(2).b,6)
(c) mean(b(1).c)　　(d) mean(b.c)
(e) b　　(f) b(1)

7.4 总结

本章重点介绍了三类数据类型：稀疏矩阵，单元阵列和结构。

7.4.1 好的编程习惯总结

当你访问一单元阵列时，不要把()与{}混淆。它们完全不同的运算。

7.4.2 MATLAB 函数命令总结

普通的 **MATLAB** 稀疏矩阵函数

类别	函数	描述
创建一个稀疏矩阵	speye	创建一个单位稀疏矩阵
	sprand	创建一个稀疏矩阵，元素是符合平均分布的随机数
	sprandn	创建一个稀疏矩阵，元素是普通的随机数
全矩阵和稀疏矩阵的转换函数	sparse	把一个全矩阵转化为一个稀疏矩阵
	full	把一个稀疏矩阵转化为全矩阵
	find	找出矩阵中非零元素和它对应的上下标
对稀疏矩阵进行操作的函数	nnz	非零元素的个数
	nonzeros	返回一个向量，其中的元素为矩阵中非零元素
	spones	用 1 代替矩阵中的非零元素
	spalloc	一个稀疏矩阵所占的内存空间
	issparse	如果是稀疏矩阵就返回 1
	spfun	给矩阵中的非零元素提供函数
	spy	用图象显示稀疏矩阵

普通的单元阵列函数

函数	描述
cell	对单元阵列进行预定义
celldisp	显示出单元阵列的内容
cellplot	画出单元阵列的结构图
cellstr	把二维字符数组转化为相应的字符串单元阵列
char	把字符串单元阵列转化相应的字符数组

支持 struct 的函数

函数	描述
fieldnames	在一个字符串单元阵列中返回域名
getfield	从一个域中得到当前的值
rmfield	从一个结构中删除一个域
setfield	在一个域中设置一个新值
struct	预定义一个结构

7.5 练习

7.1

编写一个 **MATLAB** 函数，可以接爱一个字符串单元阵列，并根据 ascii 码字母顺序对它进行升序排列。(如果你愿意的话，可以利用第六章的函数 c_strcmp 对它们进行比较。)

7.2

编写一个 **MATLAB** 函数，接受一个字符串单元阵列，并按字母表的顺序进行排序。(注意在这里不区分大小写)

7.3

创建一个 100×100 的稀疏矩阵，其中 5%的元素是按普通分布的随机数(用 sprandn 产生这些值)，其余为 0。下一步，把数组对角线上的所有元素都设置为 1。下一步，定义一个含 100 个元素稀疏列向量 b，并用 100 个符合平均分布的随机数赋值于 b。回答下面的问题。

a. 利用稀疏矩阵 a 创建一个全矩阵 a_full。比较两矩阵所需的内存？那一个更高效呢？
b. 应用 spy 函数画出 a 中元素的分布
c. 利用稀疏矩阵 b 创建一个全矩阵 b_full。比较两矩阵所需的内存？那一个更高效呢？
d. 分别用全矩阵和稀疏矩阵角方程组 a*x=b 中未知矩阵 x 的值？

7.4

创建一个函数，接受任意数目数字输入参数，计算出所有参数中单个元素的总和。用下面 4 个参数检测你的程序

a = 10， $b=\begin{bmatrix}4\\-2\\2\end{bmatrix}$，$c=\begin{bmatrix}1&0&3\\-5&1&2\\1&2&0\end{bmatrix}$，d = [1 5 -2]。

7.5

修改先前练习中的函数，使它既能够接受数字数组又能接受包含数字值的单元阵列。用下面 2 组参数检测你的程序。

a = $\begin{bmatrix} 1 & 4 \\ -2 & 3 \end{bmatrix}$，b{1} = [1 5 2]，和 b{2} = $\begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix}$

7.6

创建一个结构，画图的所有信息，在取最小值时，结构数据应当有下面的域

x_data	x 的值
y_data	y 的值
type	线性，对数等
plot_title	图象标题
x_label	x 轴标签

你也可以增加其他的域来增强你对最终图象的控制。

在结构数据被创建后，创建一个函数，它能接受包含有这个结构的数组，在这个数组中每一个结构产生一个图象。这个函数应当支持一些默认的域值。例如，如果域 plot_title 是一个空矩阵，那么产生的图象将无标题材。在编写程序之前，想好你的默认值。

为了检测你的函数，创建一个结构，包括创建三个不同图形的数据，并把这个结构传递给你的函数。这个函数应当正确的画出三个图形。