**16矩阵操作**

有许多函数可用于检查矩阵的元素是否满足某些条件，以及重新排列矩阵的元素。例如，Octave可以很容易地告诉你一个矩阵的所有元素是有限的，还是小于某个指定的值。Octave还可以旋转元素，提取上或下三角形部分，或对矩阵的列进行排序。

**16.1查找元素及检查条件**

函数any和all对于确定一个矩阵的任何元素或所有元素是否满足某些条件是有用的。find函数在确定矩阵的哪些元素满足指定条件时也很有用。

: tf = any (x)

: tf = any (x, dim)

对于vector参数，如果vector的任何元素非零，则返回true(逻辑1)。

对于矩阵参数，返回一个由逻辑1和0组成的行向量，每个元素表示矩阵对应列的任何元素是否为非零。例如:

any (eye (2, 4))

⇒ [ 1, 1, 0, 0 ]

如果提供了可选参数dim，则沿着维度dim工作。例如:

any (eye (2, 4), 2)

⇒ [ 1; 1 ]

参见:all。

: tf = all (x)

: tf = all (x, dim)

对于vector参数，如果vector的所有元素都非零，则返回true(逻辑1)。

对于矩阵参数，返回一个由逻辑1和0组成的行向量，每个元素表示矩阵对应列的所有元素是否都是非零的。例如:

all ([2, 3; 1, 0])

⇒ [ 1, 0 ]

如果提供了可选参数dim，则沿着维度dim工作。

参见:any。

由于比较运算符(请参阅比较运算符)返回由1和0组成的矩阵，因此很容易测试矩阵的许多内容，而不仅仅是元素是否为非零。例如,

all (all (rand (5) < 0.9))

⇒ 0

测试一个随机的5 × 5矩阵，看它的所有元素是否都小于0.9。

注意，在条件上下文中(如if和while语句的test子句)，Octave将测试视为您键入了all (all (condition))。

: z = xor (x, y)

: z = xor (x1, x2, …)

返回x和y的独占或。

对于布尔表达式x和y，当且仅当x或y中的一个为真，xor (x, y)为真。否则，如果x和y都为真或都为假，则xor返回假。

xor运算的真值表是

x y z

- - -

0 0 0

1 0 1

0 1 1

1 1 0

如果给定两个以上参数，则从左到右累计应用异或操作:

(…((x1 XOR x2) XOR x3) XOR …)

参见and, or, not。

: y = diff (x)

: y = diff (x, k)

: y = diff (x, k, dim)

如果x是长度为n的向量，则diff (x)是首差的向量x(2) - x(1)，…，x(n) - x(n-1)。

如果x是一个矩阵，则diff (x)是沿第一个非单维的列差矩阵。

第二个参数是可选的。如果提供了diff (x, k)，其中k是一个非负整数，则返回第k个差值。k有可能大于矩阵的第一个非单维。在这种情况下，diff继续沿着下一个非单例维取差值。

可以使用可选变量dim显式地声明取差的维度。在这种情况下，沿着这个维度计算k阶差。在k超过size (x, dim)的情况下，返回一个空矩阵。

参见:排序，合并。

: tf = isinf (x)

返回一个逻辑数组，当x的元素为无穷大时为真，当元素为非无穷大时为假。

例如:

isinf ([13, Inf, NA, NaN])

⇒ [ 0, 1, 0, 0 ]

参见:isfinite, isnan, isna。

: tf = isnan (x)

返回一个逻辑数组，如果x的元素是NaN值，则为true，如果不是NaN值，则为false。

NA值也被视为NaN值。例如:

isnan ([13, Inf, NA, NaN])

⇒ [ 0, 0, 1, 1 ]

参见:isna, isinf, isfinite。

: tf = isfinite (x)

返回一个逻辑数组，当x的元素是有限值时为真，当元素不是有限值时为假。

例如:

isfinite ([13, Inf, NA, NaN])

⇒ [ 1, 0, 0, 0 ]

参见:isinf, isnan, isna。

: [err, yi, …] = common\_size (xi, …)

确定所有输入参数是标量还是普通大小。

如果为真，err为零，并且yi是一个公共大小的矩阵，如果这是一个标量，则所有元素都等于xi，否则为xi。如果输入不能达到统一的大小，err为1,yi为xi。例如:

[err, a, b] = common\_size ([1 2; 3 4], 5)

⇒ err = 0

⇒ a = [ 1, 2; 3, 4 ]

⇒ b = [ 5, 5; 5, 5 ]

这对于实现参数既可以是标量也可以是普通大小的函数很有用。

参见:size, size\_equal, numel, ndims。

: idx = find (x)

: idx = find (x, n)

: idx = find (x, n, direction)

: [i, j] = find (…)

: [i, j, v] = find (…)

返回矩阵中非零元素的下标向量，如果x是行向量则为行，否则为列。

为了获得每个矩阵元素的单个索引，Octave假定矩阵的列形成一个长向量(类似于Fortran数组的存储方式)。例如:

find (eye (2))

⇒ [ 1; 4 ]

如果给定两个输入，则n表示从矩阵或向量的开头找到的最大元素数。

如果给出了三个输入，方向应该是“第一”或“最后”之一，分别只请求第一个或最后n个指标。但是，索引总是按升序返回。

如果请求两个输出，find返回矩阵中非零元素的行和列索引。例如:

[i, j] = find (2 \* eye (2))

⇒ i = [ 1; 2 ]

⇒ j = [ 1; 2 ]

如果请求三个输出，find也返回一个包含非零值的向量。例如:

[i, j, v] = find (3 \* eye (2))

⇒ i = [ 1; 2 ]

⇒ j = [ 1; 2 ]

⇒ v = [ 3; 3 ]

如果x是一个大小为m x n x p x…的多维数组，j包含列的位置，就好像x被平铺成一个大小为m x (n + p +…)的二维矩阵。

请注意，这个函数对于稀疏矩阵特别有用，因为它将非零元素提取为向量，然后可以使用这些向量创建原始矩阵。例如:

sz = size (a);

[i, j, v] = find (a);

b = sparse (i, j, v, sz(1), sz(2));

参见:非零。

: idx = lookup (table, y)

: idx = lookup (table, y, opt)

查找已排序表中的值。

这个函数通常用作插值的前奏。

如果表是递增的，长度为N且idx = lookup (table, y)，那么对于表中所有的y(i)， table(idx(i)) <= y(i) < table(idx(i+1))。如果y(i) < table(1)，则idx(i)为0。如果y(i) >= table(end)或isnan (y(i))，则idx(i) = N。

如果表是递减的，那么测试是相反的。对于非严格单调表，总是跳过空间隔。如果table不是单调的，或者table包含NaN，则结果是未定义的。

查找的复杂度为O(M\*log(N))，其中M为y的大小。在y也排序的特殊情况下，复杂度为O(min (M\*log(N)， M+N))。

Table和y也可以是字符串的单元格数组(或者y可以是单个字符串)。在本例中，使用字典比较执行字符串查找。

如果指定了opts，则它必须是一个字符串，其中包含表示附加选项的字母。

m

匹配。Table (idx(i)) == y(i)，如果y(i)出现在Table中;否则，idx(i)为0。

B

布尔。Idx (i)是一个逻辑1或0，表示y(i)是否包含在表中。

l

离开了。对于数字查找，最左边的子区间应扩展到负无穷(即，所有索引至少为1)。

r

正确的。对于数值查找，最右子区间应扩展到无穷大(即，所有索引最多为N-1)。

注意:如果表没有排序，查找结果将是不可预测的。

如果您希望检查变量是否存在，而不是其元素可能具有的属性，请参阅变量状态。

**16.2重新排列矩阵**

: B = fliplr (A)

从左到右翻转数组。

返回a的副本，其中列的顺序颠倒。换句话说，A沿着垂直轴从左到右翻转。例如:

fliplr ([1, 2; 3, 4])

⇒ 2 1

4 3

参见:flipud, flip, rot90, rotdim。

: B = flipud (A)

将数组上下颠倒。

返回a的副本，但行顺序相反。换句话说，A沿着水平轴上下翻转。例如:

flipud ([1, 2; 3, 4])

⇒ 3 4

1 2

参见:fliplr, flip, rot90, rotdim。

: B = flip (A)

: B = flip (A, dim)

返回数组a在维度dim上翻转的副本。

如果未指定dim，则默认为第一个非单例维度。

例子:

## row vector

flip ([1 2 3 4])

⇒ 4 3 2 1

## column vector

flip ([1; 2; 3; 4])

⇒ 4

3

2

1

## 2-D matrix along dimension 1

flip ([1 2; 3 4])

⇒ 3 4

1 2

## 2-D matrix along dimension 2

flip ([1 2; 3 4], 2)

⇒ 2 1

4 3

参见:fliplr, flipud, rot90, rotdim, permute, transpose。

: B = rot90 (A)

: B = rot90 (A, k)

以90度的增量旋转数组。

返回a的副本，其中元素以90度的增量逆时针旋转。

第二个参数是可选的，指定要应用多少次90度旋转(默认值是1)。k的负值使矩阵顺时针方向旋转。例如,

rot90 ([1, 2; 3, 4], -1)

⇒ 3 1

4 2

将给定矩阵顺时针旋转90度。以下都是等价语句:

rot90 ([1, 2; 3, 4], -1)

rot90 ([1, 2; 3, 4], 3)

rot90 ([1, 2; 3, 4], 7)

旋转总是在前两个维度的平面上进行，即行和列。要在任何其他平面上执行旋转，请使用rotdim。

参见:rotdim, fliplr, flipud, flip。

: B = rotdim (A)

: B = rotdim (A, n)

: B = rotdim (A, n, plane)

返回a的副本，其中元素以90度的增量逆时针旋转。

第二个参数n是可选的，指定要应用多少次90度旋转(默认值是1)。n的负值使矩阵沿顺时针方向旋转。

第三个参数也是可选的，它定义了旋转的平面。如果存在，则平面是包含矩阵的两个不同有效维数的两元素向量。当平面未给定时，使用前两个非单维。

例如,

rotdim ([1, 2; 3, 4], -1, [1, 2])

⇒ 3 1

4 2

将给定矩阵顺时针旋转90度。以下都是等价语句:

rotdim ([1, 2; 3, 4], -1, [1, 2])

rotdim ([1, 2; 3, 4], 3, [1, 2])

rotdim ([1, 2; 3, 4], 7, [1, 2])

参见:rot90, fliplr, flipud, flip。

: A = cat (dim, array1, array2, …, arrayN)

返回N-D数组对象，array1, array2，…，arrayN沿维度dim的连接。

A = ones (2, 2);

B = zeros (2, 2);

cat (2, A, B)

⇒ 1 1 0 0

1 1 0 0

或者，我们可以按照以下方式沿着第二维将A和B连接起来:

[A, B]

dim可以大于N-D数组对象的维度，因此结果将具有dim维度，如下例所示:

cat (4, ones (2, 2), zeros (2, 2))

⇒ ans(:,:,1,1) =

1 1

1 1

ans(:,:,1,2) =

0 0

0 0

参见:横猫，竖猫。

: A = horzcat (array1, array2, …, arrayN)

返回N-D数组对象，array1, array2，…，arrayN沿维度2的水平连接。

也可以使用创建新矩阵的语法水平连接数组。例如:

A = [ array1, array2, … ]

这种语法的效率略高，因为Octave解析器可以将数组连接起来，而不需要函数调用的开销。

参见:cat, vertcat。

: A = vertcat (array1, array2, …, arrayN)

返回N-D数组对象，array1, array2，…，arrayN沿维度1垂直拼接。

也可以使用创建新矩阵的语法垂直连接数组。例如:

A = [ array1; array2; … ]

这种语法的效率略高，因为Octave解析器可以将数组连接起来，而不需要函数调用的开销。

参见:cat, horzcat。

: B = permute (A, perm)

返回N-D数组对象A的广义转置。

排列向量perm必须包含元素1:ndims (A)(以任何顺序，但每个元素只能出现一次)。A的第N维被映射到PERM(N)维。例如:

x = zeros ([2, 3, 5, 7]);

size (x)

⇒ 2 3 5 7

size (permute (x, [2, 1, 3, 4]))

⇒ 3 2 5 7

size (permute (x, [1, 3, 4, 2]))

⇒ 2 5 7 3

## The identity permutation

size (permute (x, [1, 2, 3, 4]))

⇒ 2 3 5 7

参见:ipermute。

: A = ipermute (B, iperm)

置换函数的逆。

表达式

ipermute (permute (A, perm), perm)

返回原始数组A。

参见:permute。

: B = reshape (A, m, n, …)

: B = reshape (A, [m n …])

: B = reshape (A, …, [], …)

: B = reshape (A, size)

返回具有指定维数(m, n，…)的矩阵，其元素取自矩阵a。

矩阵的元素以列为主的顺序访问(类似于Fortran数组的存储方式)。

下面的代码演示了将1x4行向量重塑为2x2方阵。

reshape ([1, 2, 3, 4], 2, 2)

⇒ 1 3

2 4

注意，原矩阵(prod (size (A)))中的元素总数必须与新矩阵(prod ([m n…])中的元素总数匹配。

返回矩阵的单个维度可以不指定，并且Octave将自动确定其大小。空矩阵([])用于标记未指定的维度。

参见:调整大小，调整，后封，调整，挤压。

: B = resize (A, m)

: B = resize (A, m, n, …)

: B = resize (A, [m n …])

根据需要调整A的大小，切断元素。

在结果中，如果指标在A的界内，则具有某些指标的元素等于A的对应元素;否则，该元素被设置为零。

换句话说，这个声明

B = resize (A, dv)

相当于以下代码:

B = zeros (dv, class (A));

sz = min (dv, size (A));

for i = 1:length (sz)

idx{i} = 1:sz(i);

endfor

B(idx{:}) = A(idx{:});

但执行效率更高。

如果只提供了m，并且它是一个标量，则结果的维数是m × m。如果m, n，…都是标量，那么结果的维数是m × n × -....如果给定一个向量作为输入，则结果的维数由该向量的元素给出。

可以将对象的大小调整到比它现有的维度更多的维度;在这种情况下，缺失的维度被假定为1。无法将对象的大小调整为更少的维度。

参见:重塑，postpad, prepad, cat。

: y = circshift (x, n)

: y = circshift (x, n, dim)

循环移动数组x的值。

n必须是一个不超过x的维数的整数向量。n的值可以是正的也可以是负的，这决定了x的值移动的方向。如果n中的一个元素为零，则x的相应维数不会移位。如果n是标量且没有指定dim，则移位应用于第一个非奇异维度。

如果给定标量dim，则沿着指定的维度进行操作。在这种情况下n也必须是一个标量。

例子:

x = [1, 2, 3;

4, 5, 6;

7, 8, 9];

## positive shift on rows (1st non-singular dim)

circshift (x, 1)

⇒

7 8 9

1 2 3

4 5 6

## negative shift on rows (1st non-singular dim)

circshift (x, -2)

⇒

7 8 9

1 2 3

4 5 6

## no shift of rows, shift columns by 1 (2nd dimension)

circshift (x, [0,1])

⇒

3 1 2

6 4 5

9 7 8

## shift columns (2nd dimension)

circshift (x, 1, 2)

⇒

3 1 2

6 4 5

9 7 8

参见:permute, ipermute, shiftdim。

: y = shiftdim (x, n)

: [y, ns] = shiftdim (x)

将x的维数移动n，其中n必须是一个整数标量。

当n为正数时，x的维数向左移动，前导维数循环到最后。如果n为负，则x的维度向右移动，并添加n个前导单维。

使用单个参数调用shiftdim，它删除前面的单例维度，返回在第二个输出参数ns中删除的维度数。

例如:

x = ones (1, 2, 3);

size (shiftdim (x, -1))

⇒ 1 1 2 3

size (shiftdim (x, 1))

⇒ 2 3

[b, ns] = shiftdim (x)

⇒ b =

1 1 1

1 1 1

⇒ ns = 1

参见:重塑，置换，置换，循环移位，挤压。

: [s, i] = sort (x)

: [s, i] = sort (x, dim)

: [s, i] = sort (x, mode)

: [s, i] = sort (x, dim, mode)

返回x的副本，其中元素按递增顺序排列。

对于矩阵，sort对列内的元素排序

例如:

sort ([1, 2; 2, 3; 3, 1])

⇒ 1 1

2 2

3 3

如果给出了可选参数dim，则矩阵按照dim定义的维度排序。可选参数模式定义了值排序的顺序。mode的有效值为“ascend”或“descent”。

排序函数也可用于生成包含已排序矩阵中元素的原始行索引的矩阵。例如:

[s, i] = sort ([1, 2; 2, 3; 3, 1])

⇒ s = 1 1

2 2

3 3

⇒ i = 1 3

2 1

3 2

对于相等的元素，索引是这样的，相等的元素按照它们在原始列表中出现的顺序列出。

复数项的排序首先是通过大小(abs (z))和相位角(angle (z))完成的。例如:

sort ([1+i; 1; 1-i])

⇒ 1 + 0i

1 - 1i

1 + 1i

NaN值被视为大于任何其他值，并被排序到列表的末尾。

sort函数也可用于对字符串和字符串的单元格数组进行排序，在这种情况下，使用字符串的ASCII字典顺序(大写' A '先于小写' A ')。

sort中使用的算法针对部分有序列表的排序进行了优化。

参见:sortrows, issorted。

: [s, i] = sortrows (A)

: [s, i] = sortrows (A, c)

根据c中指定的列的顺序对矩阵A的行进行排序。

默认情况下(c省略，或者c中未指定的特定列)使用升序排序顺序。但是，如果c的元素是负的，则相应的列按降序排序。如果A的元素是字符串，则使用字典排序。

示例:按列2按降序排序，然后按列3按升序排序

x = [ 7, 1, 4;

8, 3, 5;

9, 3, 6 ];

sortrows (x, [-2, 3])

⇒ 8 3 5

9 3 6

7 1 4

参见:排序。

: tf = issorted (A)

: tf = issorted (A, mode)

: tf = issorted (A, "rows", mode)

如果向量A按照mode排序，则返回true, mode可以是“上升”、“下降”或“两者皆有”。

缺省情况下，mode为“ascend”。nan的处理方式与sort相同。

如果提供了可选参数"rows"，检查函数sortrows输出的矩阵是否按行排序(不带选项)。

这个函数不支持稀疏矩阵。

又见:排序，忧伤。

: nel = nth\_element (x, n)

: nel = nth\_element (x, n, dim)

使用sort定义的排序，选择向量的第n个最小元素。

结果相当于sort(x)(n)。

N也可以是一个连续范围，可以是升序的l:u，也可以是降序的u:-1:l，在这种情况下，返回一个元素范围。

如果x是数组，则nth\_element沿着dim定义的维度操作，如果没有给出dim，则沿着第一个非单元素维度操作。

编程注意:nth\_element封装了c++标准库算法nth\_element和partial\_sort。平均而言，操作的复杂度为O(M\*log(K))，其中M = size (x, dim)， K = length (n)。此函数适用于比率K/M较小的情况;否则，使用sort可能会更好。

参见:sort, min, max。

: A\_LO = tril (A)

: A\_LO = tril (A, k)

: A\_LO = tril (A, k, pack)

返回通过提取矩阵a的下三角部分并将所有其他元素设置为零而形成的新矩阵。

可选的第二个参数指定主对角线上方或下方有多少条对角线也应设置为零。k的默认值是零，它包括主对角线作为结果的一部分。如果k的值是一个非零整数，则元素的选择从正k时主对角线上方的k个对角线偏移处开始，或在负k时主对角线下方偏移k。k的绝对值不得大于次对角线或超对角线的数量。

例1:排除主对角线

tril (ones (3), -1)

⇒ 0 0 0

1 0 0

1 1 0

示例2:包含第一个超对角线

tril (ones (3), 1)

⇒ 1 1 0

1 1 1

1 1 1

如果给出了可选的第三个参数“pack”，则提取的元素不会插入到矩阵中，而是按列堆叠，并作为列向量返回。

参见:triu, istril, diag。

: A\_UP = triu (A)

: A\_UP = triu (A, k)

: A\_UP = triu (A, k, pack)

返回通过提取矩阵a的上三角部分并将所有其他元素设置为零而形成的新矩阵。

可选的第二个参数指定主对角线上方或下方有多少条对角线也应设置为零。k的默认值是零，它包括主对角线作为结果的一部分。如果k的值是一个非零整数，则元素的选择从正k时主对角线上方的k个对角线偏移处开始，或在负k时主对角线下方偏移k。k的绝对值不得大于次对角线或超对角线的数量。

例1:排除主对角线

triu (ones (3), 1)

⇒ 0 1 1

0 0 1

0 0 0

示例2:包含第一个子对角线

triu (ones (3), -1)

⇒ 1 1 1

1 1 1

0 1 1

如果给出了可选的第三个参数“pack”，则提取的元素不会插入到矩阵中，而是按列堆叠，并作为列向量返回。

参见:tril, istriu, diag。

: v = vec (x)

: v = vec (x, dim)

返回通过将矩阵x的列一个叠在另一个上而获得的向量。

如果没有dim，这相当于x(:)。

如果提供dim，则v的维度被设置为dim，所有元素都沿着最后一个维度。这相当于shiftdim (x(:)， 1-dim)。

参见:调整，调整大小，调整。

: v = vech (x)

返回通过消除方阵x的所有超对角线元素并将结果一列堆叠在另一列上而获得的向量。

这在矩阵演算中有应用，其中底层矩阵是对称的，将值保持在主对角线以上是毫无意义的。

参见:vec。

: B = prepad (A, l)

: B = prepad (A, l, c)

: B = prepad (A, l, c, dim)

将标量值c添加到向量A之前，直到它的长度为l。如果没有给出c，则使用0的值。

如果长度(A) > l，则从A开始的元素被移除，直到获得长度为l的向量。

如果A是矩阵，则从每行添加或删除元素。

如果给出了可选参数dim，则沿着这个维度操作。

如果dim大于A的维度，则结果将具有dim维度。

参见:postpad, cat, resize。

: B = postpad (A, l)

: B = postpad (A, l, c)

: B = postpad (A, l, c, dim)

将标量值c附加到向量A，直到它的长度为l。如果没有给出c，则使用0的值。

如果length (A) > l，则移除A末尾的元素，直到获得长度为l的向量。

如果A是矩阵，则从每行添加或删除元素。

如果给出了可选参数dim，则沿着这个维度操作。

如果dim大于A的维度，则结果将具有dim维度。

参见:预铺，调整，调整大小。

: M = diag (v)

: M = diag (v, k)

: M = diag (v, m, n)

: v = diag (M)

: v = diag (M, k)

返回向量v在对角线k上的对角矩阵。

第二个参数是可选的。如果它是正数，则向量位于第k条超对角线上。如果它是负的，它被放在第k次对角线上。k的默认值为0，向量位于主对角线上。例如:

diag ([1, 2, 3], 1)

⇒ 0 1 0 0

0 0 2 0

0 0 0 3

0 0 0 0

3输入形式返回一个对角矩阵，向量v在主对角线上，结果矩阵的大小为m行x n列。

给定一个矩阵参数，而不是一个向量，diag提取矩阵的第k个对角线。

: M = blkdiag (A, B, C, …)

从a, B, C，…建立一个块对角矩阵

所有参数必须是数字，并且要么是二维矩阵，要么是标量。如果任何参数的类型是稀疏的，输出也将是稀疏的。

参见:斜，横，垂直，稀疏。

**16.2重新排列矩阵**

: B = fliplr (A)

从左到右翻转数组。

返回a的副本，其中列的顺序颠倒。换句话说，A沿着垂直轴从左到右翻转。例如:

fliplr ([1, 2; 3, 4])

⇒ 2 1

4 3

参见:flipud, flip, rot90, rotdim。

: B = flipud (A)

将数组上下颠倒。

返回a的副本，但行顺序相反。换句话说，A沿着水平轴上下翻转。例如:

flipud ([1, 2; 3, 4])

⇒ 3 4

1 2

参见:fliplr, flip, rot90, rotdim。

: B = flip (A)

: B = flip (A, dim)

返回数组a在维度dim上翻转的副本。

如果未指定dim，则默认为第一个非单例维度。

例子:

## row vector

flip ([1 2 3 4])

⇒ 4 3 2 1

## column vector

flip ([1; 2; 3; 4])

⇒ 4

3

2

1

## 2-D matrix along dimension 1

flip ([1 2; 3 4])

⇒ 3 4

1 2

## 2-D matrix along dimension 2

flip ([1 2; 3 4], 2)

⇒ 2 1

4 3

参见:fliplr, flipud, rot90, rotdim, permute, transpose。

: B = rot90 (A)

: B = rot90 (A, k)

以90度的增量旋转数组。

返回a的副本，其中元素以90度的增量逆时针旋转。

第二个参数是可选的，指定要应用多少次90度旋转(默认值是1)。k的负值使矩阵顺时针方向旋转。例如,

rot90 ([1, 2; 3, 4], -1)

⇒ 3 1

4 2

将给定矩阵顺时针旋转90度。以下都是等价语句:

rot90 ([1, 2; 3, 4], -1)

rot90 ([1, 2; 3, 4], 3)

rot90 ([1, 2; 3, 4], 7)

旋转总是在前两个维度的平面上进行，即行和列。要在任何其他平面上执行旋转，请使用rotdim。

参见:rotdim, fliplr, flipud, flip。

: B = rotdim (A)

: B = rotdim (A, n)

: B = rotdim (A, n, plane)

返回a的副本，其中元素以90度的增量逆时针旋转。

第二个参数n是可选的，指定要应用多少次90度旋转(默认值是1)。n的负值使矩阵沿顺时针方向旋转。

第三个参数也是可选的，它定义了旋转的平面。如果存在，则平面是包含矩阵的两个不同有效维数的两元素向量。当平面未给定时，使用前两个非单维。

例如,

rotdim ([1, 2; 3, 4], -1, [1, 2])

⇒ 3 1

4 2

rotates the given matrix clockwise by 90 degrees. The following are all equivalent statements:

rotdim ([1, 2; 3, 4], -1, [1, 2])

rotdim ([1, 2; 3, 4], 3, [1, 2])

rotdim ([1, 2; 3, 4], 7, [1, 2])

参见:rot90, fliplr, flipud, flip。

: A = cat (dim, array1, array2, …, arrayN)

返回N-D数组对象，array1, array2，…，arrayN沿维度dim的连接。

A = ones (2, 2);

B = zeros (2, 2);

cat (2, A, B)

⇒ 1 1 0 0

1 1 0 0

或者，我们可以按照以下方式沿着第二维将A和B连接起来:

[A, B]

dim可以大于N-D数组对象的维度，因此结果将具有dim维度，如下例所示:

cat (4, ones (2, 2), zeros (2, 2))

⇒ ans(:,:,1,1) =

1 1

1 1

ans(:,:,1,2) =

0 0

0 0

参见:横猫，竖猫。

: A = horzcat (array1, array2, …, arrayN)

返回N-D数组对象，array1, array2，…，arrayN沿维度2的水平连接。

也可以使用创建新矩阵的语法水平连接数组。例如:

A = [ array1, array2, … ]

这种语法的效率略高，因为Octave解析器可以将数组连接起来，而不需要函数调用的开销。

参见:cat, vertcat。

: A = vertcat (array1, array2, …, arrayN)

返回N-D数组对象，array1, array2，…，arrayN沿维度1垂直拼接。

也可以使用创建新矩阵的语法垂直连接数组。例如:

A = [ array1; array2; … ]

这种语法的效率略高，因为Octave解析器可以将数组连接起来，而不需要函数调用的开销。

参见:cat, horzcat。

: B = permute (A, perm)

返回N-D数组对象A的广义转置。

排列向量perm必须包含元素1:ndims (A)(以任何顺序，但每个元素只能出现一次)。A的第N维被映射到PERM(N)维。例如:

x = zeros ([2, 3, 5, 7]);

size (x)

⇒ 2 3 5 7

size (permute (x, [2, 1, 3, 4]))

⇒ 3 2 5 7

size (permute (x, [1, 3, 4, 2]))

⇒ 2 5 7 3

## The identity permutation

size (permute (x, [1, 2, 3, 4]))

⇒ 2 3 5 7

参见:ipermute。

: A = ipermute (B, iperm)

置换函数的逆。

表达式

ipermute (permute (A, perm), perm)

返回原数组A。

参见:permute。

: B = reshape (A, m, n, …)

: B = reshape (A, [m n …])

: B = reshape (A, …, [], …)

: B = reshape (A, size)

返回具有指定维数(m, n，…)的矩阵，其元素取自矩阵a。

矩阵的元素以列为主的顺序访问(类似于Fortran数组的存储方式)。

下面的代码演示了将1x4行向量重塑为2x2方阵。

reshape ([1, 2, 3, 4], 2, 2)

⇒ 1 3

2 4

注意，原矩阵(prod (size (A)))中的元素总数必须与新矩阵(prod ([m n…])中的元素总数匹配。

返回矩阵的单个维度可以不指定，并且Octave将自动确定其大小。空矩阵([])用于标记未指定的维度。

参见:调整大小，调整，后封，调整，挤压。

: B = resize (A, m)

: B = resize (A, m, n, …)

: B = resize (A, [m n …])

根据需要调整A的大小，切断元素。

在结果中，如果指标在A的界内，则具有某些指标的元素等于A的对应元素;否则，该元素被设置为零。

换句话说，这个声明

B = resize (A, dv)

相当于以下代码:

B = zeros (dv, class (A));

sz = min (dv, size (A));

for i = 1:length (sz)

idx{i} = 1:sz(i);

endfor

B(idx{:}) = A(idx{:});

但执行效率更高。

如果只提供了m，并且它是一个标量，则结果的维数是m × m。如果m, n，…都是标量，那么结果的维数是m × n × -....如果给定一个向量作为输入，则结果的维数由该向量的元素给出。

可以将对象的大小调整到比它现有的维度更多的维度;在这种情况下，缺失的维度被假定为1。无法将对象的大小调整为更少的维度。

参见:重塑，postpad, prepad, cat。

: y = circshift (x, n)

: y = circshift (x, n, dim)

循环移动数组x的值。

n必须是一个不超过x的维数的整数向量。n的值可以是正的也可以是负的，这决定了x的值移动的方向。如果n中的一个元素为零，则x的相应维数不会移位。如果n是标量且没有指定dim，则移位应用于第一个非奇异维度。

如果给定标量dim，则沿着指定的维度进行操作。在这种情况下n也必须是一个标量。

例子:

x = [1, 2, 3;

4, 5, 6;

7, 8, 9];

## positive shift on rows (1st non-singular dim)

circshift (x, 1)

⇒

7 8 9

1 2 3

4 5 6

## negative shift on rows (1st non-singular dim)

circshift (x, -2)

⇒

7 8 9

1 2 3

4 5 6

## no shift of rows, shift columns by 1 (2nd dimension)

circshift (x, [0,1])

⇒

3 1 2

6 4 5

9 7 8

## shift columns (2nd dimension)

circshift (x, 1, 2)

⇒

3 1 2

6 4 5

9 7 8

参见:permute, ipermute, shiftdim。

: y = shiftdim (x, n)

: [y, ns] = shiftdim (x)

将x的维数移动n，其中n必须是一个整数标量。

当n为正数时，x的维数向左移动，前导维数循环到最后。如果n为负，则x的维度向右移动，并添加n个前导单维。

使用单个参数调用shiftdim，它删除前面的单例维度，返回在第二个输出参数ns中删除的维度数。

例如:

x = ones (1, 2, 3);

size (shiftdim (x, -1))

⇒ 1 1 2 3

size (shiftdim (x, 1))

⇒ 2 3

[b, ns] = shiftdim (x)

⇒ b =

1 1 1

1 1 1

⇒ ns = 1

参见:重塑，置换，置换，循环移位，挤压。

: [s, i] = sort (x)

: [s, i] = sort (x, dim)

: [s, i] = sort (x, mode)

: [s, i] = sort (x, dim, mode)

返回x的副本，其中元素按递增顺序排列。

对于矩阵，sort对列内的元素排序

例如:

sort ([1, 2; 2, 3; 3, 1])

⇒ 1 1

2 2

3 3

如果给出了可选参数dim，则矩阵按照dim定义的维度排序。可选参数模式定义了值排序的顺序。mode的有效值为“ascend”或“descent”。

排序函数也可用于生成包含已排序矩阵中元素的原始行索引的矩阵。例如:

[s, i] = sort ([1, 2; 2, 3; 3, 1])

⇒ s = 1 1

2 2

3 3

⇒ i = 1 3

2 1

3 2

对于相等的元素，索引是这样的，相等的元素按照它们在原始列表中出现的顺序列出。

复数项的排序首先是通过大小(abs (z))和相位角(angle (z))完成的。例如:

sort ([1+i; 1; 1-i])

⇒ 1 + 0i

1 - 1i

1 + 1i

NaN值被视为大于任何其他值，并被排序到列表的末尾。

sort函数也可用于对字符串和字符串的单元格数组进行排序，在这种情况下，使用字符串的ASCII字典顺序(大写' A '先于小写' A ')。

sort中使用的算法针对部分有序列表的排序进行了优化。

参见:sortrows, issorted。

: [s, i] = sortrows (A)

: [s, i] = sortrows (A, c)

根据c中指定的列的顺序对矩阵A的行进行排序。

默认情况下(c省略，或者c中未指定的特定列)使用升序排序顺序。但是，如果c的元素是负的，则相应的列按降序排序。如果A的元素是字符串，则使用字典排序。

示例:按列2按降序排序，然后按列3按升序排序

x = [ 7, 1, 4;

8, 3, 5;

9, 3, 6 ];

sortrows (x, [-2, 3])

⇒ 8 3 5

9 3 6

7 1 4

参见:排序。

: tf = issorted (A)

: tf = issorted (A, mode)

: tf = issorted (A, "rows", mode)

如果向量A按照mode排序，则返回true, mode可以是“上升”、“下降”或“两者皆有”。

缺省情况下，mode为“ascend”。nan的处理方式与sort相同。

如果提供了可选参数"rows"，检查函数sortrows输出的矩阵是否按行排序(不带选项)。

这个函数不支持稀疏矩阵。

又见:排序，忧伤。

: nel = nth\_element (x, n)

: nel = nth\_element (x, n, dim)

使用sort定义的排序，选择向量的第n个最小元素。

结果相当于sort(x)(n)。

N也可以是一个连续范围，可以是升序的l:u，也可以是降序的u:-1:l，在这种情况下，返回一个元素范围。

如果x是数组，则nth\_element沿着dim定义的维度操作，如果没有给出dim，则沿着第一个非单元素维度操作。

编程注意:nth\_element封装了c++标准库算法nth\_element和partial\_sort。平均而言，操作的复杂度为O(M\*log(K))，其中M = size (x, dim)， K = length (n)。此函数适用于比率K/M较小的情况;否则，使用sort可能会更好。

参见:sort, min, max。

: A\_LO = tril (A)

: A\_LO = tril (A, k)

: A\_LO = tril (A, k, pack)

返回通过提取矩阵a的下三角部分并将所有其他元素设置为零而形成的新矩阵。

可选的第二个参数指定主对角线上方或下方有多少条对角线也应设置为零。k的默认值是零，它包括主对角线作为结果的一部分。如果k的值是一个非零整数，则元素的选择从正k时主对角线上方的k个对角线偏移处开始，或在负k时主对角线下方偏移k。k的绝对值不得大于次对角线或超对角线的数量。

例1:排除主对角线

tril (ones (3), -1)

⇒ 0 0 0

1 0 0

1 1 0

示例2:包含第一个超对角线

tril (ones (3), 1)

⇒ 1 1 0

1 1 1

1 1 1

如果给出了可选的第三个参数“pack”，则提取的元素不会插入到矩阵中，而是按列堆叠，并作为列向量返回。

参见:triu, istril, diag。

: A\_UP = triu (A)

: A\_UP = triu (A, k)

: A\_UP = triu (A, k, pack)

返回通过提取矩阵a的上三角部分并将所有其他元素设置为零而形成的新矩阵。

可选的第二个参数指定主对角线上方或下方有多少条对角线也应设置为零。k的默认值是零，它包括主对角线作为结果的一部分。如果k的值是一个非零整数，则元素的选择从正k时主对角线上方的k个对角线偏移处开始，或在负k时主对角线下方偏移k。k的绝对值不得大于次对角线或超对角线的数量。

例1:排除主对角线

triu (ones (3), 1)

⇒ 0 1 1

0 0 1

0 0 0

示例2:包含第一个子对角线

triu (ones (3), -1)

⇒ 1 1 1

1 1 1

0 1 1

如果给出了可选的第三个参数“pack”，则提取的元素不会插入到矩阵中，而是按列堆叠，并作为列向量返回。

参见:tril, istriu, diag。

: v = vec (x)

: v = vec (x, dim)

返回通过将矩阵x的列一个叠在另一个上而获得的向量。

如果没有dim，这相当于x(:)。

如果提供dim，则v的维度被设置为dim，所有元素都沿着最后一个维度。这相当于shiftdim (x(:)， 1-dim)。

参见:调整，调整大小，调整。

: v = vech (x)

返回通过消除方阵x的所有超对角线元素并将结果一列堆叠在另一列上而获得的向量。

这在矩阵演算中有应用，其中底层矩阵是对称的，将值保持在主对角线以上是毫无意义的。

参见:vec。

: B = prepad (A, l)

: B = prepad (A, l, c)

: B = prepad (A, l, c, dim)

将标量值c添加到向量A之前，直到它的长度为l。如果没有给出c，则使用0的值。

如果长度(A) > l，则从A开始的元素被移除，直到获得长度为l的向量。

如果A是矩阵，则从每行添加或删除元素。

如果给出了可选参数dim，则沿着这个维度操作。

如果dim大于A的维度，则结果将具有dim维度。

参见:postpad, cat, resize。

: B = postpad (A, l)

: B = postpad (A, l, c)

: B = postpad (A, l, c, dim)

将标量值c附加到向量A，直到它的长度为l。如果没有给出c，则使用0的值。

如果length (A) > l，则移除A末尾的元素，直到获得长度为l的向量。

如果A是矩阵，则从每行添加或删除元素。

如果给出了可选参数dim，则沿着这个维度操作。

如果dim大于A的维度，则结果将具有dim维度。

参见:预铺，调整，调整大小。

: M = diag (v)

: M = diag (v, k)

: M = diag (v, m, n)

: v = diag (M)

: v = diag (M, k)

返回向量v在对角线k上的对角矩阵。

第二个参数是可选的。如果它是正数，则向量位于第k条超对角线上。如果它是负的，它被放在第k次对角线上。k的默认值为0，向量位于主对角线上。例如:

diag ([1, 2, 3], 1)

⇒ 0 1 0 0

0 0 2 0

0 0 0 3

0 0 0 0

3输入形式返回一个对角矩阵，向量v在主对角线上，结果矩阵的大小为m行x n列。

给定一个矩阵参数，而不是一个向量，diag提取矩阵的第k个对角线。

: M = blkdiag (A, B, C, …)

从a, B, C，…建立一个块对角矩阵

所有参数必须是数字，并且要么是二维矩阵，要么是标量。如果任何参数的类型是稀疏的，输出也将是稀疏的。

参见:斜，横，垂直，稀疏。

**16.3特殊效用矩阵**

: I = eye (n)

: I = eye (m, n)

: I = eye ([m n])

: I = eye (…, class)

返回一个单位矩阵。

如果用单个标量参数n调用，则返回一个正方形的NxN单位矩阵。

如果提供两个标量参数(m, n)， eye将它们作为行数和列数。如果给定一个包含两个元素的向量，eye将分别使用元素的值作为行数和列数。例如:

eye (3)

⇒ 1 0 0

0 1 0

0 0 1

下面的表达式都产生相同的结果:

eye (2)

≡

eye (2, 2)

≡

eye (size ([1, 2; 3, 4]))

可选参数class允许eye返回指定类型的数组，如

val = zeros (n,m, "uint8")

不带参数调用eye等同于带参数1调用它。任何负维数都被视为零。这些奇怪的定义是为了与MATLAB兼容。

参见:speye, 1, 0。

: val = ones (n)

: val = ones (m, n)

: val = ones (m, n, k, …)

: val = ones ([m n …])

: val = ones (…, "like", var)

: val = ones (…, class)

返回元素均为1的矩阵或n维数组。

如果使用单个标量整数参数n调用，则返回方形NxN矩阵。

如果用两个或多个标量整型参数或整型值的向量调用，则返回具有给定维度的数组。

要创建一个值都相同的常量矩阵，请使用表达式，如

val\_matrix = val \* ones (m, n)

如果在"like"后面指定变量var，则输出的val将具有与var相同的数据类型、复杂度和稀疏性。

可选参数class指定返回数组的类，默认为double。例如:

val = ones (m,n, "uint8")

参见:零。

: val = zeros (n)

: val = zeros (m, n)

: val = zeros (m, n, k, …)

: val = zeros ([m n …])

: val = zeros (…, "like", var)

: val = zeros (…, class)

返回元素均为0的矩阵或n维数组。

如果使用单个标量整数参数调用，则返回方形NxN矩阵。

如果用两个或多个标量整型参数或整型值的向量调用，则返回具有给定维度的数组。

如果在"like"后面指定变量var，则输出的val将具有与var相同的数据类型、复杂度和稀疏性。

可选参数class指定返回数组的类，默认为double。例如:

val = zeros (m,n, "uint8")

参见:ones。

: B = repmat (A, m)

: B = repmat (A, m, n)

: B = repmat (A, m, n, p …)

: B = repmat (A, [m n])

: B = repmat (A, [m n p …])

重复矩阵或N-D数组。

形成一个大小为m × n的块矩阵，每个元素取矩阵a的一个副本。

如果n不指定，则形成一个m × m的块矩阵。对于沿着两个以上维度进行复制，在第二个参数中指定在向量的每个维度m, n, p，…上进行复制的次数。

参见:bsxfun, kron, repelems。

: y = repelems (x, r)

从x中构造一个重复元素的向量。

r是一个2xN的整数矩阵，指定重复哪些元素以及重复每个元素的频率。在第一行的条目r(1,j)中选择一个元素来重复。第二行中对应的条目r(2,j)指定了重复计数。如果x是一个矩阵，那么为了选择索引的目的，可以想象x的列彼此堆叠在一起。总是返回一个行向量。

概念上，计算结果如下:

y = [];

for i = 1:columns (r)

y = [y, x(r(1,i)\*ones(1, r(2,i)))];

endfor

参见:repmat, cat。

: xxx = repelem (x, R)

: xxx = repelem (x, R\_1, …, R\_n)

从x和重复指令R\_1， ....中构造一个重复元素的数组

x必须是标量、向量或n维数组。

重复指令R\_j必须是标量或向量。如果指令是一个标量，则维度j中x的每个分量重复R\_j次。如果指令是一个向量，那么它必须具有与x的对应维度j相同的元素数。在这种情况下，维度j的第k个分量重复R\_j(k)次。

如果x是标量或矢量，则只需用一个重复指令R调用repelem, repelem将返回与输入方向相同的矢量。

如果x是矩阵，则必须指定至少两个R\_js。

注意:将repelem与向量x和R\_j的向量一起使用相当于运行长度解码。

例子:

A = [1 2 3 4 5];

B = [2 1 0 1 2];

repelem (A, B)

⇒ 1 1 2 4 5 5

A = magic (3)

⇒ A =

8 1 6

3 5 7

4 9 2

B1 = [1 2 3];

B2 = 2;

repelem (A, B1, B2)

⇒ 8 8 1 1 6 6

3 3 5 5 7 7

3 3 5 5 7 7

4 4 9 9 2 2

4 4 9 9 2 2

4 4 9 9 2 2

可以指定比x的维数更多的R\_j。任何多余的R\_j必须是标量(因为x在这些维中的大小只有1)，并且x将相应地在这些维中被复制。

A = [1 2 3 4 5];

B1 = 2;

B2 = [2 1 3 0 2];

B3 = 3;

repelem (A, B1, B2, B3)

⇒ ans(:,:,1) =

1 1 2 3 3 3 5 5

1 1 2 3 3 3 5 5

ans(:,:,2) =

1 1 2 3 3 3 5 5

1 1 2 3 3 3 5 5

ans(:,:,3) =

1 1 2 3 3 3 5 5

1 1 2 3 3 3 5 5

R\_j必须按顺序指定。占位符1可用于不需要复制的维度。

repelem ([-1, 0; 0, 1], 1, 2, 1, 2)

⇒ ans(:,:,1,1) =

-1 -1 0 0

0 0 1 1

ans(:,:,1,2) =

-1 -1 0 0

0 0 1 1

如果给定的R\_j少于x中的维数，repelem将假设这些维的R\_j为1。

A = cat (3, [-1 0; 0 1], [-1 0; 0 1])

⇒ ans(:,:,1) =

-1 0

0 1

ans(:,:,2) =

-1 0

0 1

repelem (A,2,3)

⇒ ans(:,:,1) =

-1 -1 -1 0 0 0

-1 -1 -1 0 0 0

0 0 0 1 1 1

0 0 0 1 1 1

ans(:,:,2) =

-1 -1 -1 0 0 0

-1 -1 -1 0 0 0

0 0 0 1 1 1

0 0 0 1 1 1

repelem保留x类，并处理字符串、单元格数组、NA和NAN输入。如果任何R\_j为0，则输出将是一个空数组。

repelem ("Octave", 2, 3)

⇒ OOOccctttaaavvveee

OOOccctttaaavvveee

repelem ([1 2 3; 1 2 3], 2, 0)

⇒ [](4x0)

参见:cat, kron, repmat。

函数linspace和logspace使创建具有均匀或对数间隔元素的向量变得非常容易。看到的范围。

: y = linspace (start, end)

: y = linspace (start, end, n)

返回一个行向量，在开始和结束之间有n个线性间隔的元素。

如果元素的数量n大于1，则端点start和end总是包含在范围内。如果start大于end，则元素按降序存储。如果未指定点数n，则使用值100。

当开始和结束都是标量时，linspace函数返回一个行向量。如果一个或两个输入是向量，那么linspace将它们转换为列向量并返回一个矩阵，其中每一行都是start(row\_n)， end(row\_n)之间的独立序列。

编程注意:为了与MATLAB兼容，当请求单个值(n = 1)时，返回第二个参数(end)。如果n不是整数，则使用floor (n)来四舍五入元素数。如果n为零或负，则返回一个空的1x0矩阵。

参见:冒号，日志空间。

: y = logspace (a, b)

: y = logspace (a, b, n)

: y = logspace (a, pi)

: y = logspace (a, pi, n)

返回一个行向量，其中n个元素的对数间隔从10^a到10^b。

如果未指定元素个数n，则默认为50。

如果b =，则点在10^a和之间，而不是10^a和10^pi之间，这在数字信号处理中很有用。

编程注意事项:为了与MATLAB兼容，当请求单个值(n = 1)时，返回范围的右侧(10^b)。如果n不是整数，则使用floor (n)来四舍五入元素数。如果n为零或负，则返回一个空的1x0矩阵。

参见:linspace。

: x = rand (n)

: x = rand (m, n, …)

: x = rand ([m n …])

: x = rand (…, "single")

: x = rand (…, "double")

: v = rand ("state")

: rand ("state", v)

: rand ("state", "reset")

: v = rand ("seed")

: rand ("seed", v)

: rand ("seed", "reset")

返回一个随机元素均匀分布在区间(0,1)上的矩阵。

参数的处理方法与eye的参数相同。

您可以使用表单查询随机数生成器的状态

v = rand ("state")

返回长度为625的列向量v。稍后，您可以使用表单将随机数生成器恢复到状态v

rand ("state", v)

你也可以从v的长度≤625的任意向量初始化状态向量。这个新状态将是一个基于v值的哈希，而不是基于v本身。

默认情况下，生成器通过提供来自时钟时间、CPU时间、当前秒数、进程ID和(如果可用)最多1024位的熵来初始化，这些熵来自c++随机数源random\_device，这可能是不确定的(特定于实现)。请注意，这与MATLAB不同，MATLAB总是在启动时将状态初始化为相同的状态。要获得与MATLAB相当的行为，请在Octave的启动文件中使用确定性状态向量进行初始化(参见启动文件)。

为了计算伪随机序列，rand使用周期为2^{19937}-1的Mersenne Twister(参见M. Matsumoto和T. Nishimura, Mersenne Twister: a 623维等分布均匀伪随机数生成器，ACM Trans)。建模与计算机仿真，Vol. 8, No. 1, pp. 3-30, 1998年1月，http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html)。如果没有将多个返回值安全地散列在一起，请勿用于加密，否则在读取624个连续值后可以了解生成器状态。

旧版本的Octave使用不同的随机数生成器。默认情况下使用新生成器，因为它比旧生成器快得多，并且产生的随机数具有更长的周期时间。然而，在某些情况下，可能希望获得与旧生成器产生的相同的随机序列。要做到这一点，关键字"seed"被用来指定应该使用旧的生成器，如

rand ("seed", val)

它将生成器的种子设置为val。可以使用

s = rand ("seed")

然而，应该注意的是，查询种子不会导致rand使用旧的生成器，只有设置种子才会。要使rand再次使用新的生成器，应该使用关键字"state"来重置rand的状态。

可以使用“reset”关键字将生成器的状态或种子重置为一个新的随机值。

返回值的类型可以通过后面的“double”或“single”参数来控制。这些是唯一有效的类。

参见:randn, rande, rand, randp。

: R = randi (imax)

: R = randi (imax, n)

: R = randi (imax, m, n, …)

: R = randi ([imin imax], …)

: R = randi (…, "class")

返回1:imax范围内的随机整数。

其他参数决定返回矩阵的形状。如果没有指定参数，则返回单个随机整数。如果指定一个参数n，则返回一个方阵(n x n)。两个或多个参数将返回一个多维矩阵(m x n x…)。

整数范围可以有选择地用带有下界和上界的双元素矩阵来描述，在这种情况下，返回的整数将在区间[imin, imax]上。

可选参数类将返回所请求类型的矩阵。默认值为“double”。

下面的示例返回范围为1-10的150个整数。

ri = randi (10, 150, 1)

实现说明:randi内部依赖于rand, rand使用类“double”来表示数字。这将最大整数(imax)和范围(imax - imin)限制为flintmax函数返回的值。对于IEEE浮点数，该值为2^{53}- 1。

参见:rand, randn。

: x = randn (n)

: x = randn (m, n, …)

: x = randn ([m n …])

: x = randn (…, "single")

: x = randn (…, "double")

: v = randn ("state")

: randn ("state", v)

: randn ("state", "reset")

: v = randn ("seed")

: randn ("seed", v)

: randn ("seed", "reset")

返回一个具有正态分布随机元素的矩阵，平均值为零，方差为1。

参数的处理方式与rand的参数相同。

默认情况下，randn使用Marsaglia和Tsang的“Ziggurat技术”从均匀分布转换为正态分布。

返回值的类型可以通过后面的“double”或“single”参数来控制。这些是唯一有效的类。

参考文献:g.m assaglia, w.w.w Tsang，随机变量生成的Ziggurat方法，J.统计软件，第5卷，2000,https://www.jstatsoft.org/v05/i08/

参见:rand, rand, rand, randp。

: x = rande (n)

: x = rande (m, n, …)

: x = rande ([m n …])

: x = rande (…, "single")

: x = rande (…, "double")

: v = rande ("state")

: rande ("state", v)

: rande ("state", "reset")

: v = rande ("seed")

: rande ("seed", v)

: rande ("seed", "reset")

返回一个具有指数分布随机元素的矩阵。

参数的处理方式与rand的参数相同。

默认情况下，rande使用Marsaglia和Tsang的“Ziggurat技术”将均匀分布转换为指数分布。

返回值的类型可以通过后面的“double”或“single”参数来控制。这些是唯一有效的类。

参考文献:g.m assaglia, w.w.w Tsang，随机变量生成的Ziggurat方法，J.统计软件，第5卷，2000,https://www.jstatsoft.org/v05/i08/

参见:rand, randn, rand, randp。

: x = randp (l, n)

: x = randp (l, m, n, …)

: x = randp (l, [m n …])

: x = randp (…, "single")

: x = randp (…, "double")

: v = randp ("state")

: randp ("state", v)

: randp ("state", "reset")

: v = randp ("seed")

: randp ("seed", v)

: randp ("seed", "reset")

返回一个泊松分布随机元素的矩阵，其平均值参数由第一个参数l给出。

除了参数l之外，参数的处理与rand的参数相同。

根据l的取值范围以及l是标量还是矩阵，使用了五种不同的算法。

For scalar l ≤ 12, use direct method.

W.H. Press, et al., Numerical Recipes in C, Cambridge University Press, 1992.

For scalar l > 12, use rejection method.[1]

W.H. Press, et al., Numerical Recipes in C, Cambridge University Press, 1992.

For matrix l ≤ 10, use inversion method.[2]

E. Stadlober, et al., WinRand source code, available via FTP.

For matrix l > 10, use patchwork rejection method.

E. Stadlober, et al., WinRand source code, available via FTP, or H. Zechner, Efficient sampling from continuous and discrete unimodal distributions, Doctoral Dissertation, 156pp., Technical University Graz, Austria, 1994.

For l > 1e8, use normal approximation.

L. Montanet, et al., Review of Particle Properties, Physical Review D 50 p1284, 1994.

返回值的类型可以通过后面的“double”或“single”参数来控制。这些是唯一有效的类。

参见:rand, randn, rande, rand。

: x = randg (a, n)

: x = randg (a, m, n, …)

: x = randg (a, [m n …])

: x = randg (…, "single")

: x = randg (…, "double")

: v = randg ("state")

: randg ("state", v)

: randg ("state", "reset")

: v = randg ("seed")

: randg ("seed", v)

: randg ("seed", "reset")

返回一个包含gamma (a,1)个分布随机元素的矩阵。

除了参数a之外，参数的处理与rand的参数相同。

这可以用来生成许多发行版:

gamma (a, b) for a > -1, b > 0

r = b \* randg (a)

beta (a, b) for a > -1, b > -1

r1 = randg (a, 1)

r = r1 / (r1 + randg (b, 1))

Erlang (a, n)

r = a \* randg (n)

chisq (df) for df > 0

r = 2 \* randg (df / 2)

t (df) for 0 < df < inf (use randn if df is infinite)

r = randn () / sqrt (2 \* randg (df / 2) / df)

F (n1, n2) for 0 < n1, 0 < n2

## r1 equals 1 if n1 is infinite

r1 = 2 \* randg (n1 / 2) / n1

## r2 equals 1 if n2 is infinite

r2 = 2 \* randg (n2 / 2) / n2

r = r1 / r2

negative binomial (n, p) for n > 0, 0 < p <= 1

r = randp ((1 - p) / p \* randg (n))

non-central chisq (df, L), for df >= 0 and L > 0

(use chisq if L = 0)

r = randp (L / 2)

r(r > 0) = 2 \* randg (r(r > 0))

r(df > 0) += 2 \* randg (df(df > 0)/2)

Dirichlet (a1, … ak)

r = (randg (a1), …, randg (ak))

r = r / sum (r)

返回值的类型可以通过后面的“double”或“single”参数来控制。这些是唯一有效的类。

参见:rand, randn, rande, randp。

: rng (seed)

: rng (seed, "generator")

: rng ("shuffle")

: rng ("shuffle", "generator")

: rng ("default")

: s = rng ()

: rng (s)

: s = rng (…)

设置或查询rand和randn使用的随机数生成器的种子。

输入种子是一个标量数值，用于初始化随机数生成器的状态向量。

可选的字符串生成器指定要使用的随机数生成器的类型。取值为“twister”、“v5uniform”或“v5normal”。“twister”关键字描述如下。"v5uniform"和"v5normal"指的是以前使用不同随机数生成器的Octave的旧版本。

随机数生成器的状态或种子可以使用“shuffle”将其重置为新的随机值。

随机数生成器可以使用“default”关键字重置为默认值。默认值是使用种子为0的Mersenne Twister生成器。

可选的返回值s包含调用函数时随机数生成器的状态(即，在根据输入参数修改随机数生成器之前)。它被编码为具有三个字段的结构变量:“Type”、“Seed”和“State”。随机数生成器可以使用rng (s)恢复到状态s。当算法需要相同的伪随机数序列时，这很有用。

默认情况下，使用“twister”选项，使用周期为2^{19937}-1的Mersenne twister计算伪随机序列(参见M. Matsumoto和T. Nishimura, Mersenne twister: a 623维均匀分布的均匀伪随机数生成器，ACM Trans)。建模与计算机仿真，Vol. 8, No. 1, pp. 3-30, 1998年1月，http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html)。如果没有将多个返回值安全地散列在一起，请勿用于加密，否则在读取624个连续值后可以了解生成器状态。

参见:rand, randn。

发电机以新旧两种方式同时运行，不能将两者混合使用。使用“state”或“seed”初始化任何生成器都会导致其他生成器在将来调用时切换到相同的样式。

每个生成器的状态是独立的，对不同生成器的调用可以交错进行，而不会影响最终结果。例如,

rand ("state", [11, 22, 33]);

randn ("state", [44, 55, 66]);

u = rand (100, 1);

n = randn (100, 1);

和

rand ("state", [11, 22, 33]);

randn ("state", [44, 55, 66]);

u = zeros (100, 1);

n = zeros (100, 1);

for i = 1:100

u(i) = rand ();

n(i) = randn ();

end

产生相同的结果。当生成器以“seed”的旧风格初始化时，只有rand和randn是独立的，因为旧的rand, rand和randp生成器调用rand和randn。

生成器在启动时使用随机状态初始化，因此每次运行octave时随机数序列都不相同。7如果您确实需要精确地复制数字序列，您可以将状态或种子设置为特定值。

如果不带参数调用rand和randn，则返回随机序列的单个元素。

原始的rand和randn函数使用来自RANLIB的Fortran代码，RANLIB是一个用于随机数生成的Fortran例程库，由德克萨斯大学生物数学系的Barry W. Brown和James Lovato编译，M.D.安德森癌症中心，休斯顿，TX 77030。

: v = randperm (n)

: v = randperm (n, m)

返回一个包含随机排列1:n的行向量。

如果提供了m，则返回m个唯一的条目，从1:n中采样而不替换。

复杂度在内存上为O(n)，在时间上为O(m)，除非m < n/5，在这种情况下也使用O(m)内存。随机化是使用rand()执行的。所有排列都是等可能的。

参见:烫发。

**16.4著名矩阵**

下面的函数返回著名的矩阵形式。

: gallery (name)

: gallery (name, args)

为测试创建有趣的矩阵。

: c = gallery ("cauchy", x)

: c = gallery ("cauchy", x, y)

创建柯西矩阵。

: c = gallery ("chebspec", n)

: c = gallery ("chebspec", n, k)

创建切比雪夫谱微分矩阵。

: c = gallery ("chebvand", p)

: c = gallery ("chebvand", m, p)

为切比雪夫多项式创建一个范德蒙德式矩阵。

: a = gallery ("chow", n)

: a = gallery ("chow", n, alpha)

: a = gallery ("chow", n, alpha, delta)

创建一个Chow矩阵——一个奇异的Toeplitz下Hessenberg矩阵。

: c = gallery ("circul", v)

创建一个循环矩阵。

: a = gallery ("clement", n)

: a = gallery ("clement", n, k)

创建一个零对角元素的三对角矩阵。

: c = gallery ("compar", a)

: c = gallery ("compar", a, k)

创建一个比较矩阵。

: a = gallery ("condex", n)

: a = gallery ("condex", n, k)

: a = gallery ("condex", n, k, theta)

为条件估计器创建一个“反例”矩阵。

: a = gallery ("cycol", [m n])

: a = gallery ("cycol", n)

: a = gallery (…, k)

创建一个列循环重复的矩阵。

: [c, d, e] = gallery ("dorr", n)

: [c, d, e] = gallery ("dorr", n, theta)

: a = gallery ("dorr", …)

创建一个对角占优的，病态的，三对角矩阵。

: a = gallery ("dramadah", n)

: a = gallery ("dramadah", n, k)

创建一个(0,1)矩阵，它的逆矩阵有大的整数项。

: a = gallery ("fiedler", c)

创建一个对称的费德勒矩阵。

: a = gallery ("forsythe", n)

: a = gallery ("forsythe", n, alpha)

: a = gallery ("forsythe", n, alpha, lambda)

创建一个Forsythe矩阵(一个扰动乔丹块)。

: f = gallery ("frank", n)

: f = gallery ("frank", n, k)

创建一个Frank矩阵(病态特征值)。

: c = gallery ("gcdmat", n)

创建一个最大公约数矩阵。

C是一个n × n矩阵，其值对应于其坐标值的最大公因数，即C (i,j)对应于GCD (i,j)。

: a = gallery ("gearmat", n)

: a = gallery ("gearmat", n, i)

: a = gallery ("gearmat", n, i, j)

创建齿轮矩阵。

: g = gallery ("grcar", n)

: g = gallery ("grcar", n, k)

创建具有敏感特征值的Toeplitz矩阵。

: a = gallery ("hanowa", n)

: a = gallery ("hanowa", n, d)

创建一个矩阵，其特征值位于复平面的垂直线上。

: v = gallery ("house", x)

: [v, beta] = gallery ("house", x)

创建户主矩阵。

: a = gallery ("integerdata", imax, [M N …], j)

: a = gallery ("integerdata", imax, M, N, …, j)

: a = gallery ("integerdata", [imin, imax], [M N …], j)

: a = gallery ("integerdata", [imin, imax], M, N, …, j)

: a = gallery ("integerdata", …, "class")

创建一个范围为[1,imax]的随机整数矩阵。如果给定imin，则整数在[imin, imax]范围内。

第二个输入是描述输出大小的维度矩阵。维度也可以作为逗号分隔的参数输入。

输入j是一个在[0,2 ^32-1]范围内的整数索引。对于给定大小的输入和j索引，输出矩阵的值总是完全相同(可再现性)。

最后一个可选参数确定结果矩阵的类。类的可能值:uint8, uint16, uint32, int8, int16, int32, single, double。默认值是“double”。

: a = gallery ("invhess", x)

: a = gallery ("invhess", x, y)

创建上海森伯格矩阵的逆。

: a = gallery ("invol", n)

创建一个对合矩阵。

: a = gallery ("ipjfact", n)

: a = gallery ("ipjfact", n, k)

创建一个包含阶乘元素的Hankel矩阵。

: a = gallery ("jordbloc", n)

: a = gallery ("jordbloc", n, lambda)

创建一个Jordan block。

: u = gallery ("kahan", n)

: u = gallery ("kahan", n, theta)

: u = gallery ("kahan", n, theta, pert)

创建一个卡汉矩阵(上梯形)。

: a = gallery ("kms", n)

: a = gallery ("kms", n, rho)

创建一个Kac-Murdock-Szego Toeplitz矩阵。

: b = gallery ("krylov", a)

: b = gallery ("krylov", a, x)

: b = gallery ("krylov", a, x, j)

创建一个克雷洛夫矩阵。

: a = gallery ("lauchli", n)

: a = gallery ("lauchli", n, mu)

创建一个Lauchli矩阵(矩形)。

: a = gallery ("lehmer", n)

创建一个Lehmer矩阵(对称正定)。

: t = gallery ("lesp", n)

创建一个具有实的、敏感的特征值的三对角矩阵。

: a = gallery ("lotkin", n)

创建一个Lotkin矩阵。

: a = gallery ("minij", n)

创建一个对称正定矩阵MIN(i,j)。

: a = gallery ("moler", n)

: a = gallery ("moler", n, alpha)

创建一个摩尔矩阵(对称正定)。

: [a, t] = gallery ("neumann", n)

从离散诺伊曼问题(稀疏)中创建一个奇异矩阵。

: a = gallery ("normaldata", [M N …], j)

: a = gallery ("normaldata", M, N, …, j)

: a = gallery ("normaldata", …, "class")

用标准正态分布(mean = 0, std = 1)中的随机样本创建一个矩阵。

第一个输入是描述输出大小的维度矩阵。维度也可以作为逗号分隔的参数输入。

输入j是一个在[0,2 ^32-1]范围内的整数索引。对于给定大小的输入和j索引，输出矩阵的值总是完全相同(可再现性)。

最后一个可选参数确定结果矩阵的类。类的可能值:"single"， "double"。默认值是“double”。

: q = gallery ("orthog", n)

: q = gallery ("orthog", n, k)

创建正交和近正交矩阵。

: a = gallery ("parter", n)

创建一个伙伴矩阵(一个在π附近有奇异值的Toeplitz矩阵)。

: p = gallery ("pei", n)

: p = gallery ("pei", n, alpha)

创建一个Pei矩阵。

: a = gallery ("poisson", n)

从泊松方程(稀疏)创建一个块三对角矩阵。

: a = gallery ("prolate", n)

: a = gallery ("prolate", n, w)

创建一个扩展矩阵(对称的、病态的Toeplitz矩阵)。

: h = gallery ("randhess", x)

创建一个随机的，正交的上海森伯格矩阵。

: a = gallery ("rando", n)

: a = gallery ("rando", n, k)

创建一个元素为- 1,0或1的随机矩阵。

: a = gallery ("randsvd", n)

: a = gallery ("randsvd", n, kappa)

: a = gallery ("randsvd", n, kappa, mode)

: a = gallery ("randsvd", n, kappa, mode, kl)

: a = gallery ("randsvd", n, kappa, mode, kl, ku)

用预先分配的奇异值创建一个随机矩阵。

: a = gallery ("redheff", n)

创建一个与黎曼假设相关的0和1矩阵。

: a = gallery ("riemann", n)

创建一个与黎曼假设相关的矩阵。

: a = gallery ("ris", n)

创建一个对称的汉克尔矩阵。

: a = gallery ("smoke", n)

: a = gallery ("smoke", n, k)

创建一个复杂矩阵，带有“烟圈”伪光谱。

: t = gallery ("toeppd", n)

: t = gallery ("toeppd", n, m)

: t = gallery ("toeppd", n, m, w)

: t = gallery ("toeppd", n, m, w, theta)

创建一个对称的正定Toeplitz矩阵。

: p = gallery ("toeppen", n)

: p = gallery ("toeppen", n, a)

: p = gallery ("toeppen", n, a, b)

: p = gallery ("toeppen", n, a, b, c)

: p = gallery ("toeppen", n, a, b, c, d)

: p = gallery ("toeppen", n, a, b, c, d, e)

创建一个五对角线Toeplitz矩阵(稀疏)。

: a = gallery ("tridiag", x, y, z)

: a = gallery ("tridiag", n)

: a = gallery ("tridiag", n, c, d, e)

创建一个三对角矩阵(稀疏)。

: t = gallery ("triw", n)

: t = gallery ("triw", n, alpha)

: t = gallery ("triw", n, alpha, k)

创建Kahan, Golub和Wilkinson讨论过的上三角矩阵。

: a = gallery ("uniformdata", [M N …], j)

: a = gallery ("uniformdata", M, N, …, j)

: a = gallery ("uniformdata", …, "class")

用标准均匀分布(范围[0,1])中的随机样本创建一个矩阵。

第一个输入是描述输出大小的维度矩阵。维度也可以作为逗号分隔的参数输入。

输入j是一个在[0,2 ^32-1]范围内的整数索引。对于给定大小的输入和j索引，输出矩阵的值总是完全相同(可再现性)。

最后一个可选参数确定结果矩阵的类。类的可能值:"single"， "double"。默认值是“double”。

: a = gallery ("wathen", nx, ny)

: a = gallery ("wathen", nx, ny, k)

创建沃森矩阵。

: [a, b] = gallery ("wilk", n)

创建由威尔金森设计/讨论的各种特定矩阵。

: h = hadamard (n)

构造一个大小为n × n的Hadamard矩阵(Hn)。

大小n必须是2^k \* p的形式其中p是1,12,20或28中的一个。返回的矩阵归一化，即Hn(:，1) == 1, Hn(1，:) == 1。

Hadamard矩阵的一些性质是:

kron (Hm, Hn)是一个大小为m × n的Hadamard矩阵。

Hn \* Hn' = n \* eye (n)。

Hn的行是正交的。

对于所有abs (A(i, j)) <= 1的A, det (A) <= abs (det (Hn))。

将任意行或列乘以-1，矩阵仍然是阿达玛矩阵。

参见:company, hankel, toeplitz。

: h = hankel (c)

: h = hankel (c, r)

返回由第一列c和(可选)最后一行r构造的Hankel矩阵。

如果c的最后一个元素与r的第一个元素不相同，则使用c的最后一个元素。如果省略第二个参数，则假定它是一个与c大小相同的零向量。

由m向量c和n向量r组成的汉克尔矩阵具有以下元素

H(i,j) = c(i+j-1)， i+j-1 <= m;

H(i,j) = r(i+j-m)否则

参见:hadamard, toeplitz。

: h = hilb (n)

返回n阶的希尔伯特矩阵。

希尔伯特矩阵的第i,j元素定义为

H(i, j) = 1 / (i + j - 1)

希尔伯特矩阵接近于奇异，这使得它很难用数值程序进行反演。将随机矩阵5x5矩阵的条件数与5阶希尔伯特矩阵的条件数进行比较，可以看出问题有多么困难。

cond (rand (5))

⇒ 14.392

cond (hilb (5))

⇒ 4.7661e+05

参见:invhilb。

: hinv = invhilb (n)

返回n阶希尔伯特矩阵的逆。

可以使用

(i+j) /n+i-1\ /n+j-1\ /i+j-2\ 2

A(i,j) = -1 (i+j-1)( )( ) ( )

\ n-j / \ n-i / \ i-2 /

= p(i) p(j) / (i+j-1)

where

k /k+n-1\ /n\

p(k) = -1 ( ) ( )

\ k-1 / \k/

这个公式的有效性可以很容易地通过将两个公式中的二项式系数展开为阶乘来检验。它可以通过柯西矩阵理论更直接地推导出来。参见J. W. Demmel，应用数值线性代数，第92页。

将此与inv (hilb (n))的数值计算进行比较，后者受到希尔伯特矩阵的不良条件和计算机浮点运算的有限精度的影响。

参见:hilb。

: M = magic (n)

创建一个n × n的魔方。

幻方是整数1:n^2的排列，使得行和、列和和对角线和都等于相同的值。

注意:n必须是大于或等于3的标量。如果你提供的n小于3,magic将返回一个非幻方，或者简并幻方1和[]。

: P = pascal (n)

: P = pascal (n, t)

如果t = 0，返回n阶的Pascal矩阵。

t的默认值为0。

当t = 1时，返回Pascal矩阵的伪下三角Cholesky因子(某些列的符号可能是负的)。这个矩阵是它自己的逆矩阵，就是pascal (n, 1) ^ 2 == eye (n)

如果t = -1，则返回对角线上具有严格正值的真正Cholesky因子。

如果t = 2，返回pascal (n, 1)的转置和排列版本，它是单位矩阵的立方根。也就是pascal (n, 2) ^ 3 == eye (n)

参见:chol。

: R = rosser ()

返回Rosser矩阵。

这是一个用于评估特征值算法的困难测试用例。

参见:威尔金森，第8章。

: T = toeplitz (c)

: T = toeplitz (c, r)

返回由第一列c和可选的第一行r构造的Toeplitz矩阵。

如果省略第二个参数，则认为第一行与第一列相同。如果r的第一个元素与c的第一个元素不相同，则使用c的第一个元素。

Toeplitz或对角常数矩阵沿每条对角线具有相同的值。虽然它不一定是方形的，但它往往是方形的。一个MxN Toeplitz矩阵的形式是:

c(1) r(2) r(3) … r(n)

c(2) c(1) r(2) … r(n-1)

c(3) c(2) c(1) … r(n-2)

. . . . .

. . . . .

. . . . .

c(m) c(m-1) c(m-2) … c(m-n+1)

参见:汉高。

: V = vander (c)

: V = vander (c, n)

返回Vandermonde矩阵，其倒数第二列是c。

如果指定n，则确定列数;否则，取n等于c的长度。

Vandermonde矩阵的形式是:

c(1)^(n-1) … c(1)^2 c(1) 1

c(2)^(n-1) … c(2)^2 c(2) 1

. . . . .

. . . . .

. . . . .

c(n)^(n-1) … c(n)^2 c(n) 1

参见polyfit。

: W = wilkinson (n)

返回n阶的威尔金森矩阵。

威尔金森矩阵是对称的三对角线矩阵，其特征值对几乎相等，但不完全相等。它们在测试特征值求解器的行为和性能方面是有用的。

参见:rosser, 8。

**17个算术**

除非另有说明，本章中描述的所有函数都适用于实数和复数标量、向量或矩阵参数。当给定一个矩阵参数时，描述为映射函数的函数将给定的操作单独应用于每个元素。例如:

sin ([1, 2; 3, 4])

⇒ 0.84147 0.90930

0.14112 -0.75680

**17.1指数和对数**

: y = exp (x)

对x中的每个元素计算e^x。

要计算矩阵指数，请参见线性代数。

参见:log。

: y = expm1 (x)

在零附近精确地计算exp (x) - 1。

参见:exp。

: y = log (x)

计算自然对数ln (x)对于x的每个元素。

要计算矩阵对数，请参见线性代数。

参见:exp, log1p, log2, log10, logspace。

: y = reallog (x)

返回x的每个元素的实值自然对数。

如果任何元素导致一个复杂的返回值，reallog将中止并发出一个错误。

参见:log, realpow, realsqrt。

: y = log1p (x)

在零附近精确计算log (1 + x)。

参见:log, exp, expm1。

: y = log10 (x)

计算x的每个元素以10为底的对数。

参见:log, log2, logspace, exp。

: y = log2 (x)

: [f, e] = log2 (x)

计算x的每个元素以2为底的对数。

如果调用时只有一个输出，计算以2为底的对数，使2^y = x。

如果使用两个输出参数调用，则将x拆分为二进制尾数(f)和指数(e)，使x = f \* 2^e，其中1/2 <= abs (f) < 1并且e是整数。如果x = 0, f = e = 0。

参见:pow2, log, log10, exp。

: y = pow2 (x)

: y = pow2 (f, e)

使用一个输入参数，为x的每个元素计算y = 2 .^ x。

对于两个输入参数，返回y = f .\* (2 .^ e)，其中对于复杂输入只考虑两个输入的实部，而e只考虑实整数部分。这个调用形式对应于C/ c++标准函数ldexp()。

参见:log2, nextpow2, power。

: n = nextpow2 (x)

计算比输入值大2的最小次幂的指数。

对于输入数组x中的每个元素，返回第一个整数n，使2^n≥abs (x)。

参见:pow2, log2。

: z = realpow (x, y)

计算实值的逐元素幂运算符。

这相当于x. ^ y，除了realpow报告错误，如果任何返回值是复杂的。

参见:power, reallog, realsqrt。

: y = sqrt (x)

计算x的每个元素的平方根。

如果x为负，则返回一个复杂的结果。

要计算矩阵的平方根，请参见线性代数。

参见:realsqrt, nroot。

: y = realsqrt (x)

返回x中每个元素的实数平方根。

如果任何元素导致一个复杂的返回值，realsqrt将中止并发出一个错误。

参见:sqrt, realpow, reallog。

: y = cbrt (x)

计算x的每个元素的实值立方根。

与x^(1/3)不同，如果x为负，结果将是负的。

如果x的任何元素是复杂的，则cbrt将以错误终止。

参见:nroot。

: y = nthroot (x, n)

计算x的实数(非复数)n次方根。

X必须全是实数n必须是标量。如果n是偶数且x有负项，则nroot终止并发出错误。

例子:

nthroot (-1, 3)

⇒ -1

(-1) ^ (1 / 3)

⇒ 0.50000 - 0.86603i

参见:realsqrt, sqrt, cbrt。

**17.2复杂算术**

在以下函数的描述中，z为复数x + y，其中i定义为根号(-1)。

: z = abs (x)

计算x的大小。

大小定义为|z| = sqrt (x^2 + y^2)。

例如:

abs (3 + 4i)

⇒ 5

参见:arg。

: theta = arg (z)

: theta = angle (z)

计算参数，即z的角度。

定义为= atan2 (y, x)，单位是弧度。

例如:

arg (3 + 4i)

⇒ 0.92730

参见:腹肌。

: zc = conj (z)

返回z的共轭复数。

复共轭定义为conj (z) = x - y。

参见:real, image。

: zsort = cplxpair (z)

: zsort = cplxpair (z, tol)

: zsort = cplxpair (z, tol, dim)

将数字z按实部递增顺序排列成复数共轭对。

负虚复数在每对中放在首位。所有实数(abs (imag (z)) / abs (z) < tol)都放在复对之后。

Tol是一个在[0,1]范围内的加权因子，它决定了匹配的公差。默认值是100 \* eps，给定复杂对的结果公差是tol \* abs (z(i)))。

默认情况下，复对沿z的第一个非单维排序。如果指定了dim，则复对沿此维排序。

如果某些复数不能配对，则发出错误信号。如果所有的复数都不是精确共轭(到tol以内)，则发出错误信号。注意，对于实部相同但虚部不同的对，没有确定的顺序。

cplxpair (exp (2i\*pi\*[0:4]'/5)) == exp (2i\*pi\*[3; 2; 4; 1; 0]/5)

: y = imag (z)

返回z的虚部为实数。

参见:real, conj。

: x = real (z)

返回z的实部。

参见:image, conj。

**17.3三角**

Octave提供了以下三角函数，其中角度以弧度指定。要将角度转换为弧度，请乘以pi/180或使用deg2rad函数。例如，sin (30 \* pi/180)返回sin30度。作为替代，Octave提供了许多三角函数，它们直接作用于以度数指定的参数。这些函数以基底三角函数命名，并以“d”为后缀。例如，sin期望以弧度表示一个角，而sin期望以度表示一个角。

Octave使用C库的三角函数。期望这些功能是由ISO/IEC 9899标准定义的。本标准可在http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf上获得。第F.9.1节讨论三角函数。大多数函数的行为相对简单。然而，标准行为也有一些例外。许多异常涉及-0的行为。最复杂的例子是atan2。Octave精确地实现了标准中给出的行为。包括atan2(+- 0,0)返回+- pi。

需要注意的是，MATLAB使用了不同的定义，显然不区分-0。

: rad = deg2rad (deg)

将度转换为弧度。

输入度数必须是一个标量、矢量或双或单浮点值的n维数组。度可以是复数，在这种情况下实分量和虚分量分别转换。

输出弧度的大小和形状与度相同，使用转换常数pi/180将度转换为弧度。

例子:

deg2rad ([0, 90, 180, 270, 360])

⇒ 0.00000 1.57080 3.14159 4.71239 6.28319

参见:rad2deg。

: deg = rad2deg (rad)

将弧度转换为度数。

输入rad必须是一个标量、矢量或双或单浮点值的n维数组。Rad可能是复数，在这种情况下，实分量和虚分量分别转换。

输出度与rad的大小和形状相同，弧度使用转换常数180/pi转换为度。

例子:

rad2deg ([0, pi/2, pi, 3/2\*pi, 2\*pi])

⇒ 0 90 180 270 360

参见:deg2rad。

: y = sin (x)

以弧度为单位计算x的每个元素的正弦值。

参见:asin, sind, sinh。

: y = cos (x)

以弧度计算x的每个元素的余弦值。

参见:acos, cosd, cosh。

: y = tan (z)

以弧度为单位计算x的每个元素的正切。

参见:atan, tand, tanh。

: y = sec (x)

以弧度为单位计算x的每个元素的正割。

参见:asec, secd, sech。

: y = csc (x)

计算x的每个元素的余割，以弧度为单位。

参见:acsc, cscd, csch。

: y = cot (x)

以弧度为单位计算x的每个元素的余切。

参见:acot, cotd, coth。

: y = asin (x)

计算x的每个元素的反正弦弧度。

参见:sin, asind。

: y = acos (x)

计算x中每个元素的反余弦值(弧度)

参见:cos, acosd。

: y = atan (x)

计算x中每个元素的tan反比(弧度)

参见:tan, atand。

: y = asec (x)

计算x的每个元素的sec逆弧度。

参见:sec, asecd。

: y = acsc (x)

计算x中每个元素的倒数余割弧度。

参见:csc, acscd。

: y = acot (x)

计算x的每个元素的反余切弧度。

参见:cot, acotd。

: y = sinh (x)

计算x的每个元素的双曲正弦。

参见:asinh, cosh, tanh。

: y = cosh (x)

计算x的每个元素的双曲余弦。

参见:acosh, sinh, tanh。

: y = tanh (x)

计算x的每个元素的双曲正切。

参见：亚坦，辛，柯什。

: y = sech (x)

计算x的每个元素的双曲sec。

参见:asech。

: y = csch (x)

计算x的每个元素的双曲余割。

参见:acsch。

: y = coth (x)

计算x的每个元素的双曲余切。

参见:acoth。

: y = asinh (x)

计算x的每个元素的反双曲正弦。

参见:sinh。

: y = acosh (x)

计算x的每个元素的逆双曲余弦。

参见:cosh。

: y = atanh (x)

计算x的每个元素的逆双曲正切。

参见:tanh。

: y = asech (x)

计算x的每个元素的逆双曲sec。

参见:sech。

: y = acsch (x)

计算x的每个元素的逆双曲余割。

参见:csch。

: y = acoth (x)

计算x的每个元素的逆双曲余切。

参见:coth。

: angle = atan2 (y, x)

计算atan (y / x)，求y和x对应的元素。

Y和x必须在大小和方向上匹配。y和x元素的符号用于确定每个结果值的象限。

这个函数等价于arg (complex (x, y))。

参见:tan, tand, tanh, atanh。

Octave提供了以下三角函数，其中角度以度数指定。这些函数以适当的间隔产生真零，而不是使用弧度时产生的小舍入误差。例如:

cosd (90)

⇒ 0

cos (pi/2)

⇒ 6.1230e-17

: y = sind (x)

计算x的每个元素的正弦值，单位是度。

对于较大的x值和180度的倍数(x/180是整数)，该函数比sin更精确，其中sind返回0而不是eps量级的小值。

参见:asind, sin。

: y = cosd (x)

计算x的每个元素的余弦值，单位是度。

对于较大的x值和90度的倍数(x = 90 + 180\*n, n为整数)，该函数比cos更精确，其中cosd返回0而不是eps量级的小值。

参见:acosd, cos。

: y = tand (x)

以度为单位计算x的每个元素的正切。

对于x/180为整数的元素返回0，对于(x-90)/180为整数的元素返回Inf。

参见:atand, tan。

: y = secd (x)

计算x的每个元素的sec，单位是度。

参见:asecd, sec。

: y = cscd (x)

计算x的每个元素的余割，单位是度。

参见:acscd, csc。

: y = cotd (x)

计算x的每个元素的余切，单位是度。

参见:acotd, cot。

: y = asind (x)

计算x中每个元素的反正弦函数的度数。

参见:sin, asin。

: y = acosd (x)

计算x中每个元素的逆余弦值。

参见:cosd, acos。

: y = atand (x)

计算x的每个元素的tan反切度。

参见:tand, atan。

: d = atan2d (y, x)

计算atan (y / x)对y和x中相应元素的度数。

参见:tand, atan2。

: y = asecd (x)

计算x的每个元素的sec逆的度数。

参见:second, asec。

: y = acscd (x)

计算x的每个元素的倒数余割(以度为单位)

参见:cscd, acsc。

: y = acotd (x)

计算x的每个元素的逆余切(以度为单位)

参见:cotd, acot。

最后，有两个三角函数可以提高计算特殊参数的精度。

: y = sinpi (x)

为x的每个元素精确地计算sin (x \* pi)。

普通的sin函数使用IEEE浮点数，可能产生与正确值非常接近(在几个eps内)的结果，但并不精确。sinpi函数更精确，对于x的整数值返回0，对于半整数值返回+1/-1(例如，…，-3/2，-1/2,1/2,3/2，…)。

示例

x的整数值的sin和sinpi的比较

sin ([0, 1, 2, 3] \* pi)

⇒

0 1.2246e-16 -2.4493e-16 3.6739e-16

sinpi ([0, 1, 2, 3])

⇒

0 0 0 0

参见:cospi, sin。

: y = cospi (x)

为x的每个元素精确计算cos (x \* pi)。

普通的cos函数使用IEEE浮点数，可能产生与正确值非常接近(在几个eps内)的结果，但并不精确。cospi函数更精确，对于x的半整数值返回0(例如，…，-3/2，-1/2,1/2,3/2，…)，对于整数值返回+1/-1。

示例

x的半整数值的cos和cospi的比较

cos ([-3/2, -1/2, 1/2, 3/2] \* pi)

⇒

-1.8370e-16 6.1232e-17 6.1232e-17 -1.8370e-16

cospi ([-3/2, -1/2, 1/2, 3/2])

⇒

0 0 0 0

参见sinpi, cos。

**17.4总和和乘积**

: y = sum (x)

: y = sum (x, dim)

: y = sum (…, "native")

: y = sum (…, "double")

: y = sum (…, "extra")

沿维度dim的元素之和。

如果省略dim，则默认为第一个非单例维度。

可选的“type”输入决定了用于计算的变量的类别。默认情况下，对浮点输入(双精度或单精度)的操作以其原生数据类型执行，而对整数、逻辑和字符数据类型的操作则使用双精度执行。如果给出了参数“native”，则以与原始参数相同的类型执行操作。

例如:

sum ([true, true])

⇒ 2

sum ([true, true], "native")

⇒ true

如果给出"double"，则即使对于单精度输入，也会以双精度执行求和。

对于双精度输入，“extra”选项将使用比直接求和更精确的算法。对于单精度输入，“extra”与“double”相同。对于所有其他数据类型，“extra”没有影响。

参见:cumsum, sumsq, prod。

: y = prod (x)

: y = prod (x, dim)

: y = prod (…, "native")

: y = prod (…, "double")

元素沿量纲的乘积。

如果省略dim，则默认为第一个非单例维度。

可选的“type”输入决定了用于计算的变量的类别。如果给出了参数"native"，则以与原始参数相同的类型执行操作，而不是默认的双精度类型。

例如:

prod ([true, true])

⇒ 1

prod ([true, true], "native")

⇒ true

相反，如果给出“double”，即使是单精度输入，也会以双精度执行操作。

参见:cumprod, sum。

: y = cumsum (x)

: y = cumsum (x, dim)

: y = cumsum (…, "native")

: y = cumsum (…, "double")

沿维度dim的元素累积和。

如果省略dim，则默认为第一个非单例维度。例如:

cumsum ([1, 2; 3, 4; 5, 6])

⇒ 1 2

4 6

9 12

有关可选参数"native"和"double"的解释，请参见sum。

参见:sum, cumprod。

: y = cumprod (x)

: y = cumprod (x, dim)

元素沿维数累积积。

如果省略dim，则默认为第一个非单例维度。例如:

cumprod ([1, 2; 3, 4; 5, 6])

⇒ 1 2

3 8

15 48

参见:刺，cumsum。

: y = sumsq (x)

: y = sumsq (x, dim)

沿维度dim的元素平方和。

如果省略dim，则默认为第一个非单例维度。

这个函数在概念上等同于计算

sum (x .\* conj (x), dim)

但是它使用更少的内存，并且避免在x为实数时调用conj。

参见:sum, prod。

**17.5实用功能**

: y = ceil (x)

返回不小于x的最小整数。

这相当于向正无穷四舍五入。

如果x是复数，则返回ceil (real (x)) + ceil (imag (x)) \* I。

ceil ([-2.7, 2.7])

⇒ -2 3

参见:floor, round, fix。

: y = fix (x)

截断x的小数部分并返回整数部分。

这相当于向零四舍五入。如果x是复数，则返回fix (real (x)) + fix (imag (x)) \* I。

fix ([-2.7, 2.7])

⇒ -2 2

参见:天花板，地板，圆形。

: y = floor (x)

返回不大于x的最大整数。

这相当于向负无穷四舍五入。如果x是复数，则返回floor (real (x)) + floor (imag (x)) \* I。

floor ([-2.7, 2.7])

⇒ -3 2

参见:天花板，圆形，固定。

: y = round (x)

返回最接近x的整数。

如果x是复数，则返回round (real (x)) + round (imag (x)) \* i。如果有两个最接近的整数，则返回离零较远的整数。

round ([-2.7, 2.7])

⇒ -3 3

参见:天花板，地板，固定，圆形。

: y = roundb (x)

返回最接近x的整数。如果有两个最接近的整数，则返回偶数(银行四舍五入)。

如果x是复数，则返回roundb (real (x)) + roundb (imag (x)) \* I。

参见:圆的。

: m = max (x)

: m = max (x, [], dim)

: [m, im] = max (x)

: m = max (x, y)

找出数组x的最大值。

对于矢量参数，返回最大值。对于矩阵参数，返回每个列的最大值的行向量。对于多维数组，max沿着第一个非单维操作。

如果存在可选的第三个参数dim，则沿着该维度操作。在这种情况下，第二个参数被忽略，应该设置为空矩阵。

对于两个输入(x和y)，根据广播规则返回成对最大值。

因此,

max (max (x))

返回二维矩阵x的最大元素，并且

max (2:5, pi)

⇒ 3.1416 3.1416 4.0000 5.0000

将范围为2:5的每个元素与pi进行比较，并返回最大值的行向量。

对于复杂参数，元素的大小用于比较。如果大小相同，则结果按相位角在(-pi, pi]范围内排序。因此,

max ([-1 i 1 -i])

⇒ -1

因为所有元素的大小都是1，但是-1的相位角最大，值为。

如果使用一个输入参数和两个输出参数调用max，则max还返回最大值的第一个索引。因此,

[x, ix] = max ([1, 3, 5, 2, 5])

⇒ x = 5

ix = 3

参见:min, cummax, cummin。

: m = min (x)

: m = min (x, [], dim)

: [m, im] = min (x)

: m = min (x, y)

在数组x中找到最小值。

对于矢量参数，返回最小值。对于矩阵参数，返回每个列的最小值的行向量。对于多维数组，min沿着第一个非单维操作。

如果存在可选的第三个参数dim，则沿着该维度操作。在这种情况下，第二个参数被忽略，应该设置为空矩阵。

对于两个输入(x和y)，根据广播规则返回成对最小值。

因此,

min (min (x))

返回二维矩阵x的最小元素，并且

min (2:5, pi)

⇒ 2.0000 3.0000 3.1416 3.1416

将范围为2:5的每个元素与pi进行比较，并返回最小值的行向量。

对于复杂参数，元素的大小用于比较。如果大小相同，则结果按相位角在(-pi, pi]范围内排序。因此,

min ([-1 i 1 -i])

⇒ -i

因为所有元素的大小都是1，但是- 1的相位角最小，值是- /2。

如果使用一个输入参数和两个输出参数调用min，则min还返回最小值的第一个索引。因此,

[x, ix] = min ([1, 3, 0, 2, 0])

⇒ x = 0

ix = 3

参见:max, cummin, cummax。

: M = cummax (x)

: M = cummax (x, dim)

: [M, IM] = cummax (…)

返回沿维度dim的累积最大值。

如果未指定dim，则默认为按列操作。例如:

cummax ([1 3 2 6 4 5])

⇒ 1 3 3 6 6 6

如果使用两个输出参数调用，则还返回最大值的索引。

[w, iw] = cummax ([1 3 2 6 4 5])

⇒

M = 1 3 3 6 6 6

IM = 1 2 2 4 4 4

参见:小茴香，最大，最小。

: M = cummin (x)

: M = cummin (x, dim)

: [M, IM] = cummin (x)

返回沿维度dim的累积最小值。

如果未指定dim，则默认为按列操作。例如:

cummin ([5 4 6 2 3 1])

⇒ 5 4 4 2 2 1

如果用两个输出参数调用，也会返回最小值的索引。

[M, IM] = cummin ([5 4 6 2 3 1])

⇒

M = 5 4 4 2 2 1

IM = 1 2 2 4 4 6

参见:最大，最小，最大。

: h = hypot (x, y)

: h = hypot (x, y, z, …)

逐元素计算x和y平方和的平方根。

这相当于sqrt (x.^2 + y.^2)，但计算的方式避免了x或y的大值溢出。

也可以使用2个以上参数调用Hypot;在这种情况下，参数从左到右累积:

hypot (hypot (x, y), z)

hypot (hypot (hypot (x, y), z), w), etc.

: dx = gradient (m)

: [dx, dy, dz, …] = gradient (m)

: […] = gradient (m, s)

: […] = gradient (m, x, y, z, …)

: […] = gradient (f, x0)

: […] = gradient (f, x0, s)

: […] = gradient (f, x0, x, y, …)

计算抽样数据或函数的梯度。

如果m是向量，则计算m的一维梯度。如果m是矩阵，则计算每个维度的梯度。

[dx, dy] = gradient (m)如果m是矩阵，则计算x和y方向的一维梯度。附加的返回参数可用于多维矩阵。

两点之间的恒定间距可由s参数提供。如果s是一个标量，则假定它是所有维度的间距。否则，可以通过x，…参数提供空格的单独值。标量值指定等距间距。参数x，…的向量值指定该维度的坐标。长度必须匹配它们各自的尺寸m。

在边界点处应用线性外推。用数值梯度的第一次近似计算内部点

y'(i) = 1/(x(i+1)-x(i-1)) \* (y(i-1)-y(i+1)).

如果第一个参数f是一个函数句柄，则函数在x0点处的梯度使用中心差来近似。例如，gradient (@cos, 0)近似cos函数在点x0 = 0处的梯度。与采样数据一样，估计梯度的点之间的间距值可以通过s或dx, dy，…参数设置。默认情况下，使用1的间距。

参见:diff, del2。

: z = dot (x, y)

: z = dot (x, y, dim)

计算两个向量的点积。

如果x和y是矩阵，计算沿第一个非单维的点积。

如果给出了可选参数dim，则计算沿该维度的点积。

实现注意:这相当于sum (conj (X) .\* Y, dim)，但避免形成临时数组并且更快。当X和Y是列向量时，结果等于X' \* Y。虽然dot是为整数数组定义的，但由于整数对象的范围有限，输出可能与预期结果不同。

参见:cross, divergence。

: z = cross (x, y)

: z = cross (x, y, dim)

计算两个三维向量x和y的向量外积。

如果x和y是矩阵，则沿第一维用三个元素进行叉乘。

可选参数dim强制沿指定的尺寸计算叉积。

示例代码:

cross ([1, 1, 0], [0, 1, 1])

⇒

1 -1 1

参见点，旋度，散度。

: div = divergence (x, y, z, fx, fy, fz)

: div = divergence (fx, fy, fz)

: div = divergence (x, y, fx, fy)

: div = divergence (fx, fy)

分别计算由数组fx, fy和fz或fx, fy给出的向量场的散度。

d d d

div F(x,y,z) = -- F(x,y,z) + -- F(x,y,z) + -- F(x,y,z)

dx dy dz

向量场的坐标可以分别由参数x, y, z或x, y给出。

参见:curl, gradient, del2, dot。

: [cx, cy, cz, v] = curl (x, y, z, fx, fy, fz)

: [cz, v] = curl (x, y, fx, fy)

: […] = curl (fx, fy, fz)

: […] = curl (fx, fy)

: v = curl (…)

分别计算由数组fx, fy和fz或fx, fy给出的向量场的旋度。

/ d d d d d d \

curl F(x,y,z) = | -- Fz - -- Fy, -- Fx - -- Fz, -- Fy - -- Fx |

\ dy dz dz dx dx dy /

向量场的坐标可以分别由参数x, y, z或x, y给出。V计算二维输入角速度矢量在z轴方向上的标量分量。对于三维输入，在每个网格点沿该点的矢量场方向计算标量旋转。

参见:散度，梯度，del2，交叉。

: L = del2 (M)

: L = del2 (M, h)

: L = del2 (M, dx, dy, …)

计算离散拉普拉斯算子。

对于二维矩阵M，它被定义为

1 / d^2 d^2 \

L = --- \* | --- M(x,y) + --- M(x,y) |

4 \ dx^2 dy^2 /

对于n维数组，括号中的和被扩展为包含附加高维上的二阶导数。

评价点之间的间距可以用h来定义，h是一个标量，它定义了所有维度上的等距间距。或者，每个维度上的间距可以分别用dx、dy等来定义。标量间距参数定义等距间距，而矢量参数可用于指定变量间距。间距向量的长度必须匹配m的各自维度，默认的间距值为1。

少于3个数据点的维度将被跳过。边界点由内部点的线性外推计算得到。

例子:2\*x^3的二阶导数

f = @(x) 2\*x.^3;

dd = @(x) 12\*x;

x = 1:6;

L = 4\*del2 (f(x));

assert (L, dd (x));

参见:梯度，差异。

: f = factorial (n)

返回n的阶乘，其中n是一个实非负整数。

如果n是标量，它等价于prod (1:n)。对于向量或矩阵参数，返回数组中每个元素的阶乘。

对于非整数，请参阅广义阶乘函数。请注意，阶乘函数增长得非常快，即使使用双精度值，如果n > 171也会发生溢出。对于这种情况，考虑伽玛尔。

参见:prod, gamma, gammaln。

: pf = factor (q)

: [pf, n] = factor (q)

返回q的质因数分解。

质因数分解定义为prod (pf) == q，其中pf的每个元素都是素数。如果q == 1，则返回1。输出pf与输入具有相同的数字类。

使用两个输出参数，返回唯一质因数pf及其倍数。即prod (pf .^ n) == q。

实现注意:如果输入q是单或双，那么它不能超过相应的flintmax。对于较大的输入，如果小于2^64，则将其转换为uint64:

factor (uint64 (18446744073709011493))

⇒ 571111 761213 42431951

对于更大的输入，如果你已经安装并加载了Symbolic包，则使用sym:

factor (sym ('9444733049654361449941'))

⇒ (sym)

1 1

1099511627689 ⋅8589934669

参见:gcd, lcm, isprime, primes。

: g = gcd (a1, a2, …)

: [g, v1, …] = gcd (a1, a2, …)

计算a1 a2 ....的最大公约数

所有参数必须是相同的大小或标量。对于数组，将分别为每个元素计算最大公约数。所有元素必须是普通整数或高斯(复数)整数。请注意，对于高斯整数，gcd仅在相位因子(乘以1、-1、i或-i)范围内是唯一的，因此返回四个可能的最大公约数。

可选返回参数v1，…，包含整数向量，

g = v1 .\* a1 + v2 .\* a2 + …

代码示例

gcd ([15, 9], [20, 18])

⇒ 5 9

编程技巧:要查找单个数组中所有元素的GCD，使用num2cell来代替嵌套调用或循环:

x = [30 42 70 105]; # vector or array of inputs

gcd (num2cell (x) {:})

⇒ 1

参见:lcm, factor, isprime。

: l = lcm (x, y)

: l = lcm (x, y, …)

计算x和y的最小公倍数，或所有参数列表的最小公倍数。

所有输入必须具有相同的大小或标量。所有元素必须是实整数或高斯(复数)整数。对于复杂输入，结果在相位因子(乘以+1、+i、-1或-i)范围内是唯一的，并且任意返回其中一个。

示例代码:

lcm (5:8, 9:12)

⇒ 45 30 77 24

编程技巧:要找到单个数组中所有元素的LCM，使用num2cell来代替嵌套调用或循环:

x = 1:10; # vector or array of inputs

lcm (num2cell (x) {:})

⇒ 2520

参见:factor, gcd, isprime。

: r = rem (x, y)

返回除法x / y的余数。

余数使用表达式计算

x - y .\* fix (x ./ y)

如果参数的维度不一致，或者其中任何一个参数都很复杂，则打印错误消息。

编程注意事项:在使用浮点数(双、单)进行计算时，为了与MATLAB兼容，在计算之前，整数的几eps内的值将被四舍五入为该整数。任何大于flintmax(2^53为双精度)的浮点整数将无法正确计算。对于更大的整数值，在调用此函数之前将输入转换为uint64。

按照惯例,

rem (x, 0) = NaN if x is a floating point variable

rem (x, 0) = 0 if x is an integer variable

rem (x, y) returns a value with the signbit from x

有关相反的约定，请参阅mod函数。一般来说，在计算两个正数相除后的余数时，rem是最好的。对于负数，或者当值是周期性的，mod是一个更好的选择。

参见:mod。

: m = mod (x, y)

计算x和y的模。

概念上这是由

x - y .\* floor (x ./ y)

对于整数类型返回正确的模数。这个函数正确地处理负值。也就是说，mod(- 1,3)是2，而不是rem(- 1,3)返回的-1。

如果参数的维度不一致，或者其中一个参数是复杂的，则会导致错误。

编程注意事项:在使用浮点数(双、单)进行计算时，为了与MATLAB兼容，在计算之前，整数的几eps内的值将被四舍五入为该整数。任何大于flintmax(2^53为双精度)的浮点整数将无法正确计算。对于更大的整数值，在调用此函数之前将输入转换为uint64。

按照惯例,

mod (x, 0) = x

mod (x, y) returns a value with the signbit from y

有关相反的约定，请参阅rem函数。一般来说，当任何输入都是负数或值是周期性的时候，mod是比rem更好的选择。

参见:rem。

: p = primes (n)

返回n以内的所有质数。

输出的数据类(double、single、uint32等)与n的输入类相同。使用的算法是埃拉托色尼的Sieve。

注意:对于特定数量的n个素数，调用list\_primes (n)。或者，调用primes (n\*log (k\*n))(1:n)，其中k约为5或6。这是可行的，因为从一个素数到下一个素数的距离平均与该素数的对数成正比。在积分上，有n个小于n \* log (5\*n)的素数。

参见:list\_prime, isprime。

: p = list\_primes ()

: p = list\_primes (n)

列出前n个质数。

如果n未指定，则列出前25个素数。

参见:质数，isprime。

: y = sign (x)

计算sgn函数。

它被定义为

-1, x < 0;

sign (x) = 0, x = 0;

1, x > 0.

对于复杂参数，sign返回x ./ abs (x)。

注意，符号(-0.0)是0。虽然IEEE 754浮点允许0有符号，0.0和-0.0比较相等。如果必须测试零是否有符号，请使用signbit函数。

参见:signbit。

: y = signbit (x)

如果x的值设置了其符号位，则返回逻辑true，否则返回false。

此行为与其他逻辑函数一致。参见逻辑值。该行为不同于C语言函数，如果设置了符号位，则返回非零。

这与x < 0.0不同，因为IEEE 754浮点数允许对零进行符号处理。比较-0.0 < 0.0为false，但signbit(-0.0)将返回一个非零值。

参见:sign。

**17.6特殊功能**

: a = airy (z)

: a = airy (k, z)

: a = airy (k, z, scale)

: [a, ierr] = airy (…)

计算第一类和第二类艾里函数及其导数。

K Function Scale factor (if scale is true)

--- -------- ---------------------------------------

0 Ai (Z) exp ((2/3) \* Z \* sqrt (Z))

1 dAi(Z)/dZ exp ((2/3) \* Z \* sqrt (Z))

2 Bi (Z) exp (-abs (real ((2/3) \* Z \* sqrt (Z))))

3 dBi(Z)/dZ exp (-abs (real ((2/3) \* Z \* sqrt (Z))))

函数调用airy (z)等同于airy (0, z)。

可选的第三输入标度决定是否应用如上所述的标度。默认为false。

结果a和z一样大。

可选输出ierr包含以下状态信息，其大小与结果相同。

正常返回。

输入错误，返回NaN。

溢出，返回Inf。

由于参数减少而导致的意义丧失导致不到一半的机器精度。

因参数减少而失去意义，输出可能不准确。

错误-没有计算，不满足算法终止条件，返回NaN。

: J = besselj (alpha, x)

: J = besselj (alpha, x, opt)

: [J, ierr] = besselj (…)

计算第一类贝塞尔函数。

贝塞尔函数的阶必须是实数。求值x的点可能很复杂。

如果可选参数opt为1或true，则结果J乘以exp (-abs (imag (x)))。

如果alpha是一个标量，则结果与x大小相同。如果x是一个标量，则结果与alpha大小相同。如果alpha是一个行向量，x是一个列向量，那么结果就是一个行长度为x，列长度为alpha的矩阵。否则，alpha和x必须一致，结果将是相同的大小。

如果被请求，ierr包含以下状态信息，并且与结果大小相同。

正常返回。

输入错误，返回NaN。

溢出，返回Inf。

由于参数减少而导致的意义丧失导致不到一半的机器精度。

因参数减少而失去意义，输出可能不准确。

错误-没有计算，不满足算法终止条件，返回NaN。

参见:bessely, besseli, besselk, besselh。

: Y = bessely (alpha, x)

: Y = bessely (alpha, x, opt)

: [Y, ierr] = bessely (…)

计算第二类贝塞尔函数。

贝塞尔函数的阶必须是实数。求值x的点可能很复杂。

如果可选参数opt为1或true，则结果Y乘以exp (-abs (imag (x)))。

如果alpha是一个标量，则结果与x大小相同。如果x是一个标量，则结果与alpha大小相同。如果alpha是一个行向量，x是一个列向量，那么结果就是一个行长度为x，列长度为alpha的矩阵。否则，alpha和x必须一致，结果将是相同的大小。

如果被请求，ierr包含以下状态信息，并且与结果大小相同。

正常返回。

输入错误，返回NaN。

溢出，返回Inf。

由于参数减少而导致的意义丧失导致不到一半的机器精度。

通过减少参数完全失去意义，返回NaN。

错误-没有计算，不满足算法终止条件，返回NaN。

参见:besselj, besseli, besselk, besselh。

: I = besseli (alpha, x)

: I = besseli (alpha, x, opt)

: [I, ierr] = besseli (…)

计算第一类修正贝塞尔函数。

贝塞尔函数的阶必须是实数。求值x的点可能很复杂。

如果可选参数opt为1或true，则结果I乘以exp (-abs (real (x)))。

如果alpha是一个标量，则结果与x大小相同。如果x是一个标量，则结果与alpha大小相同。如果alpha是一个行向量，x是一个列向量，那么结果就是一个行长度为x，列长度为alpha的矩阵。否则，alpha和x必须一致，结果将是相同的大小。

如果被请求，ierr包含以下状态信息，并且与结果大小相同。

正常返回。

输入错误，返回NaN。

溢出，返回Inf。

由于参数减少而导致的意义丧失导致不到一半的机器精度。

通过减少参数完全失去意义，返回NaN。

错误-没有计算，不满足算法终止条件，返回NaN。

参见:besselk, besselj, bessely, besselh。

: K = besselk (alpha, x)

: K = besselk (alpha, x, opt)

: [K, ierr] = besselk (…)

计算第二类修正贝塞尔函数。

贝塞尔函数的阶必须是实数。求值x的点可能很复杂。

如果可选参数opt为1或true，则结果K乘以exp (x)。

如果alpha是一个标量，则结果与x大小相同。如果x是一个标量，则结果与alpha大小相同。如果alpha是一个行向量，x是一个列向量，那么结果就是一个行长度为x，列长度为alpha的矩阵。否则，alpha和x必须一致，结果将是相同的大小。

如果被请求，ierr包含以下状态信息，并且与结果大小相同。

正常返回。

输入错误，返回NaN。

溢出，返回Inf。

由于参数减少而导致的意义丧失导致不到一半的机器精度。

通过减少参数完全失去意义，返回NaN。

错误-没有计算，不满足算法终止条件，返回NaN。

参见:besseli, besselj, bessely, besselh。

: H = besselh (alpha, x)

: H = besselh (alpha, k, x)

: H = besselh (alpha, k, x, opt)

: [H, ierr] = besselh (…)

计算第三类贝塞尔函数(Hankel函数)。

贝塞尔函数的阶必须是实数。汉高函数的类型由k指定，可以是第一种(k = 1)，也可以是第二种(k = 2)。默认是第一种汉高函数。求值x的点可能很复杂。

如果可选参数opt为1或true，则结果乘以exp (-I\*x) (k = 1)或exp (I\*x) (k = 2)。

如果alpha是一个标量，则结果与x大小相同。如果x是一个标量，则结果与alpha大小相同。如果alpha是一个行向量，x是一个列向量，那么结果就是一个行长度为x，列长度为alpha的矩阵。否则，alpha和x必须一致，结果将是相同的大小。

如果被请求，ierr包含以下状态信息，并且与结果大小相同。

正常返回。

输入错误，返回NaN。

溢出，返回Inf。

由于参数减少而导致的意义丧失导致不到一半的机器精度。

通过减少参数完全失去意义，返回NaN。

错误-没有计算，不满足算法终止条件，返回NaN。

参见:besselj, bessely, besseli, besselk。

: y = beta (a, b)

计算实输入a和b的函数。

函数的定义是

beta (a, b) = gamma (a) \* gamma (b) / gamma (a + b).

Beta函数可以变得非常大，通常处理输出的对数比直接处理函数更有用。参考betaln，以一种有效的方式计算Beta函数的对数。

参见:betaln, betainc, betaincinv。

: I = betainc (x, a, b)

: I = betainc (x, a, b, tail)

计算不完全函数。

它被定义为

x

/

1 |

I\_x (a, b) = ---------- | t^(a-1) (1-t)^(b-1) dt

beta (a,b) |

/

0

实x在[0,1]范围内。输入a和b必须是实数且严格正的(> 0)。如果其中一个输入不是标量，则其他输入必须是标量或具有兼容的维度。

默认情况下，尾部为“较低”，计算从0到x积分的不完全beta函数。如果尾是“上”，则计算从x到1的积分的互补函数。这两个选项通过

Betainc (x, a, b， "upper") = 1 - Betainc (x, a, b， "lower")。

Betainc使用比减法更复杂的算法，当“较低”的值很小时，得到数值精确的结果。

参考:A. Cuyt, V. Brevik Petersen, B. Verdonk, H. Waadeland, W.B. Jones，《特殊函数的连分式手册》，第18章。

参见:beta, betaincinv, betaln。

: x = betaincinv (y, a, b)

: x = betaincinv (y, a, b, "lower")

: x = betaincinv (y, a, b, "upper")

计算归一化不完全函数的逆。

归一化的不完全函数定义为

x

/

1 |

I\_x (a, b) = ---------- | t^(a-1) (1-t)^(b-1) dt

beta (a,b) |

/

0

如果两个输入是标量，则对其他每个输入返回betaincinv (y, a, b)。

如果两个或多个输入不是标量，则它们的大小必须一致，并且逐个元素应用betaincinv。

变量y必须在区间[0,1]内，而a和b必须是实数且严格为正。

默认情况下，尾部是“较低的”，计算从0到x积分的不完全beta函数的逆。如果尾是“上”，则从x到1积分的互补函数是反向的。

这个函数是用标准牛顿法计算的，通过求解

Y - β (x, a, b) = 0

参见:betainc, beta, betaln。

: lnb = betaln (a, b)

计算实输入a和b的函数的自然对数。

beta的定义为

betaln (a, b) = log (beta (a, b))

并以一种减少底流发生的方式进行计算。

Beta函数可以变得非常大，通常处理输出的对数比直接处理函数更有用。

参见beta, betainc, betaincinv, gammaln。

: b = bincoeff (n, k)

返回n和k的二项式系数。

二项式系数定义为

/ \

| n | n (n-1) (n-2) … (n-k+1)

| | = -------------------------

| k | k!

\ /

例如：

bincoeff (5, 2)

⇒ 10

在大多数情况下，nchoosek函数对于小的标量整数参数更快。它还警告大参数的精度损失。

参见:nchoosek。

: k = commutation\_matrix (m, n)

返回交换矩阵K(m,n)，它是唯一的m\*n × m\*n矩阵，使得K(m,n) \* vec(A) = vec(A')对于所有m × n矩阵A。

如果只给出一个参数m，则返回K(m,m)。

参见Magnus和Neudecker(1988)，《矩阵微分在统计学和计量经济学中的应用》。

: y = cosint (x)

计算余弦积分函数:

+oo

/

Ci (x) = - | (cos (t)) / t dt

/

x

一个等价的定义是

x

/

| cos (t) - 1

Ci (x) = gamma + log (x) + | ------------- dt

| t

/

0

参考:

M. Abramowitz和I.A. Stegun，数学函数手册，1964。

参见:sinint, expint, cos。

: d = duplication\_matrix (n)

返回复制矩阵Dn，它是唯一的N^2 × N\*(N+1)/2矩阵，使得对于所有对称N × N矩阵A, Dn \* vech (A) = vec (A)。

参见Magnus和Neudecker(1988)，《矩阵微分在统计学和计量经济学中的应用》。

: v = dawson (z)

计算道森(缩放虚误差)函数。

道森函数定义为

(sqrt (pi) / 2) \* exp (-z^2) \* erfi (z)

参见:erfc, erf, erfcx, erfi, erfinv, erfcinv。

: [sn, cn, dn, err] = ellipj (u, m)

: [sn, cn, dn, err] = ellipj (u, m, tol)

计算复参数u和实参数m的Jacobi椭圆函数sn、cn和dn。

如果m是标量，则结果大小与u相同。如果u是标量，则结果大小与m相同。如果u是列向量，m是行向量，则结果是行长度为u，列长度为m的矩阵。否则，u和m的大小必须一致，结果将与输入的大小相同。

u的值可能是复数。m的值必须为0≤m≤1。

可选的输入工具目前被忽略(MATLAB使用它来允许更快，更不准确的近似)。

如果被请求，err包含以下状态信息，并且与结果大小相同。

正常返回。

错误-没有计算，不满足算法终止条件，返回NaN。

参考:Milton Abramowitz和Irene A Stegun，《数学函数手册》，第16章(16.4,16.13和16.15节)，Dover, 1965。

参见:椭圆。

: k = ellipke (m)

: k = ellipke (m, tol)

: [k, e] = ellipke (…)

计算第一类K(m)和第二类E(m)的完全椭圆积分。

m必须为-Inf≤m≤1的标量或实数组。

可选输入工具控制算法的停止公差，默认为eps(类(m))。可以增加容差来计算更快、更不精确的近似值。

当使用一个输出调用时，只返回第一类的椭圆积分。

数学注意:

第一类椭圆积分定义为

1

/ dt

K (m) = | ------------------------------

/ sqrt ((1 - t^2)\*(1 - m\*t^2))

0

第二类椭圆积分定义为

1

/ sqrt (1 - m\*t^2)

E (m) = | ------------------ dt

/ sqrt (1 - t^2)

0

参考:Milton Abramowitz和Irene A. Stegun，《数学函数手册》，第17章，Dover, 1965年。

参见:ellipj。

: v = erf (z)

计算误差函数。

误差函数定义为

z

2 /

erf (z) = --------- \* | e^(-t^2) dt

sqrt (pi) /

t=0

参见:erfc, erfcx, erfi, dawson, erfinv, erfcinv。

: v = erfc (z)

计算互补误差函数。

互补误差函数定义为1 - erf (z)。

参见:erfcinv, erfcx, erfi, dawson, erf, erfinv。

: v = erfcx (z)

计算缩放后的互补误差函数。

缩放后的互补误差函数定义为

exp (z^2) \* erfc (z)

参见:erfc, erf, erfi, dawson, erfinv, erfcinv。

: v = erfi (z)

计算虚误差函数。

虚误差函数定义为

-i \* erf (i\*z)

参见:erfc, erf, erfcx, dawson, erfinv, erfcinv。

: y = erfinv (x)

计算误差逆函数。

逆误差函数定义如下

erf (y) == x

参见:erf, erfc, erfcx, erfi, dawson, erfcinv。

: y = erfcinv (x)

计算逆互补误差函数。

逆互补误差函数定义如下

erfc (y) == x

参见:erfc, erf, erfcx, erfi, dawson, erfinv。

: y = expint (x)

计算指数积分。

指数积分定义为:

+oo

/

| exp (-t)

E\_1 (x) = | -------- dt

| t

/

x

注:为了兼容，本函数使用MATLAB定义的指数积分。大多数其他来源将这个特定值称为E\_1 (x)，并将指数积分称为

+oo

/

| exp (-t)

Ei (x) = - | -------- dt

| t

/

-x

这两个定义是相关的，对于x的正实数，通过

E\_1 (-x) = -Ei (x) - i\*pi.

引用:

M. Abramowitz和I.A. Stegun，数学函数手册，1964。

N. Bleistein和R.A. Handelsman，积分的渐近展开式，1986。

参见:cos, sinint, exp。

: v = gamma (z)

计算函数。

函数定义为

infinity

/

gamma (z) = | t^(z-1) exp (-t) dt.

/

t=0

编程注意:即使输入值很小，函数也会变大。在许多情况下，最好在计算中使用伽马函数的自然对数(gammaln)，以尽量减少精度损失。最后的结果是exp (result\_using\_gammaln)。

参见:gammainc, gammaln, factorial。

: y = gammainc (x, a)

: y = gammainc (x, a, tail)

计算归一化的不完全函数。

它被定义为

x

1 /

gammainc (x, a) = --------- | exp (-t) t^(a-1) dt

gamma (a) /

t=0

x趋于无穷时的极限值为1。标准符号是P(a,x)，例如Abramowitz和Stegun(6.5.1)。

如果a是标量，则对x的每个元素返回gammainc (x, a)，反之亦然。

如果x和a都不是标量，则x和a的大小必须一致，并且逐个元素应用gammainc。a的元素必须是非负的。

默认情况下，尾部为“较低”，计算从0到x积分的不完全伽马函数。如果尾是“上”，则计算从x到无穷积分的互补函数。

如果tail是“scale - lower”，则较低的不完全函数乘以gamma(a+1)\*exp(x)/(x^a)。如果tail是“scaledupper”，则上端不完全伽马函数乘以相同的量。

引用:

M. Abramowitz和I.A. Stegun，数学函数手册，Dover publications, Inc.， 1972。

W. Gautschi，不完备函数的一种计算方法，计算机科学与工程学报。数学软件，第466-481页，卷5，第4期，2012年。

W. H. Press, S. A. Teukolsky, W. T. Vetterling和B. P. Flannery，《Fortran中的数值配方》77，第6.2章，第1卷，1992年。

参见:gamma, gammaincinv, gammaln。

: x = gammaincinv (y, a)

: x = gammaincinv (y, a, tail)

计算归一化不完全函数的逆。

归一化的不完全函数定义为

x

1 /

gammainc (x, a) = --------- | exp (-t) t^(a-1) dt

gamma (a) /

t=0

对于x的每个非负值，gammaincinv (gammainc (x, a)， a) = x。如果a是标量，则对于y的每个元素返回gammaincinv (y, a)，反之亦然。

如果y和a都不是标量，那么y和a的大小必须一致，并且逐个元素应用gammaincinv。变量y必须在区间[0,1]内，而变量a必须是实数且为正数。

默认情况下，尾部是“较低的”，计算从0到x积分的不完全伽马函数的逆。如果尾是“上”，则从x到无穷积分的互补函数是倒立的。

用牛顿法求解该函数

y - gammainc (x, a) = 0

参考文献:A. Gil, J. Segura, N. M. Temme，不完全伽马函数比值的高效精确计算与反演算法，计算机工程学报。计算，pp. A2965-A2981, Vol . 34, 2012。

参见:gammainc, gamma, gammaln。

: l = legendre (n, x)

: l = legendre (n, x, normalization)

计算相关的n阶、m = 0…n阶的Legendre函数。

n必须是一个实数非负整数。

X是一个实值元素在[- 1,1]范围内的向量。

可选参数normalization可以是"unnorm"、"sch"或"norm"之一。如果没有给出规范化，默认值是“unnorm”。

当可选参数normalization为“unnorm”时，计算相关的n阶和m阶的Legendre函数，并返回m = 0…n的所有值。返回值比x多一个维度。

相关的n阶m阶Legendre函数:

m m 2 m/2 d^m

P(x) = (-1) \* (1-x ) \* ---- P(x)

n dx^m n

n次的勒让德多项式:

1 d^n 2 n

P(x) = ------ [----(x - 1) ]

n 2^n n! dx^n

legendre (3, [-1.0, -0.9, -0.8]) returns the matrix:

x | -1.0 | -0.9 | -0.8

------------------------------------

m=0 | -1.00000 | -0.47250 | -0.08000

m=1 | 0.00000 | -1.99420 | -1.98000

m=2 | 0.00000 | -2.56500 | -4.32000

m=3 | 0.00000 | -1.24229 | -3.24000

当可选参数normalization为"sch"时，计算Schmidt半规范化相关的Legendre函数。Schmidt半规范化关联Legendre函数与非规范化Legendre函数的关系如下:

对于n阶0阶的Legendre函数:

0 0

SP(x) = P(x)

n n

对于n阶、m阶的Legendre函数:

m m m 2(n-m)! 0.5

SP(x) = P(x) \* (-1) \* [-------]

n n (n+m)!

当可选参数normalization为"norm"时，计算完全规范化的相关Legendre函数。完全归一化的关联Legendre函数与未归一化的关联Legendre函数的关系如下:

对于n阶m阶的勒让德函数

m m m (n+0.5)(n-m)! 0.5

NP(x) = P(x) \* (-1) \* [-------------]

n n (n+m)!

: y = gammaln (x)

: y = lgamma (x)

返回x的函数的自然对数。

编程注意:lgamma是gammaln的别名，任何一个名字都可以在Octave中使用。

参见:gamma, gamma。

: y = psi (z)

: y = psi (k, z)

计算psi (polygamma)函数。

多函数是函数对数的k阶导数。如果未指定，k默认为零。值为0计算三角函数，值为1计算三角函数，依此类推。

函数定义为:

psi (z) = d (log (gamma (z))) / dx

当计算digamma函数时(当k等于零时)，z可以是任何实值或复值。然而，对于多函数(k大于0)，z必须是实数且非负的。

参见:伽玛，伽玛，伽玛。

: y = sinint (x)

计算正弦积分函数:

x

/

Si (x) = | sin (t) / t dt

/

0

参考文献:M. Abramowitz和I.A. Stegun，《数学函数手册》，1964年。

参见:cost, expint, sin。

**17.7有理近似**

: s = rat (x)

: s = rat (x, tol)

: [n, d] = rat (…)

在tol定义的公差范围内找到x的有理近似值。

如果未指定，默认容差为1e-6 \* norm (x(:)， 1)。

当使用一个输出参数调用时，返回一个包含连分数展开(多个项)的字符串。

当带两个输出参数调用时，返回x的分数表示形式的分子和分母的数值矩阵，使得x = n ./ d。

例如:

s = rat (pi)

⇒ s = 3 + 1/(7 + 1/16)

[n, d] = rat (pi)

⇒ n = 355

⇒ d = 113

n / d - pi

⇒ 0.00000026676

编程注意:使用一个输出，生成一个连分式展开的字符串。要生成一个简单分数(一个分子，一个分母)的字符串，需要使用老鼠。

参见:老鼠，格式。

: s = rats (x)

: s = rats (x, len)

将x转换为表示为字符串的有理近似值。

浮点数的有理近似值是分子N，分母D的简单分数，使得x = N/D。

可选的第二个参数定义了表示x元素的字符串的最大长度。默认情况下，len为9。

如果最小可能的有理数近似值的长度超过len，则返回一个填充空格的星号(\*)。

示例转换从矩阵到字符串，并再次返回。

r = rats (hilb (4));

x = str2num (r)

参见:鼠，格式。

**17.8坐标变换**

: [theta, r] = cart2pol (x, y)

: [theta, r, z] = cart2pol (x, y, z)

: [theta, r] = cart2pol (C)

: [theta, r, z] = cart2pol (C)

将笛卡尔坐标转换为极坐标或柱坐标。

输入x、y(和z)必须是相同的形状或标量。如果使用单个矩阵参数调用，则C的每一行表示笛卡尔坐标对(x, y)或三元组(x, y, z)。

输出theta, r(和z)与输入的形状相匹配。对于输入矩阵C，输出将是列向量，其行对应于输入矩阵的行。

描述了在x平面上相对于正x轴的角度。

R是到z轴(0,0,z)的距离。

Z，如果存在，则不受变换的影响。

坐标变换的计算方法如下:

theta = arctan (y / x)

r = sqrt (x^2 + y^2)

z = z

注意:为了与MATLAB兼容，当使用单个返回参数调用时，此函数不再返回完整的坐标矩阵。

参见:pol2cart, cart2sph, sph2cart。

: [x, y] = pol2cart (theta, r)

: [x, y, z] = pol2cart (theta, r, z)

: [x, y] = pol2cart (P)

: [x, y, z] = pol2cart (P)

将极坐标或柱坐标转换为笛卡尔坐标。

输入r和z必须是相同的形状，或者标量。如果用单个矩阵参数调用，则P的每一行代表极坐标对(，r)或圆柱形三元组(，r, z)。

输出x、y(和z)与输入的形状相匹配。对于输入P的矩阵，输出将是列向量，其行对应于输入矩阵的行。

描述了在x平面上相对于正x轴的角度。

R是到z轴(0,0,z)的距离。

Z，如果存在，则不受变换的影响。

坐标变换的计算方法如下:

x = r \* cos (theta)

y = r \* sin (theta)

z = z

注意:为了与MATLAB兼容，当使用单个返回参数调用时，此函数不再返回完整的坐标矩阵。

参见:cart2pol, sph2cart, cart2sph。

: [theta, phi, r] = cart2sph (x, y, z)

: [theta, phi, r] = cart2sph (C)

将笛卡尔坐标变换为球坐标。

输入x, y和z必须是相同的形状或标量。如果使用单个矩阵参数调用，则C的每一行必须表示笛卡尔坐标三元组(x, y, z)。

输出和输入的形状相匹配。对于输入矩阵C，输出将是列向量，其行对应于输入矩阵的行。

描述了在x平面上测量的相对于正x轴的方位角。

是相对于xy平面的仰角。

R是到原点(0,0,0)的距离。

坐标变换的计算方法如下:

theta = arctan (y / x)

phi = arctan (z / sqrt (x^2 + y^2))

r = sqrt (x^2 + y^2 + z^2)

注意:对于MATLAB兼容性,这个函数在调用一个返回参数时不再返回一个完整的坐标矩阵。参见另外:sph2cart,2pol,pol2cart。

: [x, y, z] = sph2cart (theta, phi, r)

: [x, y, z] = sph2cart (S)

将球坐标转换为笛卡尔坐标。

输入，和r必须是相同的形状，或者标量。如果用一个矩阵参数调用，那么S的每一行必须代表一个球坐标三元组(，，r)。

输出x, y, z与输入的形状相匹配。对于输入矩阵S，输出是列向量，其行对应于输入矩阵的行。

描述了在x平面上测量的相对于正x轴的方位角。

是相对于xy平面的仰角。

R是到原点(0,0,0)的距离。

坐标变换的计算方法如下:

x = r \* cos (phi) \* cos (theta)

y = r \* cos (phi) \* sin (theta)

z = r \* sin (phi)

注意:为了与MATLAB兼容，当使用单个返回参数调用时，此函数不再返回完整的坐标矩阵。

参见:cart2sph, pol2cart, cart2pol。

**17.9数学常数**

: A = e

: A = e (n)

: A = e (n, m)

: A = e (n, m, k, …)

: A = e (…, class)

返回一个标量、矩阵或n维数组，其元素都等于自然对数的底数。

常数e满足方程log (e) = 1。

当不带参数调用时，返回值为e的标量。

当使用单个参数调用时，返回具有指定维度的方阵。

当使用多个标量参数调用时，前两个参数作为行数和列数，任何其他参数指定额外的矩阵维度。

可选参数class指定返回类型，可以是"double"或"single"。

参见log, exp, pi, I。

: p = pi

: p = pi (n)

: p = pi (n, m)

: p = pi (n, m, k, …)

: p = pi (…, class)

返回一个标量、矩阵或n维数组，其元素都等于圆的周长与其直径的比值。

当不带参数调用时，返回值为pi的标量。

当使用单个参数调用时，返回具有指定维度的方阵。

当使用多个标量参数调用时，前两个参数作为行数和列数，任何其他参数指定额外的矩阵维度。

可选参数class指定返回类型，可以是"double"或"single"。

参见:e, I。

: A = I

: A = I (n)

: A = I (n, m)

: A = I (n, m, k, …)

: A = I (…, class)

返回一个标量、矩阵或n维数组，其元素都等于定义为sqrt(-1)的纯虚单位。

I及其等价的I、j和j都是函数，因此任何名称都可以用于其他目的(例如I作为计数器变量)。

当不带参数调用时，返回值为i的标量。

当使用单个参数调用时，返回具有指定维度的方阵。

当使用多个标量参数调用时，前两个参数作为行数和列数，任何其他参数指定额外的矩阵维度。

可选参数class指定返回类型，可以是"double"或"single"。

参见e, pi, log, exp。

: A = Inf

: A = Inf (n)

: A = Inf (n, m)

: A = Inf (n, m, k, …)

: A = Inf (…, class)

返回一个标量、矩阵或n维数组，其元素都等于正无穷的IEEE表示形式。

当结果太大而无法使用IEEE浮点格式表示时，就会产生Infinity。产生无穷大的两个常见例子是除零和溢出。

[ 1/0 e^800 ]

⇒ Inf Inf

当不带参数调用时，返回值为' Inf '的标量。

当使用单个参数调用时，返回具有指定维度的方阵。

当使用多个标量参数调用时，前两个参数作为行数和列数，任何其他参数指定额外的矩阵维度。

可选参数class指定返回类型，可以是"double"或"single"。

参见:isinf, NaN。

: val = NaN

: val = NaN (n)

: val = NaN (n, m)

: val = NaN (n, m, k, …)

: val = NaN (…, "like", var)

: val = NaN (…, class)

返回一个标量、矩阵或n维数组，其元素都等于IEEE符号NaN (Not a Number)。

NaN是不能产生良好定义的数值结果的运算结果。产生NaN的常见操作是无穷大的算术(Inf - Inf)，零除以零(0/0)，以及涉及另一个NaN值的任何操作(5 + NaN)。

注意，NaN总是比较不等于NaN (NaN != NaN)。此行为由IEEE浮点运算标准指定。要查找NaN值，请使用isnan函数。

当不带参数调用时，返回值为' NaN '的标量。

当使用单个参数调用时，返回具有指定维度的方阵。

当使用多个标量参数调用时，前两个参数作为行数和列数，任何其他参数指定额外的矩阵维度。

如果在"like"后面指定变量var，则输出的val将具有与var相同的数据类型、复杂度和稀疏性。

可选参数class指定返回类型，可以是"double"或"single"。

参见:isnan, Inf。

: d = eps

: d = eps (x)

: d = eps (n, m)

: d = eps (n, m, k, …)

: d = eps (…, class)

返回一个标量、矩阵或n维数组，其元素都是eps(机器精度)。

更准确地说，eps是机器浮点系统中任意两个相邻数字之间的相对间距。这个数字显然与系统有关。在支持IEEE浮点运算的机器上，双精度eps约为2.2204e-16，单精度eps约为1.1921e-07。

当不带参数调用时，返回值为eps(1.0)的标量。

给定一个参数x，返回x与下一个最大值之间的距离。

当使用多个参数调用时，前两个参数作为行数和列数，任何其他参数指定额外的矩阵维度。可选参数class指定返回类型，可以是"double"或"single"。

参见:realmax, realmin, intmax, flintmax。

: Rmax = realmax

: Rmax = realmax (n)

: Rmax = realmax (n, m)

: Rmax = realmax (n, m, k, …)

: Rmax = realmax (…, class)

返回一个标量、矩阵或n维数组，其元素都等于可表示的最大浮点数。

实际值取决于系统。在支持IEEE浮点运算的机器上，双精度的realmax约为1.7977e+308，单精度的realmax约为3.4028e+38。

当不带参数调用时，返回值为realmax ("double")的标量。

当使用单个参数调用时，返回具有指定维度的方阵。

当使用多个标量参数调用时，前两个参数作为行数和列数，任何其他参数指定额外的矩阵维度。

可选参数class指定返回类型，可以是"double"或"single"。

参见:realmin, intmax, flintmax, eps。

: Rmin = realmin

: Rmin = realmin (n)

: Rmin = realmin (n, m)

: Rmin = realmin (n, m, k, …)

: Rmin = realmin (…, class)

返回一个标量、矩阵或n维数组，其元素都等于可表示的最小规范化浮点数。

实际值取决于系统。在支持IEEE浮点运算的机器上，双精度的realmin大约为2.2251e-308，单精度的realmin大约为1.1755e-38。

当不带参数调用时，返回值为realmin ("double")的标量。

当使用单个参数调用时，返回具有指定维度的方阵。

当使用多个标量参数调用时，前两个参数作为行数和列数，任何其他参数指定额外的矩阵维度。

可选参数class指定返回类型，可以是"double"或"single"。

参见:realmax, intmin, eps。

**18线性代数**

本章记录了Octave中提供的线性代数函数。许多这些函数的参考资料可以在Golub和Van Loan，矩阵计算，第二版，约翰霍普金斯大学，1989年和LAPACK用户指南，SIAM, 1992年找到。LAPACK用户指南可在http://www.netlib.org/lapack/lug/上获得

一个常见的文本工程课程是G.斯特朗，线性代数及其应用，第四版。它已成为线性代数的广泛参考。另一种选择是P. Lax线性代数及其应用，也是一个很好的选择。它声称适合对数学有浓厚兴趣的高中生和一年级本科生。

**18.1用于线性代数的技术**

Octave包括一个多态求解器，它根据矩阵本身的性质选择合适的矩阵分解。一般来说，确定矩阵类型的代价相对于分解矩阵本身的代价来说是很小的。在任何情况下，一旦矩阵类型被计算出来，它就会被缓存，这样就不必每次在线性方程中使用它时都重新确定它。

求解线性方程或形成矩阵逆的选择树为:

1. 如果矩阵是上三角形或下三角形稀疏，使用LAPACK xTRTRS函数进行前向或后向替换，然后转到4。
2. 如果矩阵是正方形的，厄密矩阵具有正对角线，尝试使用LAPACK xPOTRF函数进行Cholesky分解。
3. 如果Cholesky分解失败或矩阵不是具有实正对角线的厄米矩阵，并且矩阵是平方的，则使用LAPACK xGETRF函数进行分解。
4. 如果矩阵不是平方的，或者前面的任何求解器标记了一个奇异或接近奇异的矩阵，那么使用LAPACK xGELSD函数找到最小二乘解。

用户可以使用matrix\_type函数强制设置矩阵的类型。这克服了发现矩阵类型的代价。但是，应该注意的是，不正确地识别矩阵的类型将导致不可预测的结果，因此应该谨慎使用matrix\_type。

应该注意的是，上面通过matrix\_type函数执行的关于矩阵是否为Cholesky分解的候选的测试并不能确定矩阵是厄米矩阵。然而，对矩阵进行因式分解的尝试将很快检测到一个非厄米矩阵。

**18.2基本矩阵函数**

: AA = balance (A)

: AA = balance (A, opt)

: [DD, AA] = balance (A, opt)

: [D, P, AA] = balance (A, opt)

: [CC, DD, AA, BB] = balance (A, B, opt)

平衡矩阵A以减少将来计算中的数值误差。

计算AA = DD \ A \* DD，其中AA是一个行范数和列范数大小大致相等的矩阵，DD = P \* D，其中P是一个置换矩阵，D是一个2的幂的对角矩阵。这允许在不舍入的情况下计算平衡。特征值计算的结果通常通过先平衡来改善。

如果请求两个输出值，balance将对角线D和排列P分别作为向量返回。在本例中，DD = eye(n)(:，P) \* diag (D)，其中n为矩阵大小。

如果需要四个输出值，则计算AA = CC\*A\*DD和BB = CC\*B\*DD，其中AA和BB具有大致相同大小的非零元素，CC和DD是与代数特征值问题DD中相同的排列对角矩阵。

特征值平衡选项opt可以是:

"noperm", "S"

规模;不要调换位置。

"noscal", "P"

排列;不要扩展。

代数特征值平衡使用标准LAPACK例程。

广义特征值问题平衡使用Ward算法(SIAM Journal on Scientific and Statistical Computing, 1981)。

: bw = bandwidth (A, type)

: [lower, upper] = bandwidth (A)

计算A的带宽。

类型参数是字符串“lower”表示较低带宽，“upper”表示较高带宽。如果不指定类型，则同时返回A的下带宽和上带宽。

矩阵的下/上带宽是具有非零条目的次对角线/超对角线的个数。

参见:is带状，isdiag, istril, istriu。

: c = cond (A)

: c = cond (A, p)

计算一个矩阵关于反演的p范数条件数。

cond (A)定义为norm (A, p) \* norm (inv (A)， p)。

默认情况下，使用p = 2，这意味着(相对较慢的)奇异值分解。其他可能的选择是p = 1, Inf，“fro”，这些通常更快。有关可能p值的完整讨论，请参见norm。

矩阵的条件数量化了当矩阵元素发生微小变化时矩阵反演操作的灵敏度。理想情况下，条件数将接近于1。当这个数字很大时，这表明小的变化(如下溢或舍入错误)将在结果输出中产生很大的变化。在这种情况下，数值计算的结果不太可能是准确的。

参见:condest, rd, condeig, norm, svd。

: c = condeig (a)

: [v, lambda, c] = condeig (a)

根据特征值计算矩阵的条件数。

条件数是左右特征向量夹角余弦的倒数;大的值表示矩阵有多个不同的特征值。

输入a必须是一个平方数值矩阵。

输出如下:

* C是a的特征值的条件数的向量。
* v是a的右特征向量的矩阵。结果等价于调用[v, lambda] = eig (a)。
* lambda是a的特征值的对角矩阵。结果等价于调用[v, lambda] = eig (a)。

例子

a = [1, 2; 3, 4];

c = condeig (a)

⇒ c =

1.0150

1.0150

参见:第八，第二，平衡。

: d = det (A)

: [d, rcond] = det (A)

计算det (A)

如果请求，返回互反条件数的估计值。

编程注意事项:来自LAPACK的例程用于完整矩阵，来自UMFPACK的代码用于稀疏矩阵。

行列式不应该用来检验矩阵的奇异性。为此，请使用任意条件编号函数:cond、condest、rcond。

参见:cond, condest, record。

: lambda = eig (A)

: lambda = eig (A, B)

: [V, lambda] = eig (A)

: [V, lambda] = eig (A, B)

: [V, lambda, W] = eig (A)

: [V, lambda, W] = eig (A, B)

: […] = eig (A, balanceOption)

: […] = eig (A, B, algorithm)

: […] = eig (…, eigvalOption)

计算一个矩阵或一对矩阵的特征值(lambda)和可选的右特征向量(V)和左特征向量(W)。

标志balanceOption可以是:

"balance" (default)

初步平衡已开启。

"nobalance"

禁用初始均衡。

标志eigvalOption可以是:

"matrix"

返回对角矩阵中的特征值。(如果请求2或3个输出，则默认为)

"vector"

返回列向量中的特征值。(如果只请求一个输出，默认值为lambda = eig (A))

标志算法可以是:

"chol"

使用B的Cholesky分解(默认如果A是对称的(厄米)和B是对称的(厄米)正定)

"qz"

使用QZ算法。(当A或B不对称时使用)

|  | **no flag** | **chol** | **qz** |
| --- | --- | --- | --- |
| both are symmetric | *"chol"* | *"chol"* | *"qz"* |
| at least one is not symmetric | *"qz"* | *"qz"* | *"qz"* |

由eig返回的特征值是无序的。

参见:eigs, svd。

: G = givens (x, y)

: [c, s] = givens (x, y)

计算给定的旋转矩阵G。

给定矩阵是一个2 × 2的正交矩阵

G = [ c , s

-s', c]

以至于

G \* [x; y] = [\*; 0]

用x和y做标量。

如果请求两个输出参数，则返回因子c和s，而不是Givens旋转矩阵。

例如:

givens (1, 1)

⇒ 0.70711 0.70711

-0.70711 0.70711

注:Givens矩阵表示二维平面的逆时针旋转，可用于在完全分解之前将零引入矩阵。

参见:planerot, qr。

: S = gsvd (A, B)

: [U, V, X, C, S] = gsvd (A, B)

: [U, V, X, C, S] = gsvd (A, B, 0)

计算(A, B)的广义奇异值分解。

广义奇异值分解由以下关系定义:

A = U\*C\*X'

B = V\*S\*X'

C'\*C + S'\*S = eye (columns (A))

函数gsvd通常只返回广义奇异值的向量sqrt (diag (C'\*C) ./ diag (S'\*S))。如果请求五个返回值，它还计算U、V、X和C。

如果存在可选的第三个输入，gsvd构建“经济大小”的分解，其中U、V的列数和C、S的行数小于或等于a的列数。

编程注意:代码是相应LAPACK dggsvd和zggsvd例程的包装器。如果矩阵A和B都是秩不足的，那么LAPACK将返回一个不正确的分解。程序员应该避免这种组合。

参见:svd。

: [G, y] = planerot (x)

计算双元素列向量x的Givens旋转矩阵。

给定矩阵是一个2 × 2的正交矩阵

G = [ c , s

-s', c]

以至于

y = G \* [x(1); x(2)] ≡ [\*; 0]

注:Givens矩阵表示二维平面的逆时针旋转，可用于在完全分解之前将零引入矩阵。

参见:given, qr。

: x = inv (A)

: [x, rcond] = inv (A)

: […] = inverse (…)

计算方阵A的逆。

如果请求，返回一个互反条件数的估计值，否则，如果互反条件数很小，则警告一个病态矩阵。

一般来说，最好避免直接计算矩阵的逆。例如，用y = A \ b而不是y = inv (A) \* b来求解方程组(A\*x = b)既快又准确。

如果使用稀疏矩阵调用，那么通常x将是一个需要更多存储空间的完整矩阵。如果可能的话，避免形成稀疏矩阵的逆。

Inverse是一个别名，可以用来代替inv。

编程说明：inverse 是 inv 的别名，可以互换使用。

参见:ldivide, rdivide, pinv。

: x = linsolve (A, b)

: x = linsolve (A, b, opts)

: [x, R] = linsolve (…)

求解线性方程组A\*x = b。

如果没有选项，这个函数相当于左除法运算符(x = A \ b)或矩阵左除法函数(x = mldivide (A, b))。

Octave通常检查矩阵A的性质，并选择最适合矩阵的求解器。通过将结构选项传递给linsolve，你可以直接告知Octave矩阵a。在这种情况下，Octave将跳过矩阵检查，直接求解线性系统。

警告:如果矩阵A没有在options结构中列出的属性，那么结果将不准确并且不会给出警告。当有疑问时，让Octave检查矩阵并选择合适的求解器，因为这一步花费的时间很少，并且结果被缓存，因此每个线性系统只执行一次。

可能的选项字段(设置值为true/false):

LT

A是下三角形

UT

A是上三角形

UHESS

A是上黑森堡(目前没有区别)

SYM

A是对称的或复厄米的(目前没有区别)

POSDEF

A是正定的

RECT

A是一般矩形(目前没有区别)

TRANSA

解A'\*x = b如果为真，而不是A\*x = b

可选的第二个输出R是A的逆条件数(如果矩阵是奇异的，则为零)。

参见:mldivide, matrix\_type, rcond。

: type = matrix\_type (A)

: type = matrix\_type (A, "nocompute")

: A = matrix\_type (A, type)

: A = matrix\_type (A, "upper", perm)

: A = matrix\_type (A, "lower", perm)

: A = matrix\_type (A, "banded", nl, nu)

确定矩阵类型或将矩阵标记为特定类型。

这使得涉及A的线性方程的更快的解得以执行。

使用单个参数调用matrix\_type，返回矩阵的类型并将其缓存以备将来使用。

使用多个参数调用matrix\_type时，可以定义矩阵的类型。

如果给出了选项"nocompute"，则如果类型仍然未知，该函数将不会尝试猜测类型。这对于调试很有用。

可能的矩阵类型取决于矩阵是满的还是稀疏的，可以是以下类型之一

"unknown"

删除任何先前缓存的矩阵类型，并将类型标记为未知。

"full"

将矩阵标记为满。

"positive definite"

可能满正定矩阵。

"diagonal"

对角矩阵。(仅限稀疏矩阵)

"permuted diagonal"

置换对角矩阵。排列不需要特别指出，因为矩阵的结构明确地给出了这一点。(仅限稀疏矩阵)

"upper"

上三角。如果给出了可选的第三个参数perm，则假定该矩阵是一个由向量perm定义的排列的上三角形。

"lower"

下三角。如果给出了可选的第三个参数perm，则假定该矩阵是一个由向量perm定义的排列的下三角形。

"banded"

"banded positive definite"

条带矩阵，在对角线下面条带大小为nl，在对角线上面条带大小为nu。如果nl和nu为1，则矩阵是三对角线的，并且用专门的代码处理。此外，矩阵可以标记为可能是正定的。(仅限稀疏矩阵)

"singular"

假定矩阵是奇异的，用最小范数解来处理。

请注意，矩阵类型将在第一次尝试解决涉及a的线性方程时自动发现。因此，matrix\_type仅用于给出矩阵类型的Octave提示。不正确地定义矩阵类型将导致线性方程解的不正确结果;正确识别矩阵类型完全是用户的责任。

同样，对于具有正对角线的厄米矩阵，正确定性的检验是一种低成本的检验。这并不能保证矩阵是正定的，而只能保证它是一个可能的候选者。当这样的矩阵被分解时，首先尝试Cholesky分解，如果失败，则用LU分解处理矩阵。一旦矩阵被分解，matrix\_type将返回矩阵的正确分类。

: n = norm (A)

: n = norm (A, p)

: n = norm (A, p, opt)

计算矩阵A的p范数。

如果没有给出第二个参数，则使用p = 2。

如果A是矩阵(或稀疏矩阵):

Compute the p-norm of the matrix A.

If the second argument is not given, p = 2 is used.

If A is a matrix (or sparse matrix):

p = 1

1-norm, the largest column sum of the absolute values of A.

p = 2

Largest singular value of A.

p = Inf or "inf"

Infinity norm, the largest row sum of the absolute values of A.

p = "fro"

Frobenius norm of A, sqrt (sum (diag (A' \* A))).

other p, p > 1

maximum norm (A\*x, p) such that norm (x, p) == 1

If A is a vector or a scalar:

p = Inf or "inf"

max (abs (A)).

p = -Inf

min (abs (A)).

p = "fro"

Frobenius norm of A, sqrt (sumsq (abs (A))).

p = 0

Hamming norm—the number of nonzero elements.

other p, p > 1

p-norm of A, (sum (abs (A) .^ p)) ^ (1/p).

other p p < 1

the p-pseudonorm defined as above.

如果opt是值“rows”，则将每行视为一个向量并计算其范数。结果作为列向量返回。类似地，如果opt是“columns”或“cols”，则计算每列的范数并返回一个行向量。

参见:normest, normst1, vecnorm, cond, svd。

: Z = null (A)

: Z = null (A, tol)

返回A的零空间的标准正交基Z。

取零空间Z的维数为A不大于l的奇异值的个数。如果缺少参数tol，则计算为

max (size (A)) \* max (svd (A, 0)) \* eps

参见:north, svd。

: B = orth (A)

: B = orth (A, tol)

返回A的范围空间的一个标准正交基。

极差空间的维数取A大于tol的奇异值的个数。如果缺少参数tol，则计算为

max (size (A)) \* max (svd (A)) \* eps

参见:null。

: [y, h] = mgorth (x, v)

用改进的Gram-Schmidt方法将给定的列向量x与由v的列组成的一组正交向量正交。

在出口，y是一个单位向量，满足:

norm (y) = 1

v' \* y = 0

x = [v, y]\*h'

: B = pinv (A)

: B = pinv (A, tol)

返回A的Moore-Penrose伪逆。

小于tol的奇异值将被忽略。

如果省略第二个参数，则认为是

tol = max ([rows(x), columns(x)]) \* norm (x) \* eps

参见:inv, ldivide。

: k = rank (A)

: k = rank (A, tol)

用奇异值分解计算矩阵A的秩。

秩被认为是大于指定公差的A的奇异值的个数。如果省略第二个参数，则认为是

tol = max (size (A)) \* sigma(1) \* eps;

其中eps为机器精度，sigma(1)为A的最大奇异值。

矩阵的秩是线性无关的行或列的个数，等于行和列空间的维数。函数可以用来计算列空间的标准正交基。

用于测试线性方程组a \*x = b是否可解，可以使用

rank (A) == rank ([A b])

在这种情况下，x = A \ b找到一个特解x，通解是x加上矩阵A的零空间。函数null可以用来计算零空间的一组基。

例子:

A = [1 2 3

4 5 6

7 8 9];

rank (A)

⇒ 2

在这个例子中，线性无关的行数只有2，因为最后一行是前两行的线性组合:

A(3,:) == -A(1,:) + 2 \* A(2,:)

参见:null, north, sprank, svd, eps。

: c = rcond (A)

计算LAPACK返回的互反条件数的1范数估计。

如果矩阵条件良好，那么c将接近1如果矩阵条件不佳，它将接近0。

矩阵A一定不是稀疏的。如果矩阵是稀疏的，则应该使用condest (A)或rcond (full (A))。

参见:condest, condest。

: t = trace (A)

计算A的轨迹，主对角线上元素的和。

实现很简单:sum (diag (A))。

参见:8。

: r = rref (A)

: r = rref (A, tol)

: [r, k] = rref (…)

返回A的行简化阶梯形。

tol默认为eps \* max (size (A)) \* norm (A, inf)。

可选的返回参数k包含“绑定变量”的向量，它是那些已经执行消去的列。

: n = vecnorm (A)

: n = vecnorm (A, p)

: n = vecnorm (A, p, dim)

返回数组A中元素沿维度dim的向量p-范数。

向量的p范数定义为

p-norm (A, p) = sum (abs (A) .^ p)) ^ (1/p)

输入p必须是一个正标量。如果省略，则默认为2(欧几里德范数或距离)。p的其他特殊值是1(曼哈顿范数，绝对值之和)和Inf(最大元素的绝对值)。

输入dim指定函数操作的数组的维度，并且必须是一个正整数。如果省略，则使用第一个非单例维度。

参见:norm。

**18.3矩阵分解**

: R = chol (A)

: [R, p] = chol (A)

: [R, p, Q] = chol (A)

: [R, p, Q] = chol (A, "vector")

: [L, …] = chol (…, "lower")

: [R, …] = chol (…, "upper")

计算实对称或复厄米正定矩阵A的上Cholesky因子R。

上Cholesky因子R由矩阵A的上三角部分计算得到，定义为

R' \* R = A.

使用可选的“upper”标志调用chol具有相同的行为。相反，使用可选的“lower”标志，chol返回下三角形分解，通过使用矩阵A的下三角形部分计算，这样

L \* L' = A.

如果矩阵A不是正定的，用一个输出参数调用chol失败。注意，如果矩阵A不是实对称或复厄米矩阵，则下三角部分被认为是上三角部分的(复共轭)转置，反之亦然，给定“下”标志。

带两个或多个输出参数调用时，p标志矩阵A是否为正定，而chol是否失败。p为零表示矩阵A是正定的，R给出了因数分解。否则，p将是一个正值。

如果用三个输出参数调用矩阵A，则矩阵A必须是稀疏的，并且在分解之前对矩阵A应用保持稀疏性的行/列排列。也就是R是A(Q,Q)的因式分解，使得

R' \* R = Q' \* A \* Q.

保持稀疏性的排列通常以矩阵的形式返回。然而，给定可选标志“vector”，Q将作为这样的矢量返回

R' \* R = A(Q, Q).

一般来说，对于稀疏矩阵，下三角分解的速度要快得多。

参见:hess, lu, qr, qz, schur, svd, ichol, cholinv, chol2inv, cholupdate, cholinsert, choldelete, cholshift.

: Ainv = cholinv (A)

用Cholesky分解法计算对称正定矩阵A的逆。

参见:chol, chol2inv, inv.

: Ainv = chol2inv (R)

从一个对称的正定方阵的Cholesky分解R求逆。

注意R应该是一个对角元素为正的上三角矩阵。chol2inv (U)提供inv (R'\*R)，但比使用inv快得多。

参见:chol, cholinv, inv.

: [R1, info] = (R, u, op)

更新或更新Cholesky分解。

给定一个上三角矩阵R和一个列向量u，尝试确定另一个上三角矩阵R1这样

* R1’\*R1 = R’\*R + u\*u’ if op is "+"
* R1’\*R1 = R’\*R - u\*u’ if op is "-"

如果op为“-”，则info为

* 0 if the downdate was successful,
* 1 if R’\*R - u\*u’ is not positive definite,
* 2 if R is singular.

如果info不存在，则在情形1和情形2中打印错误消息。

参见:chol, cholinsert, choldelete, cholshift.

: R1 = cholinsert (R, j, u)

: [R1, info] = cholinsert (R, j, u)

更新一个Cholesky分解给定的行或列插入到原始分解矩阵。

给定实对称或复厄米正定矩阵a = R ' \*R, R上三角形的Cholesky分解，返回A1的Cholesky分解，其中A1(p,p) = a, A1(:，j) = A1(j，:) ' = u,p = [1:j-1,j+1:n+1]。U (j)应该是正的。

返回时，info被设置为

如果插入成功，则为0;

如果A1不是正定的，

如果R是单数，则是2。

如果info不存在，则在情形1和情形2中打印错误消息。

参见：chol, cholupdate, choldelete, cholshift.

: R1 = choldelete (R, j)

更新一个给定行或列的Cholesky分解，以便从原始分解矩阵中删除。

给定实对称或复厄米正定矩阵a = R ' \*R, R上三角形的Cholesky分解，返回a (p,p)的Cholesky分解，其中p = [1:j-1,j+1:n+1]。

参见:chol, cholupdate, cholinsert, cholshift.

: R1 = cholshift (R, i, j)

更新给定列范围的Cholesky分解，以在原始分解矩阵中移位。

给定实对称或复厄米正定矩阵a = R ' \*R, R上三角形的Cholesky分解，返回a (p,p)的Cholesky分解，其中p为排列

p = [1:i-1, shift(i:j, 1), j+1:n] if i < j

或

p = [1:j-1, shift(j:i,-1), i+1:n] if j < i.

参见:chol, cholupdate, cholinsert, choldelete.

: H = hess (A)

: [P, H] = hess (A)

计算矩阵A的海森伯格分解。

Hessenberg分解是P \* H \* P' = A，其中P是一个正方阵(P' \* P = I，使用复共轭转置)，H是上Hessenberg (H(I, j) = 0，对于所有I > j+1)。

海森伯格分解通常用作特征值计算的第一步，但也有其他应用(参见Golub, Nash, and Van Loan, IEEE Transactions on Automatic Control, 1979)。

参见:eig, chol, lu, qr, qz, schur, svd.

: [L, U] = lu (A)

: [L, U, P] = lu (A)

: [L, U, P, Q] = lu (S)

: [L, U, P, Q, R] = lu (S)

: […] = lu (S, thresh)

: y = lu (…)

: […] = lu (…, "vector")

计算A的LU分解。

如果A是满的，则使用LAPACK的子程序，如果A是稀疏的，则使用UMFPACK。

根据可选的返回值p，以排列形式返回结果。例如，给定矩阵a = [1,2;3、4),

[L, U, P] = lu (A)

返回值

L =

1.00000 0.00000

0.33333 1.00000

U =

3.00000 4.00000

0.00000 0.66667

P =

0 1

1. 0

矩阵不需要是方形的。

当使用两个或三个输出参数和稀疏输入矩阵调用时，lu 不会尝试执行稀疏性保留列排列。使用第四个输出参数进行调用，返回保留稀疏性的列变换 Q，使得 P \* A \* Q = L \* U。这是使用稀疏输入矩阵调用 lu 的首选方法。

使用第五个输出参数和稀疏输入矩阵进行调用，lu 尝试在输入矩阵上使用缩放因子 R，使得 P \* (R \ A) \* Q = L \* U。这通常会导致更稀疏且更稳定的分解。

可以给出定义旋转阈值的附加输入参数 thresh。 thresh 可以是标量，在这种情况下，它定义对称和非对称情况下的 UMFPACK 旋转容差。如果 thresh 是 2 元素矢量，则第一个元素定义非对称 UMFPACK 旋转策略的旋转容差，第二个元素定义对称策略的旋转容差。默认情况下，使用 spparms 定义的值 ([0.1, 0.001])。

给定字符串参数“矢量”，lu 将 P 和 Q 的值作为矢量值返回，这样对于完整矩阵，A(P,:) = L \* U 和 R(P,:) \* A(:,Q ) = L \* U。

如果有两个输出参数，则返回上三角矩阵和下三角矩阵的排列形式，例如 A = L \* U。如果有一个输出参数 y，则返回 LAPACK 例程返回的矩阵。如果输入矩阵是稀疏矩阵，则将矩阵 L 嵌入到 U 中以给出类似于完整情况的返回值。对于满矩阵和稀疏矩阵， lu 都会丢失排列信息。

参见:luupdate, ilu, chol, hess, qr, qz, schur, svd.

: [L, U] = luupdate (L, U, x, y)

: [L, U, P] = luupdate (L, U, P, x, y)

给定实矩阵或复矩阵a = L\*U, L个下单位梯形和U个上单位梯形的LU分解，返回a + x\*y的LU分解。，其中x和y是列向量(rank-1更新)或具有相等列数的矩阵(rank-k更新)。

可选地，行枢轴更新可以通过提供行排列(枢轴)矩阵P来使用;在这种情况下，将返回更新后的排列矩阵。注意，如果L, U, P是由LU得到的主LU分解:

[L, U, P] = lu (A);

然后是a +x\*y的因数分解。的形式可以写成

[L1, U1] = lu (L, U, P\*x, y)

或

[L1, U1, P1] = lu (L, U, P, x, y)

第一种形式使用非枢轴算法，它更快，但不太稳定。第二种形式使用较慢的枢轴算法，它更稳定。

矩阵的情况是作为rank-1更新的序列完成的;因此，对于足够大的k，从头开始重新计算分解会更快更准确。

参见:lu, cholupdate, qrupdate.

: [Q, R] = qr (A)

: [Q, R, P] = qr (A)

: X = qr (A) # non-sparse A

: R = qr (A) # sparse A

: X = qr (A, B) # sparse A

: [C, R] = qr (A, B)

: […] = qr (…, 0)

: […] = qr (…, "econ")

: […] = qr (…, "vector")

: […] = qr (…, "matrix")

计算QR分解A，使用标准LAPACK子程序。

QR分解是

Q \* R = A

其中Q是一个正交矩阵，R是上三角形。

例如，给定矩阵A = [1,2;3、4),

[Q, R] = qr (A)

返回值

Q =

-0.31623 -0.94868

-0.94868 0.31623

R =

-3.16228 -4.42719

0.00000 -0.63246

它们相乘返回原始矩阵

Q \* R

⇒

1.0000 2.0000

3.0000 4.0000

如果只请求一个返回值，那么它要么是R，如果a是稀疏的，要么是X，这样如果a是满的，R = triu (X)。(注意:与大多数命令不同，当请求多个值时，单个返回值不是第一个返回值。)

如果请求第三个输出P，则qr计算排列后的qr分解

Q \* R = A \* P

其中Q为正交矩阵，R为上三角矩阵，P为置换矩阵。

如果A是密集的，排列QR分解具有额外的性质，即R的对角线项是按大小递减排序的。换句话说，abs (diag (R))将按从大到小的顺序排列。

如果A是稀疏的，则P是A列的减填充排序。在这种情况下，R的对角线项不是按递减幅度排序的。

例如，给定矩阵A = [1,2;3、4),

[Q, R, P] = qr (A)

返回值

Q =

-0.44721 -0.89443

-0.89443 0.44721

R =

-4.47214 -3.13050

0.00000 0.44721

P =

0 1

1. 0

如果输入矩阵A是稀疏的，则使用SPQR或CXSPARSE(例如，如果SPQR不可用)计算稀疏QR分解。因为矩阵Q通常是一个完整的矩阵，所以建议只请求一个返回值R。在这种情况下，计算避免了Q的构造，并返回一个稀疏的R，使得R = chol (a ' \* a)。

如果A是密集的，则提供一个额外的矩阵B，并请求两个返回值，则qr返回C，其中C = Q' \* B。这允许A \ B的最小二乘近似计算为

[C, R] = qr (A, B)

X = R \ C

如果A是一个稀疏的MxN矩阵，并且提供了一个额外的矩阵B，则可能有一个或两个返回值。如果请求一个返回值X且M < N，则X是A \ b的最小二范数解。如果M >= N，则X是A \ b的最小二乘逼近。如果请求两个返回值，则C和R具有与密集情况相同的含义(C是密集的，R是稀疏的)。应该优先使用具有一个返回参数的版本，因为它使用更少的内存，并且可以更好地处理缺乏秩的矩阵。

如果最后一个参数是字符串“向量”，那么P是一个置换向量(a的列)，而不是置换矩阵。在本例中，定义关系为:

Q \* R = A(:, P)

然而，默认情况是返回一个排列矩阵，这可以通过使用最后参数"matrix"来显式指定。

如果最后一个参数是标量0或字符串“econ”，则返回经济分解。如果原始矩阵A的大小为MxN且M > N，则经济分解将只计算R中的N行和Q中的N列，省略R中的0。如果M≤N，则经济分解与标准分解没有区别。当计算经济分解时，A是密集的，输出P总是一个向量而不是一个矩阵。如果A是稀疏的，则输出P是一个稀疏排列矩阵。

背景:QR分解在求解最小二乘问题中有一定的应用

min norm (A\*x - b)

对于过定方程组(即，A是一个又高又细的矩阵)。

置换QR分解[Q, R, P] = QR (A)允许构造张成(A)的正交基。

参见:chol, hess, lu, qz, schur, svd, qrupdate, qrinsert, qrdelete, qrshift.

: [Q1, R1] = qrupdate (Q, R, u, v)

给定更新向量或矩阵，更新QR分解。

给定实矩阵或复矩阵a = Q\*R, Q酉矩阵和R上梯形的QR分解，返回a + u\*v '的QR分解，其中u和v是列向量(rank-1更新)或具有相等列数的矩阵(rank-k更新)。请注意，后一种情况是作为rank-1更新的序列完成的;因此，当k足够大时，从头开始重新计算分解会更快更准确。

所提供的QR分解可以是完整的(Q是平方的)或节省的(R是平方的)。

参见:qr, qrinsert, qrdelete, qrshift.

: [Q1, R1] = qrinsert (Q, R, j, x, orient)

更新QR分解给定的行或列插入到原始分解矩阵。

给定实矩阵或复矩阵a = Q\*R, Q酉矩阵和R上梯形的QR分解，返回[a (:，1:j-1) x a (:，j:n)]的QR分解，其中u是插入到a中的列向量(如果orient是“col”)，或者[a (1:j-1，:);x; a (:，j:n)]的QR分解，其中x是插入到a中的行向量(如果orient是“row”)。

orient的默认值是“col”。如果orient为“col”，则u可能是一个矩阵，j是一个索引向量，导致矩阵B的QR分解，使得B(:，j)得到u, B(:，j) =[]得到a。注意，后一种情况是作为k个插入的序列完成的;因此，当k足够大时，从头开始重新计算分解会更快更准确。

如果orient是“冷”，则提供的QR分解可能是满的(Q是正方形的)或节省的(R是正方形的)。

如果orient是“row”，则需要进行完全分解。

参见:qr, qrupdate, qrdelete, qrshift.

: [Q1, R1] = qrdelete (Q, R, j, orient)

更新QR分解给定的行或列，从原始分解矩阵中删除。

给定实矩阵或复矩阵a = Q\*R, Q酉矩阵和R上梯形的QR分解，返回[a (:，1:j-1)， U, a (:，j:n)]的QR分解，其中U是要插入到a中的列向量(如果方向是“col”)，或者[a (1:j-1，:);X; a (:，j:n)]的QR分解，其中X是行方向是“row”)。orient的默认值是“col”。

如果orient为“col”，则j可能是一个索引向量，导致矩阵B的QR分解，使得a (:，j) =[]得到B。注意，后一种情况是作为k删除的序列完成的;因此，当k足够大时，从头开始重新计算分解会更快更准确。

如果orient是“冷”，则提供的QR分解可能是满的(Q是正方形的)或节省的(R是正方形的)。

如果orient是“row”，则需要进行完全分解。

参见:qr, qrupdate, qrinsert, qrshift.

[Q1, R1] = qrshift (Q, R, i, j)

更新QR分解给定列的范围，以在原始分解矩阵中移位。

给定实矩阵或复矩阵a = Q\*R, Q酉矩阵和R上梯形的QR分解，返回a (:，p)的QR分解，其中p为排列

p = [1:i-1, shift(i:j, 1), j+1:n] if i < j

或

p = [1:j-1, shift(j:i,-1), i+1:n] if j < i.

参见:qr, qrupdate, qrinsert, qrdelete.

: [AA, BB, Q, Z, V, W] = qz (A, B)

: [AA, BB, Q, Z, V, W] = qz (A, B, opt)

计算广义特征值问题的QZ分解。

广义特征值问题定义为

A x = lambda B x

函数有两种调用形式:

[AA, BB, Q, Z, V, W, lambda] = qz (A, B)

计算复QZ分解、广义特征向量和广义特征值。

AA = Q \* A \* Z, BB = Q \* B \* Z

A \* V \* diag (diag (BB)) = B \* V \* diag (diag (AA))

diag (diag (BB)) \* W' \* A = diag (diag (AA)) \* W' \* B

AA和BB为上三角形，Q和Z为酉型。矩阵V和W分别包含左、右广义特征向量。

[AA, BB, Z {, lambda}] = qz (A, B, opt)

opt参数必须等于“实”或“复”。如果它等于"complex"，那么这个调用形式等于只有两个输入参数的第一个调用形式。

如果opt等于“real”，则计算实QZ分解。其中，AA只保证为拟上三角形，对角线上有1 × 1和2 × 2块，Q和Z是正交的。上述关于左右广义特征向量的恒等式只有在AA是上三角(即当所有的广义特征值都是实数，此时实QZ和复QZ重合)时才得到验证。

注意:qz执行排列平衡，但不执行缩放(参见balance)，这可能导致比8更不准确的结果。为了与MATLAB兼容，选择了输出参数的顺序。

参见:eig, gsvd, balance, chol, hess, lu, qr, qzhess, schur.

: [aa, bb, q, z] = qzhess (A, B)

计算矩阵pencil (A, B)的hessenberg -三角形分解，返回aa = q \* A \* z, bb = q \* B \* z，其中q和z正交。

例如:

[aa, bb, q, z] = qzhess ([1, 2; 3, 4], [5, 6; 7, 8])

⇒ aa =

-3.02244 -4.41741

0.92998 0.69749

⇒ bb =

-8.60233 -9.99730

0.00000 -0.23250

⇒ q =

-0.58124 -0.81373

-0.81373 0.58124

⇒ z =

Diagonal Matrix

1 0

1. 1

hessenberg -三角形分解是Moler和Stewart的QZ分解算法的第一步。

算法取自Golub和Van Loan，矩阵计算，第二版。

参见:lu, chol, hess, qr, qz, schur, svd.

: S = schur (A)

: S = schur (A, "real")

: S = schur (A, "complex")

: S = schur (A, opt)

: [U, S] = schur (…)

计算A的舒尔分解。

方阵a的舒尔分解定义为

S = U' \* A \* U

其中U是酉矩阵(U'\* U是单位矩阵)，S是上三角矩阵。A(和S)的特征值是S的对角元素。如果矩阵A是实数，则计算实数Schur分解，其中矩阵U是正交的，S是沿对角线的块大小最多为2 x 2的块上三角形。

实矩阵的默认值是实舒尔分解。通过传递“complex”标志可以强制进行复分解。

特征值可根据opt的值沿对角线任意排序:

opt = "a"

将具有负实部的特征值移动到s的前导块中。代数Riccati方程的助记符:“a”，在这里这种排序是有用的。

opt = "d"

将大小小于1的特征值移动到s的前导块上。对于离散代数里卡蒂方程，助记符:“d”，在这里这种排序是有用的。

opt = "u"

无序。特征值没有特定的顺序(默认)。

U的前k列总是张成a不变子空间对应于S的前k个特征值。

参见:rsf2csf, ordschur, ordeig, lu, chol, hess, qr, qz, svd, eig。

: [U, T] = rsf2csf (UR, TR)

将实的上拟三角形舒尔形式TR转换为复的上三角形舒尔形式T。

注意以下关系成立:

UR \* TR \* UR' = U \* T \* U'， U' \* U是单位矩阵I。

还要注意，U和T不是唯一的。

参见:schur。

: [UR, SR] = ordschur (U, S, select)

将用Schur函数得到的实Schur分解(U,S)重新排序，使所选特征值出现在拟三角形Schur矩阵的左上角对角块中。

逻辑向量选择指定所选择的特征值，因为它们沿着S的对角线出现。

例如，给定矩阵A = [1,2;[3,4]，及其Schur分解

[U, S] = schur (A)

其返回值

U =

-0.82456 -0.56577

0.56577 -0.82456

S =

-0.37228 -1.00000

0.00000 5.37228

可以对分解进行重新排序，使正特征值位于左上角，方法如下:

[U, S] = ordschur (U, S, [0,1])

参见:schur, order, ordqz。

: [AR, BR, QR, ZR] = ordqz (AA, BB, Q, Z, keyword)

: [AR, BR, QR, ZR] = ordqz (AA, BB, Q, Z, select)

重新排序广义特征值问题的QZ分解。

广义特征值问题定义为

A x = lambda B x

使用qz算法计算其广义Schur分解:

[AA, BB, Q, Z] = qz (A, B)

AA, BB, Q, Z在哪里

AA = Q \* A \* Z, BB = Q \* B \* Z

ordqz函数计算一个幺正变换QR和ZR，使得AA和BB对角线上的特征值的阶改变。得到的重排序矩阵AR和BR满足:

AR = QR \* A \* ZR, BR = QR \* B \* ZR

该函数可以用关键字参数调用，该参数选择AR和BR的左上角块中的特征值，方法如下:

"S", "udi"

Small:先导块全部|lambda| < 1

"B", "udo"

大:导块全部|lambda|≥1

"-", "lhp"

负实部:导块在开放的左半平面上具有所有特征值

"+", "rhp"

非负实部:导块在封闭的右半平面内具有所有特征值

如果给出逻辑向量选择而不是关键字，则ordqz函数将所有特征值k重新排序到select(k)为真的左侧块。

注意:关键字与qr中的关键字兼容。

参见:eig,ordered, qz, schur, ordschur。

: lambda = ordeig (A)

: lambda = ordeig (A, B)

返回拟三角形矩阵在矩阵A中出现的顺序的特征值。

拟三角矩阵A通常是舒尔分解的结果。如果用第二个输入B调用，则对a, B的广义特征值按矩阵a -lambda\*B的出现顺序返回。对A, B通常是QZ分解的结果。

参见:ordschur, ordqz, eight, schur, qz。

: angle = subspace (A, B)

确定由矩阵A和B的列张成的两个子空间之间的最大主角。

: s = svd (A)

: [U, S, V] = svd (A)

: [U, S, V] = svd (A, "econ")

: [U, S, V] = svd (A, 0)

计算A的奇异值分解。

奇异值分解由关系定义

A = U\*S\*V'

函数svd通常只返回奇异值的向量。当使用三个返回值调用时，它计算U、S和v。例如:

svd (hilb (3))

返回值

ans =

1.4083189

0.1223271

0.0026873

和

[u, s, v] = svd (hilb (3))

返回值

u =

-0.82704 0.54745 0.12766

-0.45986 -0.52829 -0.71375

-0.32330 -0.64901 0.68867

s =

1.40832 0.00000 0.00000

0.00000 0.12233 0.00000

0.00000 0.00000 0.00269

v =

-0.82704 0.54745 0.12766

-0.45986 -0.52829 -0.71375

-0.32330 -0.64901 0.68867

当给定第二个不为0的参数时，svd返回经济大小的分解，消除U或V中不必要的行或列。

如果第二个参数恰好为0，则分解的选择基于矩阵A。如果A的行数多于列数，则返回经济大小的分解，否则计算常规分解。

算法说明:在计算完全分解(除了奇异值外，左、右奇异矩阵)时，LAPACK中有两个例程可供选择。Octave使用的默认例程是gesvd。另一种方法是gessd，它比gessd快5倍，但可能使用更多内存，并且对于某些输入矩阵可能不准确。还有第三种常规gejsv，适用于在极端尺度下获得更好的精度。有关选择驱动程序的更多信息，请参阅svd\_driver的文档。

参见:svd\_driver, svds, eig, lu, chol, hess, qr, qz。

: val = svd\_driver ()

: old\_val = svd\_driver (new\_val)

: old\_val = svd\_driver (new\_val, "local")

查询或设置svd使用的底层LAPACK驱动进程。

目前公认的值为“gesdd”、“gesvd”和“gejsv”。默认值为“gesvd”。

当使用“local”选项从函数内部调用时，变量会在函数及其调用的任何子例程中本地更改。退出函数时恢复原始变量值。

算法注释： LAPACK 库例程 gesvd 和 gesdd 仅在计算完整奇异值分解（左奇异矩阵和右奇异矩阵以及奇异值）时不同。当仅计算奇异值时，以下讨论不相关。

较新的 gesdd 例程基于分而治之算法，该算法比基于 QR 分解的替代 gesvd 快 5 倍。然而，新算法可以使用更多的内存。对于 MxN 输入矩阵，内存使用量的数量级为 O(min(M,N) ^ 2)，而替代矩阵的数量级为 O(max(M,N))。

例程 gejsv 使用预处理的 Jacobi SVD 算法。与 gesvd 和 gesdd 不同，在 gejsv 中，不存在在某些极端情况下可能会影响准确性的双对角化步骤。此外，众所周知，gejsv 在某种意义上是最准确的。然而，速度较慢（其核心是单线程）并且使用更多内存（O(min(M,N) ^ 2 M N)）。

除了速度和内存问题之外，还存在一些输入矩阵无法被 gesdd 准确分解的情况。查看当前活跃的错误 https://savannah.gnu.org/bugs/?55564。在新版本的 LAPACK 库中解决这些准确性问题之前，Octave 中的默认驱动进程已设置为“gesvd”。

参见：svd。

: [housv, beta, zer] = housh (x, j, z)

计算Householder反射向量housv，将x反射为单位矩阵的第j列，即

(I - beta\*housv\*housv')x = norm (x)\*e(j) if x(j) < 0,

(I - beta\*housv\*housv')x = -norm (x)\*e(j) if x(j) >= 0

输入

x

向量

j

向量索引

z

零阈值(通常应该是数字0)

产出(见Golub和Van Loan):

beta

如果beta = 0，则不需要施加反射(zer设置为0)

housv

户主向量

: [u, h, nu] = krylov (A, V, k, eps1, pflg)

构造块克雷洛夫子空间的正交基u。

块Krylov子空间具有如下形式:

[v a\*v a^2\*v ... a^(k+1)\*v]

该建筑采用了户主反射，以防止失去正交性。

如果V是向量，则h包含满足a\*u == u\*h+rk\*ek'的Hessenberg矩阵，其中rk = a\*u(:，k)-u\*h(:，k)， ek'为长度为k的向量[0,0，…，1]，否则h没有意义。

如果V是一个向量，且k大于长度(a) - 1，则h包含使得a\*u == u\*h的海森伯格矩阵。

nu的值是Krylov子空间张成的维数(基于eps1)。

如果b是向量且k大于m-1，则h包含a的Hessenberg分解。

可选参数eps1是零的阈值。默认值为1e-12。

如果可选参数pflg非零，则使用行枢轴来改进数值行为。缺省值为0。

参考文献:A. Hodel, P. Misra，大型稀疏系统的Krylov子空间计算中的偏轴，第42届IEEE决策与控制会议论文集，2003年12月。

**18.4矩阵的函数**

: r = expm (A)

返回矩阵的指数。

矩阵指数被定义为无穷泰勒级数

expm (A) = I + A + A^2/2! + A^3/3! + ...

然而，泰勒级数不是计算矩阵指数的方法;参见Moler和Van Loan，计算矩阵指数的十九种可疑方法，SIAM Review, 1978。该例程使用Ward的对角pad<s:1>近似方法和三步预处理(SIAM Journal on Numerical Analysis, 1977)。对角逼近是矩阵的有理多项式

-1

D (A) N (A)

它的泰勒级数匹配上面泰勒级数的前2q+1项;当Dq(A)是病态的时，可能需要直接评估泰勒级数(具有相同的预处理步骤)来代替pad<s:1>近似。

参见:logm, sqrtm。

: s = logm (A)

: s = logm (A, opt\_iters)

: [s, iters] = logm (…)

计算方阵A的矩阵对数。

该实现利用了一个帕德帕近似和恒等式

logm (A) = 2^k \* logm (A^(1 / 2^k))

可选的输入opt\_iters是要计算的平方根的最大数目，默认为100。

可选的输出数是实际计算的平方根的个数。

参见:expm, sqrtm。

: s = sqrtm (A)

: [s, error\_estimate] = sqrtm (A)

计算矩阵A的平方根。

参考:N.J.海厄姆。一个新的MATLAB sqrtm。数值分析报告第336号，曼彻斯特计算数学中心，曼彻斯特，英国，1999年1月。

参见:expm, logm。

: C = kron (A, B)

: C = kron (A1, A2, …)

形成两个或多个矩阵的克罗内克积。

它被逐块定义为

c = [ a(i,j)\*b ]

例如：

kron (1:4, ones (3, 1))

⇒ 1 2 3 4

1 2 3 4

1 2 3 4

如果有两个以上的输入参数A1, A2，…，An, Kronecker积计算为

kron (kron (A1, A2), ..., An)

因为克罗内克积是结合式的，所以这个定义很好。

参见:tensorprod。

: C = tensorprod (A, B, dimA, dimB)

: C = tensorprod (A, B, dim)

: C = tensorprod (A, B)

: C = tensorprod (A, B, "all")

: C = tensorprod (A, B, …, "NumDimensionsA", value)

计算数值张量A和B之间的张量积。

收缩的A和B的维度分别由dimA和dimB定义。dimA和dimB是定义要匹配的维度的标量或等长向量。A和B的匹配维度必须具有相同数量的元素。

当只使用dim时，它等价于dimA = dimB = dim。

当没有指定维度时，dimA = dimB =[]。它计算的是A和B的外积。

使用"all"选项会导致A和B之间的内积。这要求size (A) == size (B)。

当A具有应该转移到c的尾随单维时，使用属性名称为“NumDimensionsA”的属性-值对。指定的值应该是A的总维数。

MATLAB兼容性:Octave目前不支持“NumDimensionsA”参数的“property\_name=value”语法。

参见:kron, dot, mtimes。

: C = blkmm (A, B)

计算矩阵块的乘积。

这些块以数组A, B的二维子数组形式给出。A的大小必须为[m,k，…]，B的大小必须为[k,n，…]。结果的大小为[m,n，…]，计算方法如下:

for i = 1:prod (size (A)(3:end))

C(:,:,i) = A(:,:,i) \* B(:,:,i)

endfor

: X = sylvester (A, B, C)

解西尔维斯特方程。

Sylvester方程定义为:

A X + X B = C

解是使用标准LAPACK子程序计算的。

例如:

sylvester ([1, 2; 3, 4], [5, 6; 7, 8], [9, 10; 11, 12])

⇒ [ 0.50000, 0.66667; 0.66667, 0.50000 ]

**18.5专门的求解器**

: x = bicg (A, b)

: x = bicg (A, b, tol)

: x = bicg (A, b, tol, maxit)

: x = bicg (A, b, tol, maxit, M)

: x = bicg (A, b, tol, maxit, M1, M2)

: x = bicg (A, b, tol, maxit, M, [], x0)

: x = bicg (A, b, tol, maxit, M1, M2, x0)

: x = bicg (A, b, tol, maxit, M, [], x0, …)

: x = bicg (A, b, tol, maxit, M1, M2, x0, …)

: [x, flag, relres, iter, resvec] = bicg (A, b, …)

用双共轭梯度迭代法求解线性方程组A \* x = b。

输入参数是:

* A是线性系统的矩阵它必须是平方的。A可以作为矩阵、函数句柄或内联函数Afcn传递，使得Afcn (x， "notransp") = A \* x和Afcn (x， "transp") = A' \* x。Afcn的附加参数可以在x0之后传递。
* B是右边的向量。它必须是一个列向量，和a有相同的行数。
* tol是残余误差b - A \* x所需的相对容差。如果norm (b - A \* x)≤tol \* norm (b)，则迭代停止。如果tol省略或为空，则使用1e-6的容差。
* Maxit是允许的最大迭代次数;如果maxit省略或为空，则使用值20。
* M1, M2是预热剂。预条件M为M = M1 \* M2。M1和M2都可以作为矩阵或函数句柄或内联函数g传递，使得g (x， "notransp") = M1 \ x或g (x， "notransp") = M2 \ x和g (x， "transp") = M1' \ x或g (x， "transp") = M2' \ x。如果M1省略或为空，则不应用预处理。前置系统在理论上相当于将bicg方法应用于线性系统inv (M1) \* A \* inv (M2) \* y = inv (M1) \* b和inv (M2') \* A' \* inv (M1') \* z = inv (M2') \* b，然后设x = inv (M2) \* y。
* X0是最初的猜测。如果x0省略或为空，则该函数默认将x0设置为零向量。

x0后面的任何参数都被视为参数，并以适当的方式传递给任何函数(Afcn或Mfcn)或已给定给bci的函数。

输出参数为:

* x为A \* x = b解的计算近似值，如果算法不收敛，则x为残差最小的迭代。
* Flag表示退出状态:
* 0:算法收敛到规定公差内。
* 1:算法不收敛，达到最大迭代次数。
* 2:预条件矩阵是奇异的。
* 3:算法停滞，即当前迭代x与前一次迭代差值的绝对值小于eps \* norm (x,2)。
* 4:算法无法继续，因为中间值太小或太大，无法进行可靠的计算。
* relres是最终残差与其初始值的比值，用欧几里得范数表示。
* Iter是计算x的迭代。
* Resvec是一个包含每次迭代残差的向量。执行的迭代总数由length (resvec) - 1给出。

考虑一个关于三对角矩阵的平凡问题

n = 20;

A = toeplitz (sparse ([1, 1], [1, 2], [2, 1] \* n ^ 2, 1, n)) + ...

toeplitz (sparse (1, 2, -1, 1, n) \* n / 2, ...

sparse (1, 2, 1, 1, n) \* n / 2);

b = A \* ones (n, 1);

restart = 5;

[M1, M2] = ilu (A); # in this tridiag case, it corresponds to lu (A)

M = M1 \* M2;

Afcn = @(x, string) strcmp (string, "notransp") \* (A \* x) + ...

strcmp (string, "transp") \* (A' \* x);

Mfcn = @(x, string) strcmp (string, "notransp") \* (M \ x) + ...

strcmp (string, "transp") \* (M' \ x);

M1fcn = @(x, string) strcmp (string, "notransp") \* (M1 \ x) + ...

strcmp (string, "transp") \* (M1' \ x);

M2fcn = @(x, string) strcmp (string, "notransp") \* (M2 \ x) + ...

strcmp (string, "transp") \* (M2' \ x);

例1:bicg的最简单用法

x = bicg (A, b)

例2:一个计算a \*x和a '\*x的函数

x = bicg (Afcn, b, [], n)

例3:具有预条件矩阵M的矩阵

x = bicg (A, b, 1e-6, n, M)

例4:用函数作为前置条件的函数

x = bicg (Afcn, b, 1e-6, n, Mfcn)

例5:具有预条件矩阵M1和M2的矩阵

x = bicg (A, b, 1e-6, n, M1, M2)

例6:将函数用作前置条件

x = bicg (Afcn, b, 1e-6, n, M1fcn, M2fcn)

例7:将需要参数的函数作为输入

function y = Ap (A, x, string, z)

## compute A^z \* x or (A^z)' \* x

y = x;

if (strcmp (string, "notransp"))

for i = 1:z

y = A \* y;

endfor

elseif (strcmp (string, "transp"))

for i = 1:z

y = A' \* y;

endfor

endif

endfunction

Apfcn = @(x, string, p) Ap (A, x, string, p);

x = bicg (Apfcn, b, [], [], [], [], [], 2);

参考:

杨志强，稀疏线性系统的迭代方法，第2版，2003年。

另请参见:bigstab、cgs、gmres、pcg、qmr、tfqmr。

: x = bicgstab (A, b, tol, maxit, M1, M2, x0, …)

: x = bicgstab (A, b, tol, maxit, M, [], x0, …)

: [x, flag, relres, iter, resvec] = bicgstab (A, b, …)

用稳定双共轭梯度迭代法求解A x = b。

输入参数为:

* + A是线性系统的矩阵它必须是平方的。A可以作为矩阵、函数句柄或内联函数Afcn传递，例如Afcn(x) = A \* x。Afcn的附加参数在x0之后传递。
  + B是右边的向量。它必须是一个列向量，和a有相同的行数。
  + tol是残余误差b - A \* x所需的相对容差。如果norm (b - A \* x)≤tol \* norm (b)，则迭代停止。如果tol省略或为空，则使用1e-6的容差。
  + Maxit外部迭代的最大次数，如果没有给出或设置为[]，则使用默认值min (20, numel (b))。
  + M1,M2是前置调节器。预调节器M为M = M1 \* M2。M1和M2都可以作为一个矩阵传递,或者作为一个函数句柄或内联函数g,这样g(x)= M1 \ x或g(x)= M2 \ x,使用的技术是正确的预处理,即。,它得到了一个\*发票(M)\* y = b,然后是x = invoice(M)\* y。
  + x0最初的猜测,如果没有给出或设置为[]默认值0(大小(b))被使用。

遵循x0的参数被视为参数,并以适当的方式传递给一个被传递到bicstab的函数(a或M)。

输出参数为:

* X是计算出的近似值。如果该方法不收敛，则它是残差最小的迭代方法。
* Flag表示退出状态:
* 0:迭代收敛到所选容差范围内
* 1:达到收敛前的最大迭代次数
* 2:预条件矩阵是奇异的
* 3:算法达到停滞
* 4:由于被零除，算法无法继续
* relres为as (A\*x-b) / norm(b)得到的相对残差。
* Iter是计算x的(可能是一半)迭代。如果是半迭代，则为iter + 0.5
* resvec是一个包含每次半迭代和总迭代的残差的向量(也有半迭代，因为x在每次迭代中分两步计算)。执行(length(resvec) - 1) / 2可以看到执行的总迭代次数。

让我们考虑一个关于三对角矩阵的平凡问题

n = 20;

A = toeplitz (sparse ([1, 1], [1, 2], [2, 1] \* n ^ 2, 1, n)) + ...

toeplitz (sparse (1, 2, -1, 1, n) \* n / 2, ...

sparse (1, 2, 1, 1, n) \* n / 2);

b = A \* ones (n, 1);

restart = 5;

[M1, M2] = ilu (A); # in this tridiag case, it corresponds to lu (A)

M = M1 \* M2;

Afcn = @(x) A \* x;

Mfcn = @(x) M \ x;

M1fcn = @(x) M1 \ x;

M2fcn = @(x) M2 \ x;

例1:最简单的bigstab用法

x = bicgstab (A, b, [], n)

例2:带有计算a \* x的函数的bigstab

x = bicgstab (Afcn, b, [], n)

例3:带有预条件矩阵M的bigstab

x = bicgstab (A, b, [], 1e-06, n, M)

例4:使用函数作为前置条件的bigstab

x = bicgstab (Afcn, b, 1e-6, n, Mfcn)

例5:具有预条件矩阵M1和M2的bigstab

x = bicgstab (A, b, [], 1e-6, n, M1, M2)

例6:使用函数作为前置条件的bigstab

x = bicgstab (Afcn, b, 1e-6, n, M1fcn, M2fcn)

例7:将需要参数的函数作为输入的bigstab

function y = Ap (A, x, z) # compute A^z \* x

y = x;

for i = 1:z

y = A \* y;

endfor

endfunction

Apfcn = @(x, string, p) Ap (A, x, string, p);

x = bicgstab (Apfcn, b, [], [], [], [], [], 2);

例8:显示bigstab使用正确的前置条件的显式示例

[M1, M2] = ilu (A + 0.1 \* eye (n)); # factorization of A perturbed

M = M1 \* M2;

## reference solution computed by bicgstab after one iteration

[x\_ref, fl] = bicgstab (A, b, [], 1, M)

## right preconditioning

[y, fl] = bicgstab (A / M, b, [], 1)

x = M \ y # compare x and x\_ref

参考:

杨志强，稀疏线性系统的迭代方法，第2版，2003年

参见:bicg, cgs, gmres, pcg, qmr, tfqmr。

: x = cgs (A, b, tol, maxit, M1, M2, x0, …)

: x = cgs (A, b, tol, maxit, M, [], x0, …)

: [x, flag, relres, iter, resvec] = cgs (A, b, …)

用共轭梯度平方法求解A x = b，其中A是一个方阵。

输入参数是:

* A是线性系统的矩阵它必须是平方的。A可以作为矩阵、函数句柄或内联函数Afcn传递，例如Afcn(x) = A \* x。Afcn的附加参数在x0之后传递。
* B是右边的向量。它一定是a的列向量有相同的行数。
* Tol是相对容差，如果没有给出或设置为[]，则使用默认值1e-6。
* Maxit外部迭代的最大次数，如果没有给出或设置为[]，则使用默认值min (20, numel (b))。
* M1, M2是预热剂。预条件矩阵为M = M1 \* M2。M1和M2都可以作为矩阵或函数句柄或内联函数g传递，使得g(x) = M1 \ x或g(x) = M2 \ x。如果M1为空或未传递则不应用前置条件。所使用的技术是正确的预处理，即求解A\*inv(M)\*y = b，然后x = inv(M)\*y。
* X0是初始猜测值，如果没有给出或设置为[]，则使用默认值0 (size (b))。

x0后面的参数被视为参数，并以适当的方式传递给传递给cgs的任何函数(a或P)。

输出参数为:

* X是计算出的近似值。如果该方法不收敛，则它是残差最小的迭代方法。
* Flag表示退出状态:
* 0:迭代收敛到所选容差范围内
* 1:达到收敛前的最大迭代次数
* 2:预条件矩阵是奇异的
* 3:算法达到停滞
* 4:算法因除零而无法继续，relres为as (a \*x-b) / norm(b)得到的相对残差。
* Iter是计算x的迭代。
* Resvec是一个包含每次迭代残差的向量。执行length(resvec) - 1可以查看执行的迭代总数。
* 让我们考虑一个关于三对角矩阵的平凡问题

n = 20;

A = toeplitz (sparse ([1, 1], [1, 2], [2, 1] \* n ^ 2, 1, n)) + ...

toeplitz (sparse (1, 2, -1, 1, n) \* n / 2, ...

sparse (1, 2, 1, 1, n) \* n / 2);

b = A \* ones (n, 1);

restart = 5;

[M1, M2] = ilu (A); # in this tridiag case it corresponds to chol (A)'

M = M1 \* M2;

Afcn = @(x) A \* x;

Mfcn = @(x) M \ x;

M1fcn = @(x) M1 \ x;

M2fcn = @(x) M2 \ x;

例1:cgs的最简单用法

x = cgs (A, b, [], n)

例2:cgs与计算a \* x的函数

x = cgs (Afcn, b, [], n)

例3:具有预条件矩阵M的cgs

x = cgs (A, b, [], 1e-06, n, M)

例4:使用函数作为前置条件的cgs

x = cgs (Afcn, b, 1e-6, n, Mfcn)

例5:具有预条件矩阵M1和M2的cgs

x = cgs (A, b, [], 1e-6, n, M1, M2)

例6:具有前置功能的cgs

x = cgs (Afcn, b, 1e-6, n, M1fcn, M2fcn)

例7:cgs将需要参数的函数作为输入

function y = Ap (A, x, z) # compute A^z \* x

y = x;

for i = 1:z

y = A \* y;

endfor

endfunction

Apfcn = @(x, string, p) Ap (A, x, string, p);

x = cgs (Apfcn, b, [], [], [], [], [], 2);

例8:显示cgs使用正确前置条件的显式示例

[M1, M2] = ilu (A + 0.3 \* eye (n)); # factorization of A perturbed

M = M1 \* M2;

## reference solution computed by cgs after one iteration

[x\_ref, fl] = cgs (A, b, [], 1, M)

## right preconditioning

[y, fl] = cgs (A / M, b, [], 1)

x = M \ y # compare x and x\_ref

引用:

杨志强，稀疏线性系统的迭代方法，第2版，2003年

参见:pcg, bigstab, bigg, gmres, qmr, tfqmr。

: x = gmres (A, b, restart, tol, maxit, M1, M2, x0, …)

: x = gmres (A, b, restart, tol, maxit, M, [], x0, …)

: [x, flag, relres, iter, resvec] = gmres (A, b, …)

用带重启的Preconditioned GMRES迭代法求解A x = b，又称PGMRES(restart)。

输入参数是:

* A是线性系统的矩阵它必须是平方的。A可以作为矩阵、函数句柄或内联函数Afcn传递，例如Afcn(x) = A \* x。Afcn的附加参数在x0之后传递。
* B是右边的向量。它必须是一个列向量，和a有相同的行数。
* Restart是方法重新启动之前的迭代次数。如果是[]或N = numel (b)，则不应用重启。
* tol是预条件残差inv (M) \* (b - a \* x)所需的相对容差。如果norm (inv (M) \* (b - a \* x))≤tol \* norm (inv (M) \* b)，则迭代停止。如果tol省略或为空，则使用1e-6的容差。
* maxit是外部迭代的最大次数，如果没有给出或设置为[]，则使用默认值min (10, N / restart)。注意，如果restart为空，则maxit是迭代的最大次数。如果restart和maxit不为空，则迭代的最大次数为restart \* maxit。如果restart和maxit都为空，则最大迭代次数设置为min (10, N)。
* M1, M2是预热剂。预条件M为M = M1 \* M2。M1和M2都可以作为矩阵、函数句柄或内联函数g传递，使得g(x) = M1 \ x或g(x) = M2 \ x。如果M1为[]或未给出，则不应用前置条件。所用的技术是左预处理，即求解inv(M) \* A \* x = inv(M) \* b，而不是A \* x = b。
* X0是初始猜测，如果没有给出或设置为[]，则使用默认值0 (size (b))。

x0后面的参数被视为参数，并以适当的方式传递给传递给gmres的任何函数(a或M或M1或M2)。

输出如下:

-x是计算近似值。如果该方法不收敛，则为残差最小迭代。

-flag表示退出状态:

0:迭代收敛到指定公差范围内

1:最大迭代次数

2:预条件矩阵是奇异的

3:算法达到停滞(连续两次迭代的相对差小于eps)

relres是近似x的相对预条件残差的值。

iter是一个包含计算x的外部迭代次数和内部迭代次数的向量，即:

* Iter(1):外部迭代的次数，即方法重新启动的次数。(如果restart为空或N，则为1，如果不是1≤iter(1)≤maxit)。
* Iter(2):重启前执行的迭代次数，即当Iter (2) = restart时，方法重启。如果restart为空或为N，则1≤iter(2)≤maxit。

为了更清楚，近似值x是在迭代(iter(1) - 1) \* restart + iter(2)时计算的。由于输出x对应于最小预条件残差解，因此该方法执行的总迭代次数由length (resvec) - 1给出。

-resvec是一个向量，它包含每次迭代的预条件相对残差，包括第0次迭代范数(a \* x0 - b)。

让我们考虑一个关于三对角矩阵的平凡问题

n = 20;

A = toeplitz (sparse ([1, 1], [1, 2], [2, 1] \* n ^ 2, 1, n)) + ...

toeplitz (sparse (1, 2, -1, 1, n) \* n / 2, ...

sparse (1, 2, 1, 1, n) \* n / 2);

b = A \* ones (n, 1);

restart = 5;

[M1, M2] = ilu (A); # in this tridiag case, it corresponds to lu (A)

M = M1 \* M2;

Afcn = @(x) A \* x;

Mfcn = @(x) M \ x;

M1fcn = @(x) M1 \ x;

M2fcn = @(x) M2 \ x;

例1:gmres的最简单用法

x = gmres (A, b, [], [], n)

例2:gmres和一个计算a \* x的函数

x = gmres (Afcn, b, [], [], n)

例3:重启时使用gmres

x = gmres (A, b, restart);

例4:在重启和不重启的情况下，具有预调理矩阵M的gmres

x = gmres (A, b, [], 1e-06, n, M)

x = gmres (A, b, restart, 1e-06, n, M)

例5:用函数作为前置条件的gmres

x = gmres (Afcn, b, [], 1e-6, n, Mfcn)

例6:带有预条件矩阵M1和M2的gmres

x = gmres (A, b, [], 1e-6, n, M1, M2)

例7:具有前置功能的gmres

x = gmres (Afcn, b, 1e-6, n, M1fcn, M2fcn)

例8:输入一个需要参数的函数

function y = Ap (A, x, p) # compute A^p \* x

y = x;

for i = 1:p

y = A \* y;

endfor

endfunction

Apfcn = @(x, p) Ap (A, x, p);

x = gmres (Apfcn, b, [], [], [], [], [], [], 2);

例9:显示gmres使用左前置条件的显式示例

[M1, M2] = ilu (A + 0.1 \* eye (n)); # factorization of A perturbed

M = M1 \* M2;

## reference solution computed by gmres after two iterations

[x\_ref, fl] = gmres (A, b, [], [], 1, M)

## left preconditioning

[x, fl] = gmres (M \ A, M \ b, [], [], 1)

x # compare x and x\_ref

参考:

杨志强，稀疏线性系统的迭代方法，第2版，2003年

参见:bigg, bigstab, cgs, pcg, pcr, qmr, tfqmr。

: x = qmr (A, b, rtol, maxit, M1, M2, x0)

: x = qmr (A, b, rtol, maxit, P)

: [x, flag, relres, iter, resvec] = qmr (A, b, …)

用拟最小残差迭代法求解A x = b(无预查)。

Rtol是相对公差，如果没有给出或设置为[]，则使用默认值1e-6。

Maxit外部迭代的最大次数，如果没有给出或设置为[]，则使用默认值min (20, numel (b))。

X0是初始猜测值，如果没有给出或设置为[]，则使用默认值0 (size (b))。

A可以作为矩阵或函数句柄或内联函数f传递，使得f(x， "notransp") = A\*x和f(x， "transp") = A'\*x。

预调节器P为P = M1 \* M2。M1和M2都可以作为矩阵或函数句柄或内联函数g传递，使得g(x， "notransp") = M1 \ x或g(x， "notransp") = M2 \ x和g(x， "transp") = M1 \ x或g(x， "transp") = M2' \ x。

如果使用多个输出参数调用

Flag表示退出状态:

0:迭代收敛到所选容差范围内

1:达到收敛前的最大迭代次数

3:算法达到停滞

(值2是未使用的，但为了兼容性而跳过)。

Relres是相对残差的最终值。

Iter是执行的迭代次数。

Resvec是一个包含每次迭代的残差范数的向量。

引用:

R. Freund和N. Nachtigal . QMR:非厄米线性系统的拟极小残差方法，数学学报，1991,60,pp. 315-339。

R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhour, R. Pozo, C. Romine和H. van der Vorst，线性系统解的模板:迭代方法的构建块，SIAM，第2版，1994。

参见:bigg, bigstab, cgs, gmres, pcg。

: x = tfqmr (A, b, tol, maxit, M1, M2, x0, …)

: x = tfqmr (A, b, tol, maxit, M, [], x0, …)

: [x, flag, relres, iter, resvec] = tfqmr (A, b, …)

基于cgs，使用转置树qmr方法求解A x = b。

输入参数为:

A是线性系统的矩阵它必须是平方的。A可以作为矩阵、函数句柄或内联函数Afcn传递，例如Afcn(x) = A \* x。Afcn的附加参数在x0之后传递。

B是右边的向量。它必须是一个列向量，和a有相同的行数。

Tol是相对容差，如果没有给出或设置为[]，则使用默认值1e-6。

Maxit外部迭代的最大次数，如果没有给出或设置为[]，则使用默认值min (20, numel (b))。为了兼容，由于该方法作为不同行为中的迭代次数是奇数还是偶数，因此在tfqmr中认为迭代是整个奇偶循环。也就是说，为了完成一个完整的迭代，算法执行两个子迭代:奇数迭代和偶数迭代。

M1, M2是预热剂。预条件M为M = M1 \* M2。M1和M2都可以作为矩阵或函数句柄或内联函数g传递，使得g(x) = M1 \ x或g(x) = M2 \ x。使用的技术是正确的预处理，即求解a \*inv(M)\*y = b，然后x = inv(M)\*y，而不是a x = b。

X0是初始猜测值，如果没有给出或设置为[]，则使用默认值0 (size (b))。

x0后面的参数被视为参数，并以适当的方式传递给传递给tfqmr的任何函数(a或M)。

输出参数为:

X是计算出的近似值。如果该方法不收敛，则它是残差最小的迭代方法。

Flag表示退出状态:

0:迭代收敛到所选容差范围内

1:达到收敛前的最大迭代次数

2:预条件矩阵是奇异的

3:算法达到停滞

4:算法因除零而无法继续，得到的相对残差为(a \*x-b) /范数(b)。

relres为获得的相对残差(A\*x-b) / norm (b)。

Iter是计算x的迭代。

resvec是一个向量，包含每次迭代的残差(包括范数(b - a x0))。执行length (resvec) - 1可以查看执行的迭代总数。

让我们考虑一个关于三对角矩阵的平凡问题

n = 20;

A = toeplitz (sparse ([1, 1], [1, 2], [2, 1] \* n ^ 2, 1, n)) + ...

toeplitz (sparse (1, 2, -1, 1, n) \* n / 2, ...

sparse (1, 2, 1, 1, n) \* n / 2);

b = A \* ones (n, 1);

restart = 5;

[M1, M2] = ilu (A); # in this tridiag case it corresponds to chol (A)'

M = M1 \* M2;

Afcn = @(x) A \* x;

Mfcn = @(x) M \ x;

M1fcn = @(x) M1 \ x;

M2fcn = @(x) M2 \ x;

例1:tfqmr的最简单用法

x = tfqmr (A, b, [], n)

例2:tfqmr和一个计算a \* x的函数

x = tfqmr (Afcn, b, [], n)

例3:具有预条件矩阵M的tfqmr

x = tfqmr (A, b, [], 1e-06, n, M)

例4:使用一个函数作为前置条件的tfqmr

x = tfqmr (Afcn, b, 1e-6, n, Mfcn)

例5:具有预条件矩阵M1和M2的tfqmr

x = tfqmr (A, b, [], 1e-6, n, M1, M2)

例6:使用函数作为前置条件的tfmqr

x = tfqmr (Afcn, b, 1e-6, n, M1fcn, M2fcn)

例7:tfqmr输入一个需要参数的函数

function y = Ap (A, x, z) # compute A^z \* x

y = x;

for i = 1:z

y = A \* y;

endfor

endfunction

Apfcn = @(x, string, p) Ap (A, x, string, p);

x = tfqmr (Apfcn, b, [], [], [], [], [], 2);

例8:显示tfqmr使用正确前置条件的显式示例

[M1, M2] = ilu (A + 0.3 \* eye (n)); # factorization of A perturbed

M = M1 \* M2;

## reference solution computed by tfqmr after one iteration

[x\_ref, fl] = tfqmr (A, b, [], 1, M)

## right preconditioning

[y, fl] = tfqmr (A / M, b, [], 1)

x = M \ y # compare x and x\_ref

参考:

杨志强，稀疏线性系统的迭代方法，第2版，2003年

参见:bigg, bigstab, cgs, gmres, pcg, qmr, pcr。

**19矢量化和更快的代码执行**

向量化是一种使用向量操作而不是逐元素循环操作的编程技术。除了经常生成更简洁的Octave代码外，向量化还允许在随后的实现中进行更好的优化。优化可以在Octave自己的Fortran、C或c++内部实现中进行，也可以在更低的级别上进行，具体取决于用于构建Octave的编译器和外部数字库。最终目标是尽可能利用硬件的矢量指令，或者在软件中执行其他优化。

向量化并不是Octave独有的概念，但它特别重要，因为Octave是一种面向矩阵的语言。在大多数情况下，矢量化的Octave代码将看到一个戏剧性的速度提升(10X-100X)。

本章讨论向量化和其他编写更快代码的技术。

**19.1基本矢量化**

首先，向量化的目标是编写避免循环并使用整个数组操作的代码。举一个简单的例子

for i = 1:n

for j = 1:m

c(i,j) = a(i,j) + b(i,j);

endfor

endfor

相比之下要简单得多

c = a + b;

这不仅更容易写;它在内部也更容易优化。Octave将此操作委托给底层实现，在其他优化中，该实现可能使用特殊的矢量硬件指令，或者甚至可以并行执行加法。一般来说，如果代码是向量化的，底层实现可以更自由地做出假设，以实现更快的执行速度。

这对于具有“廉价”主体的循环尤其重要。通常只对最内层循环进行矢量化就足以获得可接受的性能。一般的经验法则是，向量化体的“顺序”应该大于或等于封闭循环的“顺序”。

举个不那么简单的例子，而不是

for i = 1:n-1

a(i) = b(i+1) - b(i);

endfor

写作

a = b(2:n) - b(1:n-1);

这显示了一个重要的通用概念，即使用数组进行索引，而不是在索引变量上循环。参见索引表达式。也要大量使用布尔索引。如果需要测试一个条件，这个条件也可以写成布尔索引。例如，而不是

for i = 1:n

if (a(i) > 5)

a(i) -= 20

endif

endfor

写作

a(a>5) -= 20;

它利用了a > 5产生布尔索引的事实。

尽可能使用元素向量操作符以避免循环(操作符如。\*和。^)。参见算术运算符。

还要在这些元素操作符中利用广播来避免循环和不必要的中间内存分配。看到广播。

尽可能使用内置函数和库函数。内置函数和编译函数都非常快。即使使用m-file库函数，也很可能已经进行了优化，或者将在未来的版本中进行更多优化。

例如，甚至比

a = b(2:n) - b(1:n-1);

是

a = diff (b);

大多数Octave函数都是用矢量和数组参数编写的。如果您发现自己正在用一个非常简单的操作编写一个循环，那么很可能这样的函数已经存在。以下函数在向量化代码中经常出现:

* Index manipulation
* find
* sub2ind
* ind2sub
* sort
* unique
* lookup
* ifelse / merge
* Repetition
* repmat
* repelems
* Vectorized arithmetic
* sum
* prod
* cumsum
* cumprod
* sumsq
* diff
* dot
* cummax
* cummin
* Shape of higher dimensional arrays
* reshape
* resize
* permute
* squeeze
* deal

**19.2广播**

广播指的是Octave二进制运算符和函数在其矩阵或数组操作数或参数大小不同时的行为。从3.6.0版本开始，当使用二元运算符和函数时，Octave现在会自动广播向量、矩阵和数组。一般来说，较小的阵列在较大的阵列上“广播”，直到它们具有兼容的形状。规则是对应的数组维度必须是

1. 大于等于
2. 其中一个必须是1。

在所有维度相等的情况下，不发生广播，只进行普通的逐元素运算。对于更高维度的数组，如果维度数不相同，则缺失的尾随维度被视为1。当其中一个维度为1时，具有该单维度的数组沿着该维度复制，直到它匹配另一个数组的维度。例如，考虑

x = [1 2 3;

4 5 6;

7 8 9];

y = [10 20 30];

x + y

如果没有广播，x + y将是一个错误，因为维度不一致。然而，通过广播，就好像执行了以下操作:

x = [1 2 3

4 5 6

7 8 9];

y = [10 20 30

10 20 30

10 20 30];

x + y

⇒ 11 22 33

14 25 36

17 28 39

也就是说，沿着单例维度(行数)复制大小为[1 3]的较小数组，直到它为[3 3]。然而，实际的复制并没有发生。内部实现沿着必要的维度重用元素，以便在不复制内存的情况下获得所需的效果。

两个数组都可以相互传播，例如，一个向量的元素与自身的所有成对差异:

y - y'

⇒ 0 10 20

-10 0 10

-20 -10 0

在这里，大小为[1 3]和[31 1]的向量在进行普通矩阵减法之前都被广播到大小为[3 3]的矩阵中。

广播的一个特殊情况是，当被广播的数组的所有维数都为1时，即数组是一个标量。因此，例如，像x - 42和max (x, 2)这样的操作是广播的基本示例。

对于一个高维的例子，假设img是一个大小为[m n 3]的RGB图像，我们希望将每种颜色乘以一个不同的标量。下面的代码通过广播实现了这一点:

img .\*= permute ([0.8, 0.9, 1.2], [1, 3, 2]);

注意使用permute将[0.8,0.9,1.2]向量的维度与img匹配。

对于没有使用广播语义编写的函数，bsxfun可以用于强制它们进行广播。

: C = bsxfun (f, A, B)

对两个数组参数a和B逐个元素应用二进制函数f，必要时在任意一个输入参数中扩展单元素维度。

F是一个函数句柄、内联函数或包含要求值的函数名的字符串。函数f必须能够接受两个长度相等的列向量参数，或者一个列向量参数和一个标量。

A和B的维数必须相等或为单维。数组的单例维度将被扩展到与另一个数组相同的维度。

参见:arrayfun, cellfun。

只有当两个广播条件之一成立时，才会应用广播。然而，像往常一样，当两个维度不同时，广播也不适用，1也不适用:

x = [1 2 3

4 5 6];

y = [10 20

30 40];

x + y

这将产生关于不一致参数的错误。

除了常见的算术运算外，还有几个双参数函数也可以广播。广播的函数和操作符的完整列表

plus +

minus -

times .\*

rdivide ./

ldivide .\

power .^

lt <

le <=

eq ==

gt >

ge >=

ne != ~=

and &

or |

atan2

hypot

max

min

mod

rem

xor

+= -= .\*= ./= .\= .^= &= |=

这是广播力量的一个真实例子。Floyd-Warshall算法用于计算图中每对顶点之间的最短路径长度。一个n阶图邻接矩阵的简单实现可能是这样的:

for k = 1:n

for i = 1:n

for j = 1:n

dist(i,j) = min (dist(i,j), dist(i,k) + dist(k,j));

endfor

endfor

endfor

对最内层循环进行矢量化后，它可能看起来像这样:

for k = 1:n

for i = 1:n

dist(i,:) = min (dist(i,:), dist(i,k) + dist(k,:));

endfor

endfor

使用双向广播，它看起来像这样:

for k = 1:n

dist = min (dist, dist(:,k) + dist(k,:));

endfor

对于具有100个顶点的给定图，这三种技术的相对时间性能对于原始代码为7.3秒，对于单个矢量化代码为87毫秒，对于完全广播代码为1.3毫秒。对于一个有1000个顶点的图，向量化需要11.7秒，而广播只需要1.15秒。因此，为了提高性能，编写带有广播语义的代码是值得的。

但是，如果更简单的操作就足够了，请注意不要诉诸广播。对于矩阵a和b，考虑如下:

c = sum (permute (a, [1, 3, 2]) .\* permute (b, [3, 2, 1]), 3);

该操作在元素乘法期间广播两个具有排列维度的矩阵，以获得更大的3-D数组，然后沿着第三维对该数组求和。稍作思考就会证明，这个运算只不过是快得多的普通矩阵乘法，c = A \*b;

关于术语的说明:“广播”是由Python编程语言中的Numpy数值环境普及的术语。在其他编程语言和环境中，广播也可能被称为二进制单例扩展(MATLAB中的BSX，以及bsxfun函数名称的起源)、回收(R编程语言)、单指令多数据(SIMD)或复制。

**19.2.1广播和遗留代码**

新的广播语义几乎不会影响以前版本Octave中的代码。因此，所有在以前版本的Octave中工作的从MATLAB继承的代码应该仍然可以在Octave中工作而无需更改。唯一的例外是这样的代码

try

c = a.\*b;

catch

c = a.\*a;

end\_try\_catch

这可能依赖于不同大小的矩阵产生错误。因为这样的操作现在是有效的Octave语法，这将不再产生错误。相反，应该使用以下代码:

if (isequal (size (a), size (b)))

c = a .\* b;

else

c = a .\* a;

endif

**19.3功能应用**

作为一般规则，在编写函数时应该考虑到矩阵参数，并且应该以矢量化的方式考虑整个矩阵操作。有时，由于各种原因，以这种方式编写函数似乎很困难或不可能。对于这些情况，Octave提供了将函数应用于数组、单元格或结构体的每个元素的功能。

: B = arrayfun (fcn, A)

: B = arrayfun (fcn, A1, A2, …)

: [B1, B2, …] = arrayfun (fcn, A, …)

: B = arrayfun (…, "UniformOutput", val)

: B = arrayfun (…, "ErrorHandler", errfcn)

对数组的每个元素执行一个函数。

这对于不接受数组参数的函数很有用。如果函数不接受数组参数，最好直接调用函数。

第一个输入参数fcn可以是字符串、函数句柄、内联函数或匿名函数。输入参数A可以是逻辑数组、数字数组、字符串数组、结构数组或单元格数组。arrayfun将A的所有元素单独传递给函数fcn并收集结果。等效伪码为

cls = class (fcn (A(1));

B = zeros (size (A), cls);

for i = 1:numel (A)

B(i) = fcn (A(i))

endfor

命名函数也可以接受两个以上的输入参数，其中输入参数为第三个输入参数A2，第四个输入参数A2，…如果给定多个数组输入参数，则所有输入参数必须具有相同的大小。例如:

arrayfun (@atan2, [1, 0], [0, 1])

⇒ [ 1.57080 0.00000 ]

如果在另一个字符串输入参数“UniformOutput”之后的参数val被设置为true(默认值)，那么命名函数fcn必须返回一个元素，然后将其连接到返回值中，并且类型为矩阵。否则，如果该参数设置为false，则输出将连接到单元格数组中。例如:

arrayfun (@(x,y) x:y, "abc", "def", "UniformOutput", false)

⇒

{

[1,1] = abcd

[1,2] = bcde

[1,3] = cdef

}

如果给出了多个输出参数，则命名函数必须返回所期望的返回值的数量，例如:

[A, B, C] = arrayfun (@find, [10; 0], "UniformOutput", false)

⇒

A =

{

[1,1] = 1

[2,1] = [](0x0)

}

B =

{

[1,1] = 1

[2,1] = [](0x0)

}

C =

{

[1,1] = 10

[2,1] = [](0x0)

}

如果在另一个字符串输入参数“ErrorHandler”之后的参数errfcn是另一个字符串、函数句柄、内联函数或匿名函数，则errfcn定义一个函数，以便在fcn生成错误时调用。函数的定义必须是这样的形式

function […] = errfcn (s, …)

其中errfcn相对于fcn有一个额外的输入参数，由s给出。这是一个结构体，其元素“identifier”、“message”和“index”分别给出错误标识符、错误消息和导致错误的数组元素的索引。errfcn的输出参数的大小必须与fcn的输出参数的大小相同，否则将抛出一个真正的错误。例如:

function y = ferr (s, x), y = "MyString"; endfunction

arrayfun (@str2num, [1234],

"UniformOutput", false, "ErrorHandler", @ferr)

⇒

{

[1,1] = MyString

}

参见:spfun, cellfun, structfun。

: y = spfun (f, S)

计算S的非零元素f (S)

输入函数f只应用于输入矩阵S的非零元素，它通常是稀疏的。函数f可以作为字符串、函数句柄或内联函数传递。

输出y是一个稀疏矩阵，与输入S具有相同的稀疏结构。spfun保留了稀疏结构，这与当f(0) != 0时简单地将函数f应用于稀疏矩阵S不同。

例子

保持稀疏性的spfun与普通函数应用程序

S = pi \* speye (2,2)

S =

Compressed Column Sparse (rows = 2, cols = 2, nnz = 2 [50%])

(1, 1) -> 3.1416

(2, 2) -> 3.1416

y = spfun (@cos, S)

y =

Compressed Column Sparse (rows = 2, cols = 2, nnz = 2 [50%])

(1, 1) -> -1

(2, 2) -> -1

y = cos (S)

y =

Compressed Column Sparse (rows = 2, cols = 2, nnz = 4 [100%])

(1, 1) -> -1

(2, 1) -> 1

(1, 2) -> 1

(2, 2) -> -1

参见:arrayfun, cellfun, structfun。

: A = cellfun ("fcn", C)

: A = cellfun ("size", C, k)

: A = cellfun ("isclass", C, class)

: A = cellfun (@fcn, C)

: A = cellfun (fcn, C)

: A = cellfun (fcn, C1, C2, …)

: [A1, A2, …] = cellfun (…)

: A = cellfun (…, "ErrorHandler", errfcn)

: A = cellfun (…, "UniformOutput", val)

对单元格数组C的元素计算名为“fcn”的函数。

C中的元素被单独传递给命名函数。函数fcn可以是其中一个函数

isempty

对于空元素返回1。

islogical

对于逻辑元素返回1。

isnumeric

对于数字元素返回1。

isreal

对于实元素返回1。

length

返回单元格元素长度的向量。

ndims

返回每个元素的维数。

numel

prodofsize

返回每个单元格元素中包含的元素数。该数字是每个单元格元素处对象的尺寸的乘积。

size

返回沿第k维的大小。

isclass

类的元素返回1。

此外，cellfun还接受任意函数fcn，其形式为内联函数、函数句柄或函数名(以字符串形式)。该函数可以接受一个或多个参数，输入参数由C1、C2等给出。例如:

cellfun ("atan2", {1, 0}, {0, 1})

⇒ [ 1.57080 0.00000 ]

cellfun的输出参数数量与函数的输出参数数量匹配，并且可以大于1。当函数有多个输出时，它们将被收集到cellfun的输出参数中，如下所示:

function [a, b] = twoouts (x)

a = x;

b = x\*x;

endfunction

[aa, bb] = cellfun (@twoouts, {1, 2, 3})

⇒

aa =

1 2 3

bb =

1 4 9

请注意，默认情况下，输出参数是与输入参数大小相同的数组。单个(1x1)单元格的输入参数将自动扩展到其他参数的大小。

如果参数“UniformOutput”被设置为true(默认值)，那么函数必须返回标量，这些标量将被连接到返回数组中。如果“UniformOutput”为false，输出将被连接到一个单元格数组(或多个单元格数组)中。例如:

cellfun ("tolower", {"Foo", "Bar", "FooBar"},

"UniformOutput", false)

⇒ {"foo", "bar", "foobar"}

给定参数“ErrorHandler”，然后errfcn定义一个函数，以便在fcn生成错误时调用。函数的形式是

function […] = errfcn (s, …)

其中errfcn相对于fcn有一个额外的输入参数，由s给出。这是一个包含元素“identifier”，“message”和“index”的结构，分别给出了错误标识符，错误消息和导致错误的元素的输入参数的索引。例如:

function y = foo (s, x), y = NaN; endfunction

cellfun ("factorial", {-1,2}, "ErrorHandler", @foo)

⇒ [NaN 2]

明智地使用手机。cellfun函数是避免循环的有用工具。它经常与匿名函数句柄一起使用;但是，调用匿名函数所涉及的开销与m-file函数的开销相当。将句柄传递给内置函数更快，因为解释器不参与内部循环。例如:

C = {…}

v = cellfun (@(x) det (x), C); # compute determinants

v = cellfun (@det, C); # 40% faster

参见:arrayfun, structfun, spfun。

: A = structfun (fcn, S)

: A = structfun (…, "ErrorHandler", errfcn)

: A = structfun (…, "UniformOutput", val)

: [A, B, …] = structfun (…)

在结构S的字段上计算名为name的函数。S的字段分别传递给函数fcn。

Structfun以内联函数、函数句柄或函数名(以字符串形式)的形式接受任意函数FCN。在字符串参数的情况下，函数必须接受一个名为x的参数，并且它必须返回一个字符串值。如果函数返回多个参数，它们将作为单独的输出变量返回。

如果参数“UniformOutput”被设置为true(默认值)，那么该函数必须返回一个元素，该元素将被连接到返回值中。如果“UniformOutput”为false，输出将被放入与输入结构具有相同字段名的结构中。

s.name1 = "John Smith";

s.name2 = "Jill Jones";

structfun (@(x) regexp (x, '(\w+)$', "matches"){1}, s,

"UniformOutput", false)

⇒ scalar structure containing the fields:

name1 = Smith

name2 = Jones

给定参数“ErrorHandler”，errfcn定义了一个函数，以便在fcn生成错误时调用。函数的形式是

function […] = errfcn (se, …)

其中errfcn相对于fcn有一个额外的输入参数，由se给出。这是一个包含元素“identifier”、“message”和“index”的结构，分别给出错误标识符、错误消息和导致错误的元素的输入参数的索引。有关如何使用错误处理程序的示例，请参见cellfun。

参见:cellfun, arrayfun, spfun。

与之前的建议一致，尽可能使用Octave内置函数以获得最佳性能。这个建议特别适用于上面的四个函数。例如，当将两个数组一个元素一个元素地添加在一起时，可以使用内置加法函数@plus的句柄或定义一个匿名函数@(x,y) x + y。但是，匿名函数比第一种方法慢60%。请参阅操作符重载，以获取可用于代替匿名函数的基本函数列表。

**19.4积累**

在执行计算时，只要有可能根据索引对数组元素进行分类，累积函数就会很有用。

: A = accumarray (subs, vals)

: A = accumarray (subs, vals, sz)

: A = accumarray (subs, vals, sz, fcn)

: A = accumarray (subs, vals, sz, fcn, fillval)

: A = accumarray (subs, vals, sz, fcn, fillval, issparse)

通过将向量的元素累加到其下标定义的位置来创建数组。

下标由矩阵下标的行定义，值由值定义。每一行子元素对应于vals中的一个值。如果vals是一个标量，它将用于每一行子元素。如果subs是向量的单元数组，则所有向量必须具有相同的长度，并且第k个向量中的下标必须对应于结果的第k维。

矩阵的大小将由下标本身决定。然而，如果定义了sz，它决定了矩阵的大小。sz的长度必须对应于subs中的列数。一个例外是如果subs只有一列，在这种情况下，sz可以是一个向量的维数，subs的下标被作为它的下标。

accumarray的默认操作是对具有相同下标的元素求和。这种行为可以通过定义fcn函数来修改。这应该是一个接受列向量并返回标量的函数或函数句柄。函数的结果不应该依赖于下标的顺序。

返回数组中没有关联下标的元素被设置为零。将fillval定义为其他值允许定义这些值。但是，对于某些fcn值，这种行为会发生变化。如果fcn为@min(分别为@max)，则如果vals为整型，则结果将填充最小(分别为最大)整数;如果vals为逻辑类型，则为逻辑假(分别为逻辑真);如果fillval为零且所有值均为非正(分别为非负)，则为零;否则为NaN。

默认情况下，accumarray返回一个完整的矩阵。如果issparse在逻辑上为真，则返回一个稀疏矩阵。

下面的accumarray示例构造了一个频率表，在第一列中计算第二列中每个数字出现的次数，取自向量x。注意，使用unique为x的所有重复元素分配相同的索引(参见unique)。

x = [91, 92, 90, 92, 90, 89, 91, 89, 90, 100, 100, 100];

[u, ~, j] = unique (x);

[accumarray(j', 1), u']

⇒ 2 89

3 90

2 91

2 92

3 100

另一个例子，结果是一个多维三维数组，默认值(零)出现在输出中:

accumarray ([1, 1, 1;

2, 1, 2;

2, 3, 2;

2, 1, 2;

2, 3, 2], 101:105)

⇒ ans(:,:,1) = [101, 0, 0; 0, 0, 0]

⇒ ans(:,:,2) = [0, 0, 0; 206, 0, 208]

稀疏选项可以用作稀疏构造函数的替代方法(参见sparse)。因此

sparse (i, j, sv)

可以写与伏数组为

accumarray ([i, j], sv', [], [], 0, true)

对于重复索引，sparse会添加相应的值。要取最小值，使用min作为累加器函数:

accumarray ([i, j], sv', [], @min, 0, true)

对于非稀疏情况，一般情况下，伏数组的复杂度一般为O(M+N)，其中N为下标个数，M为最大下标(在多维情况下线性化)。如果fcn是@sum(默认)，@max， @min或@(x) {x}中的一个，则使用优化的代码路径。请注意，对于一般的简化函数，解释器开销可能起主要作用，并且执行多个accumarray调用并以矢量化的方式计算结果可能更有效。

参见:堆积的，独特的，稀疏的。

: A = accumdim (subs, vals)

: A = accumdim (subs, vals, dim)

: A = accumdim (subs, vals, dim, n)

: A = accumdim (subs, vals, dim, n, fcn)

: A = accumdim (subs, vals, dim, n, fcn, fillval)

通过沿着指定维度将数组的切片累积到由其下标定义的位置来创建数组。

下标由索引向量下标定义。维度由dim指定。如果未给出，则默认为第一个非单元素维度。子节点的长度必须等于size (vals, dim)。

结果矩阵在工作维中的范围将由下标本身决定。然而，如果n是定义的，它决定了这个范围。

accumdim的默认操作是对具有相同下标的子数组求和。这种行为可以通过定义fcn函数来修改。这应该是一个函数或函数句柄，它接受一个数组和一个维度，并沿着这个维度减少数组。作为一个特殊的例外，内置的min和max函数可以直接使用，并且在调用它们时使用中间的空参数。

返回的数组中没有关联下标的切片被设置为零。将fillval定义为其他值允许定义这些值。

使用accumdim的一个例子是:

accumdim ([1, 2, 1, 2, 1], [ 7, -10, 4;

-5, -12, 8;

-12, 2, 8;

-10, 9, -3;

-5, -3, -13])

⇒ [-10,-11,-1;-15,-3,5]

参见:accumarray。

**19.5记忆有关**

记忆是一种缓存缓慢函数调用结果的技术，并在使用相同的输入再次调用函数时返回缓存的值，而不是重新计算它。如果相同的输入以一种已知的、可预测的方式反复发生，那么用查找表替换函数调用是很常见的。记忆的核心是这种实践的扩展，在这种实践中，查找表甚至在运行时为以前未见过的新参数进行扩展。基本的理论背景可以在维基百科或任何本科水平的计算机科学教科书中找到。

Octave的memoize函数为任何用户函数或Octave函数(包括编译函数)提供了插入式记忆功能。

: mem\_fcn\_handle = memoize (fcn\_handle)

创建函数fcn\_handle的记忆版本mem\_fcn\_handle。

每次调用内存版本mem\_fcn\_handle都会根据内部维护的表检查输入，如果输入以前发生过，则从表本身返回函数调用的结果，而不是再次计算整个函数。这加快了使用相同输入多次调用的函数的执行速度。

例如，这里我们取一个慢速的用户编写函数slow\_fcn，并将其记忆到一个新的句柄cyc。两个版本的第一次执行所用的时间相同，但随后执行的记忆版本返回先前计算的值，从而将2.4秒的运行时减少到仅2.4毫秒。最后的检查验证从两个版本返回的结果是否相同。

>> tic; p = slow\_fcn (5040); toc

Elapsed time is 2.41244 seconds.

>> tic; p = slow\_fcn (5040); toc

Elapsed time is 2.41542 seconds.

>> cyc = memoize (@slow\_fcn);

>> tic; r = cyc (5040); toc

Elapsed time is 2.42609 seconds.

>> tic; r = cyc (5040); toc

Elapsed time is 0.00236511 seconds.

>> all (p == r)

ans = 1

参见:clearAllMemoizedCaches。

要记忆函数z = foo(x, y)，请使用以下通用模式:

foo2 = memoize (@(x, y) foo(x, y));

z = foo2 (x, y);

在上面的例子中，第一行创建了函数foo的记忆版本foo2。对于只有简单换行的简单函数，这行也可以缩短为:

foo2 = memoize (@foo);

第二行z = foo2 (x, y);调用调用版本foo2而不是原始函数，如果输入之前发生过，则允许memoize拦截调用并将其替换为从表中查找的值，而不是再次计算原始函数。

注意，这不会加速对函数的第一次调用，而只会加速随后的调用。

请注意，由于memoize为每个函数创建和管理查找表所带来的开销，因此该技术仅对执行时间至少为几秒钟的函数有用。这样的函数可以被表查找取代，只需要一毫秒左右的时间，但是如果原始函数本身只需要几毫秒，那么记忆它不会加快它的速度。

递归函数也可以记忆，使用如下模式:

function z = foo (x, y)

persistent foo2 = memoize (@foo);

foo2.CacheSize = 1e6;

## Call the memoized version when recursing

z = foo2 (x, y);

endfunction

如果预期会有大量的函数调用，比如从递归函数内部调用，可以选择性地增加CacheSize。如果超过CacheSize，则会调整记忆表的大小，从而导致速度变慢。因此，增加CacheSize就像预分配一样可以加快执行速度。

clearAllMemoizedCaches函数在不再需要记忆表时清除它们。

: clearAllMemoizedCaches ()

清除所有记忆缓存。

记忆维护内部表，表中记录了哪些函数被调用了哪些输入。这个函数清除这些表以释放内存，或者重新开始。

参见:memoize。

**19.6杂项技术**

这里有一些提高Octave程序执行速度的其他方法。

避免多次计算代价高昂的中间结果。Octave目前不能消除常见的子表达式。此外，某些内部计算结果将为变量缓存。例如，如果一个矩阵变量被多次用作索引，那么检查索引(以及内部到整数的转换)只会执行一次。

注意惰性拷贝(写时拷贝)。创建对象的副本时，不会立即复制数据，而是共享数据。实际的复制被推迟到需要修改复制的数据时。例如:

a = zeros (1000); # create a 1000x1000 matrix

b = a; # no copying done here

b(1) = 1; # copying done here

延迟复制适用于整个Octave对象，例如矩阵、单元格、结构体，以及单个单元格或结构体元素(不是数组元素)。

此外，当Octave可以确定索引部分在内存中是连续的时，索引表达式也会使用延迟复制。例如:

A = 0 (1000);#创建一个1000x1000矩阵b = a(:，10:100);b = a(10:100，:);# copy done here

这适用于数组(矩阵)、单元格数组和使用'()'索引的结构体。在某些情况下，生成逗号分隔列表的索引表达式也可以从浅复制中获益。特别地，当a是一个结构数组时，像{a。x},{(:, 2)。X}将使用延迟复制，这样数据就可以在struct数组和cell数组之间共享。

大多数索引表达式的寿命不会比它们的父对象长。然而，在极少数情况下，延迟复制的片比它的父片寿命更长，在这种情况下，它成为孤儿，仍然占用不必要的内存。为了提供在大多数实际情况下都有效的补救措施，Octave在某些情况下检查孤立的惰性切片，当值存储在“永久”位置时，例如命名变量或单元格或结构元素，并可能节省它们。例如:

a = zeros (1000); # create a 1000x1000 matrix

b = a(:,10:100); # lazy slice

a = []; # the original "a" array is still allocated

c{1} = b; # b is reallocated at this point

避免深度递归。对m-file函数的函数调用会带来相当大的开销，因此将递归重写为循环通常会有所帮助。另外，请注意递归的最大级别是有限的。

避免不必要地调整矩阵大小。在通过一系列计算构建单个结果矩阵时，首先设置结果矩阵的大小，然后向其中插入值。写

result = zeros (big\_n, big\_m)

for i = over:and\_over

ridx = …

cidx = …

result(ridx, cidx) = new\_value ();

endfor

代替

result = [];

for i = ever:and\_ever

result = [ result, new\_value() ];

endfor

有时不能提前计算项目的数量，需要进行类似堆栈的操作。当数组末尾重复插入或删除元素时，Octave将其检测为堆栈使用，并尝试使用更智能的内存管理策略，将数组预先分配到更大的块中。这种策略也适用于单元格和结构体数组。

a = [];

while (condition)

…

a(end+1) = value; # "push" operation

…

a(end) = []; # "pop" operation

…

endwhile

避免过度调用eval或feval。解析输入或在符号表中查找函数名是相对昂贵的操作。

如果您只是将eval用作异常处理机制，而不是因为需要执行任意文本，那么请使用try语句。参见try语句。

在适当的时候使用ignore\_function\_time\_stamp。如果您正在调用许多函数，并且在运行期间不需要更改它们中的任何一个，请将变量ignore\_function\_time\_stamp设置为“all”。这将阻止Octave检查函数文件的时间戳，以查看它是否在程序运行时被更新。

**19.7的例子**

以下是Octave实际用户提出的矢量化问题的示例及其解决方案。

对于向量a，下面的循环

n = length (A) - 1;

B = zeros (n, 2);

for i = 1:n

## this will be two columns, the first is the difference and

## the second the mean of the two elements used for the diff.

B(i,:) = [A(i+1)-A(i), (A(i+1) + A(i))/2];

endfor

可以转换成以下一行代码:

B = [diff(A)(:), 0.5\*(A(1:end-1)+A(2:end))(:)]

注意使用冒号索引将中间结果平铺成列向量。这是一个常见的矢量化技巧。

**20非线性方程**

**20.1解决者**

八度可以解出一组非线性方程的形式

F (x) = 0

使用函数fsolve，该函数基于MINPACK子例程hybrid。这是一种迭代技术，因此必须提供一个起点。这也会导致即使存在解也不能保证收敛。

: x = fsolve (fcn, x0)

: x = fsolve (fcn, x0, options)

: [x, fval] = fsolve (…)

: [x, fval, info] = fsolve (…)

: [x, fval, info, output] = fsolve (…)

: [x, fval, info, output, fjac] = fsolve (…)

解一个由函数fcn定义的非线性方程组。

FCN是一个函数句柄、内联函数或包含要求值的函数名的字符串。FCN应该接受定义未知变量的向量(数组)，并返回方程左侧的向量。右边被定义为0。换句话说，这个函数试图确定一个向量x，使得fcn (x)(近似)全为零。

X0是解的初始猜测。x0的形状在对fcn的所有调用中都被保留，但除此之外被视为列向量。

Options是一个指定控制算法的附加参数的结构。目前，fsolve可以识别这些选项:“AutoScaling”、“ComplexEqn”、“FinDiffType”、“FunValCheck”、“Jacobian”、“MaxFunEvals”、“MaxIter”、“OutputFcn”、“TolFun”、“TolX”、“TypicalX”和“Updating”。

如果“AutoScaling”是“on”，则变量将根据(估计的)雅可比矩阵的列规范自动缩放。因此，“TolFun”变得独立于规模。默认情况下，此选项为“关闭”，因为它有时可能会提供意想不到的(尽管在数学上是正确的)结果。

如果“ComplexEqn”为“on”，则fsolve将尝试求解复变量中的复方程，假设该方程具有复导数(即全纯)。如果这不是您想要的，那么您应该解压缩系统的实部和虚部，以获得一个真实的系统。

如果“Jacobian”为“on”，则它指定fcn——当使用2个输出参数调用时——也会返回请求点右侧的雅可比矩阵。

“MaxFunEvals”禁止停止优化之前函数求值的最大次数。默认值是100 \* number\_of\_variables，即100 \* length (x0)。取值必须为正整数。

如果“更新”为“on”，则该函数将尝试使用Broyden更新来更新雅可比矩阵，以减少雅可比矩阵的计算次数。如果您的用户函数总是计算雅可比矩阵(不管输出参数的数量)，那么这个选项没有任何优势，应该禁用。

“TolX”指定未知变量的终止容差，而“TolFun”是方程的容差。“TolX”和“TolFun”的默认值都是1e-6。

有关其他选项的说明，请参见optimset。使用options = optimset ("fsolve")为fsolve初始化一个默认值的选项结构。

第一个输出x是解，而第二个输出fval包含函数fcn在x处的值(理想情况下是全0的向量)。

第三个输出info报告算法是否成功，可以取以下值之一:

1

收敛到一个解点。相对残差小于TolFun指定的值。

2

最后相对步长小于TolX。

3

残留的最后相对减少量小于TolFun。

0

超出了迭代限制(MaxIter或MaxFunEvals)。

-1

被OutputFcn停止。

-2

雅可比矩阵变得太小，搜索陷入了停滞。

3

信任域半径过小。处理步骤

输出是一个包含有关fsolve算法的运行时信息的结构。结构中的字段有:

iterations

通过循环的迭代次数。

successful

成功迭代的次数。

funcCount

函数求值的次数。

最终输出fjac包含雅可比矩阵在x处的值。

注意:如果你只有一个单一变量的非线性方程，使用f0通常是一个更好的主意。

关于用户提供的雅可比矩阵的注意事项:作为算法的固有属性，对于残差向量已知的解向量总是要求雅可比矩阵，并且它是最后接受的成功步骤。通常这将是最后两个呼叫之一，但并不总是这样。如果在雅可比矩阵计算中重用残差计算的中间结果节省的量很大，那么最好的策略是使用OutputFcn:在对一个矢量求残差后，如果使用该矢量调用OutputFcn，则应该保存中间结果以供将来的雅可比矩阵求值，并且应该保留中间结果，直到请求进行雅可比矩阵求值，或者直到使用不同的矢量调用OutputFcn，在这种情况下，它们应该被丢弃，以支持这个最新的矢量。以下是如何实现这一目标的一个简短示例:

function [fval, fjac] = user\_fcn (x, optimvalues, state)

persistent sav = [], sav0 = [];

if (nargin == 1)

## evaluation call

if (nargout == 1)

sav0.x = x; # mark saved vector

## calculate fval, save results to sav0.

elseif (nargout == 2)

## calculate fjac using sav.

endif

else

## outputfcn call.

if (all (x == sav0.x))

sav = sav0;

endif

## maybe output iteration status, etc.

endif

endfunction

## …

fsolve (@user\_fcn, x0, optimset ("OutputFcn", @user\_fcn, …))

参见:fzero, optimset。

下面是一个完整的示例。来解这组方程

-2x^2 + 3xy + 4 sin(y) = 6

3x^2 - 2xy^2 + 3 cos(x) = -4

首先需要编写一个函数来计算给定函数的值。例如:

function y = f (x)

y = zeros (2, 1);

y(1) = -2\*x(1)^2 + 3\*x(1)\*x(2) + 4\*sin(x(2)) - 6;

y(2) = 3\*x(1)^2 - 2\*x(1)\*x(2)^2 + 3\*cos(x(1)) + 4;

endfunction

然后，在给定初始条件下调用fsolve来求方程组的根。例如，给定上面定义的函数f，

[x, fval, info] = fsolve (@f, [1; 2])

解决方案的结果

x =

0.57983

2.54621

fval =

-5.7184e-10

5.5460e-10

info = 1

info = 1表示解已经收敛。

当没有提供雅可比矩阵时(如上面的例子)，它是数值近似的。这需要更多的函数求值，因此效率较低。在上面的例子中我们可以解析地计算雅可比矩阵

function [y, jac] = f (x)

y = zeros (2, 1);

y(1) = -2\*x(1)^2 + 3\*x(1)\*x(2) + 4\*sin(x(2)) - 6;

y(2) = 3\*x(1)^2 - 2\*x(1)\*x(2)^2 + 3\*cos(x(1)) + 4;

if (nargout == 2)

jac = zeros (2, 2);

jac(1,1) = 3\*x(2) - 4\*x(1);

jac(1,2) = 4\*cos(x(2)) + 3\*x(1);

jac(2,1) = -2\*x(2)^2 - 3\*sin(x(1)) + 6\*x(1);

jac(2,2) = -4\*x(1)\*x(2);

endif

endfunction

雅可比矩阵可以通过下面的调用来求解:

[x, fval, info] = fsolve (@f, [1; 2], optimset ("jacobian", "on"));

得到和之前一样的解。

: x = fzero (fcn, x0)

: x = fzero (fcn, x0, options)

: [x, fval] = fzero (…)

: [x, fval, info] = fzero (…)

: [x, fval, info, output] = fzero (…)

求一个单变量函数的零点。

FCN是一个函数句柄、内联函数或包含要求值的函数名的字符串。

X0应该是一个双元素向量，指定两个点，它们的括号是0。换句话说，函数的符号在x0(1)和x0(2)之间一定有变化。更精确地说，下面的条件必须成立

sign (fcn(x0(1))) \* sign (fcn(x0(2))) <= 0

如果x0是单个标量，则探测几个附近和远处的值，以尝试获得有效的括号。如果不成功，则函数失败。

Options是一个指定附加选项的结构。目前，fzero识别这些选项:“Display”，“FunValCheck”，“MaxFunEvals”，“MaxIter”，“OutputFcn”和“TolX”。

“MaxFunEvals”禁止在停止搜索之前计算函数的最大次数。默认值为Inf。取值必须为正整数。

“MaxIter”禁止搜索停止前算法迭代的最大次数。默认值为Inf。取值必须为正整数。

“TolX”指定解决方案x的终止容忍度，默认值为eps。

有关其他选项的说明，请参见optimset。使用options = optimset ("fzero")初始化一个具有fzero默认值的选项结构。

在退出时，函数返回近似零点x和函数在x处的值fval。

第三个输出info报告算法是否成功，可以取以下值之一:

该算法收敛为一个解。

0已达到迭代或函数求值的最大次数。

-1算法已被用户OutputFcn终止。

-5算法可能已经收敛到一个奇点。

输出是一个包含fzero算法运行时信息的结构。结构中的字段有:

iterations循环迭代次数。

funcCount函数计算次数。

字符串“平分，插值”。

一个两元素的矢量，沿着x轴的最后一个零的括号。

一个两元素的矢量，最后的零沿y轴加括号。

参见:optimset, fsolve。

**20.2解**

通常，找到函数的最小值比仅仅找到与x轴相交的零点更有用。fminbind是为更简单但很常见的单变量函数设计的，其中要搜索的区间是有界的。对于可能有许多变量的函数的无界最小化，使用fminunc或fminsearch。这两个函数使用不同的内部算法，并且需要对目标函数有一定的了解。对于可微分的函数，用fminunc比较合适。对于不连续的函数，或者梯度搜索会失败的函数，使用fminsearch。关于约束函数存在的最小化，请参见优化。注意，可以通过简单地反转目标函数(Fto\_max = -Fto\_min)来搜索最大值。

: x = fminbnd (fcn, a, b)

: x = fminbnd (fcn, a, b, options)

: [x, fval, info, output] = fminbnd (…)

求单变量函数的最小值点。

FCN是一个函数句柄、内联函数或包含要求值的函数名的字符串。

起始间隔由a(左边界)和b(右边界)指定。端点必须是有限的。

Options是一个指定控制算法的附加参数的结构。目前，fminbind可以识别这些选项:“Display”，“FunValCheck”，“MaxFunEvals”，“MaxIter”，“OutputFcn”，“TolX”。

“MaxFunEvals”禁止停止优化之前函数求值的最大次数。缺省值是500。取值必须为正整数。

“MaxIter”禁止在优化停止之前算法迭代的最大次数。缺省值是500。取值必须为正整数。

“TolX”指定解决方案x的终止容差，默认为1e-4。

有关其他选项的说明，请参见optimset。使用options = optimset (" fminbind ")初始化一个带有fminbind默认值的选项结构。

在退出时，函数返回近似最小点x和函数对x求值的fval。

第三个输出info报告算法是否成功，可以取以下值之一:

* 1 该算法收敛为一个解。
* 0 超过了迭代限制(MaxIter或MaxFunEvals)。
* -1算法被用户OutputFcn终止。

编程注意:搜索最小值被限制在a和b的有限区间内。如果你只有一个初始点开始搜索，那么你需要使用无约束最小化算法，如fminunc或fminsearch。fminbind内部使用黄金分割搜索策略。

参见:fzero, fminunc, fminsearch, optimset。

: x = fminunc (fcn, x0)

: x = fminunc (fcn, x0, options)

: [x, fval] = fminunc (fcn, …)

: [x, fval, info] = fminunc (fcn, …)

: [x, fval, info, output] = fminunc (fcn, …)

: [x, fval, info, output, grad] = fminunc (fcn, …)

: [x, fval, info, output, grad, hess] = fminunc (fcn, …)

求解一个由函数fcn定义的无约束优化问题。

Fminunc试图确定一个向量x，使得FCN (x)是一个局部最小值。

FCN是一个函数句柄、内联函数或包含要求值的函数名的字符串。FCN应该接受定义未知变量的向量(数组)，并返回目标函数值，可选地带有梯度。

X0决定了开始的猜测。x0的形状在对fcn的所有调用中都被保留，但除此之外被视为列向量。

Options是一个指定控制算法的附加参数的结构。目前，fminunc识别这些选项:“AutoScaling”，“FinDiffType”，“FunValCheck”，“GradObj”，“MaxFunEvals”，“MaxIter”，“OutputFcn”，“TolFun”，“TolX”，“TypicalX”。

如果“AutoScaling”是“on”，则变量将根据(估计的)雅可比矩阵的列规范自动缩放。因此，“TolFun”变得独立于规模。默认情况下，此选项为“关闭”，因为它有时可能会提供意想不到的(尽管在数学上是正确的)结果。

如果“GradObj”处于“on”状态，则它指定fcn——当使用两个输出参数调用时——也返回所请求点的偏一阶导数的雅可比矩阵。

“MaxFunEvals”禁止停止优化之前函数求值的最大次数。默认值是100 \* number\_of\_variables，即100 \* length (x0)。取值必须为正整数。

“MaxIter”禁止在优化停止之前算法迭代的最大次数。缺省值是400。取值必须为正整数。

“TolX”指定未知变量x的终止容差，而“TolFun”是目标函数值fval的容差。这两个选项的默认值都是1e-6。

有关其他选项的说明，请参见optimset。

返回时，x是最小值的位置，fval包含目标函数在x处的值。

Info可能是以下值之一:

1

收敛到一个解点。相对梯度误差小于TolFun指定的。

2

最后相对步长小于TolX。

3

函数值的最后相对变化小于TolFun。

0

超出迭代限制—算法迭代的最大次数MaxIter或函数求值的最大次数MaxFunEvals。

-1

由OutputFcn终止的算法。

3

信任域半径过小。处理步骤

可选地，fminunc可以返回一个具有收敛统计(输出)、解x处的输出梯度(grad)和解x处的近似Hessian (hess)的结构。

应用注意:如果目标函数是一个单变量的非线性方程，那么使用fminbind通常是更好的选择。

fminunc使用的算法是一种依赖于目标函数可微的梯度搜索。如果函数有不连续点，最好使用无导数的算法，如fminsearch。

参见:fminbind, fminsearch, optimset。

: x = fminsearch (fcn, x0)

: x = fminsearch (fcn, x0, options)

: x = fminsearch (problem)

: [x, fval, exitflag, output] = fminsearch (…)

求一个使多变量函数fcn最小的x值。

FCN是一个函数句柄、内联函数或包含要求值的函数名的字符串。

搜索从点x0开始，并使用Nelder & Mead Simplex算法(一种无导数方法)进行迭代。该算法更适合于具有不连续的函数或基于梯度的搜索(如fminunc)失败的函数。

搜索的选项在使用函数optimset的参数选项中提供。目前，fminsearch接受选项:“Display”，“FunValCheck”，“MaxFunEvals”，“MaxIter”，“OutputFcn”，“TolFun”，“TolX”。

“MaxFunEvals”禁止停止优化之前函数求值的最大次数。默认值是200 \* number\_of\_variables，即200 \* length (x0)。取值必须为正整数。

“MaxIter”禁止在优化停止之前算法迭代的最大次数。默认值是200 \* number\_of\_variables，即200 \* length (x0)。取值必须为正整数。

有关其他选项的说明，请参见optimset。使用options = optimset ("fminsearch")为fminsearch初始化一个默认值的选项结构。

Fminsearch也可以使用单个结构参数调用，参数包含以下字段:

objective

目标函数

x0

最初的起点

solver

必须设置为“fminsearch”。

options

从optimset或空矩阵返回的结构，用于指示应该使用默认值。

字段options是可选的。所有其他的都是必需的。

退出时，函数返回最小值点x和最小值处的函数值fval。

第三个输出exitflag报告算法是否成功，可以取以下值之一:

1

如果算法收敛(单纯形的大小小于TolX且迭代之间的函数值步长小于TolFun)。

0

如果超过了最大迭代次数或最大函数求值次数。

－1

如果迭代被“OutputFcn”停止。

第四个输出是包含算法运行时的结构输出。结构中的字段是funcCount，包含对fcn的函数调用次数，iterations包含迭代步骤的次数，algorithm包含搜索算法的名称(总是:"Nelder-Mead simplex direct search")， message包含退出消息。

例子:

fminsearch (@(x) (x(1)-5).^2+(x(2)-8).^4, [0;0])

注意:如果您需要找到单个变量函数的最小值，最好使用fminbind。

参见:fminbind, fminunc, optimset。

函数humps是测试零点和极值查找函数的有用函数。

: y = humps (x)

: [x, y] = humps (x)

用多个极小值、最大值和零点交叉点计算一个函数。

输出y是有理函数的求值:

1200\*x^4 - 2880\*x^3 + 2036\*x^2 - 348\*x - 88

y = - ---------------------------------------------

200\*x^4 - 480\*x^3 + 406\*x^2 - 138\*x + 17

X可以是标量、向量或数组。如果省略x，则使用默认范围[0:05 . 1]。

当使用两个输出参数[x, y]调用时，x将包含输入值，y将包含来自驼峰的输出。

编程注意事项:驼峰在x = 0.300和0.893附近有两个局部最大值，在x = 0.637附近有一个局部最小值，在x = -0.132和1.300附近有零。驼峰是一个有用的函数，用于测试找到零点或局部极小值和最大值的算法。

尝试演示humps以查看humps函数的图。

参见:fzero, fminbind, fminunc, fminsearch。

**21对角矩阵和置换矩阵**

**21.1创建和操作对角/排列矩阵**

对角矩阵被定义为在主对角外没有元素的矩阵;也就是说，如果i != j，则D(i,j) == 0。大多数情况下，考虑方形对角线矩阵;然而，这个定义同样可以应用于非方阵，在这种情况下，我们通常说的是矩形对角矩阵。

排列矩阵定义为在每一行和每一列中具有一个等于1的单个元素的方阵;其他元素都是0。也就是说，存在一个置换(向量)p，使得当j == p(i,j)时p(i,j) == 1，否则p(i,j) == 0。

Octave提供了实和复矩形对角矩阵的特殊处理，以及置换矩阵。它们作为特殊对象存储，使用高效的存储和算法，便于在Octave语言中编写可读和高效的矩阵代数表达式。可以通过使用函数optimize\_diagonal\_matrix和optimize\_permutation\_matrix禁用这种特殊处理。

: val = optimize\_diagonal\_matrix ()

: old\_val = optimize\_diagonal\_matrix (new\_val)

: old\_val = optimize\_diagonal\_matrix (new\_val, "local")

查询或设置是否使用特殊的空间高效格式来存储对角矩阵。

默认值为true。如果此选项设置为false, Octave将对角矩阵存储为完整矩阵。

当从带有“local”选项的函数内部调用时，将在本地更改函数及其调用的任何子例程的设置。退出该功能时，恢复原来的设置。

参见:optimize\_range, optimize\_permutation\_matrix。

: val = optimize\_permutation\_matrix ()

: old\_val = optimize\_permutation\_matrix (new\_val)

: old\_val = optimize\_permutation\_matrix (new\_val, "local")

查询或设置是否使用特殊的空间高效格式来存储置换矩阵。

默认值为true。如果此选项设置为false, Octave将把排列矩阵存储为完整矩阵。

当从带有“local”选项的函数内部调用时，将在本地更改函数及其调用的任何子例程的设置。退出该功能时，恢复原来的设置。

参见:optimize\_range, optimize\_diagonal\_matrix。

如下面的代码所示，节省的空间非常可观。

x = diag (rand (10, 1));

xf = full (x);

sizeof (x)

⇒ 80

sizeof (xf)

⇒ 800

**21.1.1创建对角矩阵**

创建对角矩阵最常见和最简单的方法是使用内置函数diag。表达式diag (v)，以v为向量，将创建一个方形对角矩阵，其元素位于v的元素所给出的主对角线上，并且大小等于v的长度。diag (v, m, n)可用于构造一个矩形对角矩阵。这些表达式的结果将是一个特殊的对角矩阵对象，而不是一般的矩阵对象。

单位元素的对角矩阵可以用眼睛创建。其他一些内置函数也可以返回对角矩阵。例子包括balance或invv。

例子:

diag (1:4)

⇒

Diagonal Matrix

1 0 0 0

0 2 0 0

0 0 3 0

0 0 0 4

diag (1:3,5,3)

⇒

Diagonal Matrix

1 0 0

0 2 0

0 0 3

0 0 0

0 0 0

**21.1.2创建置换矩阵**

对于创建排列矩阵，Octave并没有引入新的函数，而是覆盖了现有的语法:可以通过按排列向量索引单位矩阵来方便地创建排列矩阵。也就是说，如果q是长度为n的排列向量，表达式

P = eye (n) (:, q);

将创建一个排列矩阵——一个特殊的矩阵对象。

eye (n) (q, :)

也将工作(并创建一个行排列矩阵)，以及

eye (n) (q1, q2).

例如：

eye (4) ([1,3,2,4],:)

⇒

Permutation Matrix

1 0 0 0

0 0 1 0

0 1 0 0

0 0 0 1

eye (4) (:,[1,3,2,4])

⇒

Permutation Matrix

1 0 0 0

0 0 1 0

0 1 0 0

0 0 0 1

在数学上，单位矩阵既是对角矩阵又是置换矩阵。在Octave中，eye (n)返回一个对角矩阵，因为一个矩阵只能有一个类。您可以通过单位置换对该对角矩阵进行索引，将其转换为置换矩阵，如下所示。这是单位矩阵的一个特殊性质;索引其他对角矩阵通常产生一个完整的矩阵。

eye (3)

⇒

Diagonal Matrix

1 0 0

0 1 0

0 0 1

eye(3)(1:3,:)

⇒

Permutation Matrix

1 0 0

0 1 0

0 0 1

其他一些内置函数也可以返回排列矩阵。例子包括inv或lu。

**21.1.3显式和隐式转换**

对角线矩阵和排列矩阵在它们自己的权利是特殊的对象。为这些矩阵定义了许多操作和内置函数，以便使用特殊的、比在同一位置使用完整矩阵更有效的代码。示例将在后面的章节中给出。

为了方便与全矩阵的平滑混合、向后兼容性以及与MATLAB的兼容性，对角矩阵和置换矩阵应该允许在全矩阵上工作的任何操作，并且将对其进行特殊处理，或者隐式地将自己转换为全矩阵。

实例包括矩阵索引(提取单个元素或前导子矩阵除外)、索引赋值或应用大多数映射器函数(如exp)。

可以使用内置函数full请求显式转换为完整矩阵。还应该注意的是，对角线和排列矩阵对象将在第一次请求转换结果(显式或隐式)后缓存转换结果，因此后续转换将非常便宜。

**21.2具有对角/置换矩阵的线性代数**

正如已经说过的，对角矩阵和置换矩阵使得在保持自然线性代数语法的同时使用有效的算法成为可能。本节详细描述在这些特殊矩阵对象上执行的特殊操作。

**21.2.1涉及对角矩阵的表达式**

假设D是一个对角矩阵。如果M是一个满矩阵，那么D\*M将缩放M的行，这意味着，如果S = D\*M，那么对于每一对指标i,j，它成立

S(i,j) = D(i,i) \* M(i,j).

类似地，M\*D将进行列缩放。

矩阵D也可以是矩形的，m × n，其中m != n。如果m < n，则表达式D\* m等价于

D(:,1:m) \* M(1:m,:),

即，忽略后面的n-m行M。若m > n，则D\* m等价于

[D(1:n,:) \* M; zeros(m-n, columns (M))],

即，将空行追加到结果。右乘M\*D的情况也是类似的。

表达式D \ M和M / D执行逆标度。它们等价于在最小二乘最小范数意义上求解对角线(或矩形对角线)。在精确算术中，这等价于乘以一个伪逆。一个矩形对角矩阵的伪逆也是一个交换了维数的矩形对角矩阵，其中每个非零对角元素被它的倒数所取代。事实上，矩阵除法算法确实使用除法而不是用倒数乘法来获得更好的数值精度;否则，他们遵守上述定义。注意，对角矩阵永远不会因为条件不良而被截断;否则，它对扩展没有多大用处。这通常符合线性代数的需要。一个恰好是对角线的完整矩阵(因此不是一个特殊的对象)当然是正常处理的。

对角矩阵的乘法和除法在与稀疏矩阵结合时也能有效地工作，即D\*S，其中D为对角矩阵，S为稀疏矩阵，对稀疏矩阵的行进行缩放并返回稀疏矩阵。表达式S\*D, D\S, S/D可以类比地工作。

如果D1和D2都是对角矩阵，那么表达式

D1 + D2

D1 - D2

D1 \* D2

D1 / D2

D1 \ D2

再次生成对角矩阵，前提是遵循标准尺寸匹配规则。使用的关系与上面描述的相同。

此外，对角矩阵D可以乘以或除以一个标量，或者如果它是平方，则可以乘以一个标量幂，从而产生所有情况下的对角矩阵结果。

对角矩阵也可以进行转置或共轭转置，得到预期的结果。提取对角矩阵的前导子矩阵，即D(1:m,1:n)，生成对角矩阵，其他索引表达式隐式转换为满矩阵。

将一个对角矩阵添加到一个满矩阵中，只对对角元素进行操作。因此,

A = A + eps \* eye (n)

是增大矩阵对角线的有效方法。减法的工作原理类似。

当涉及到其他逐元素运算符(.\*，./，.\或.^)的表达式时，将发生到完整矩阵的隐式转换。这并不总是严格必要的，但选择它是为了更好地与MATLAB保持一致。

**21.2.2涉及置换矩阵的表达式**

如果P是一个排列矩阵，M是一个矩阵，则表达式P\*M将对M的行进行排列。类似地，M\*P将产生列排列。矩阵除法P\M和M/P可用于逆置换。

前面描述的用于创建置换矩阵的语法实际上可以帮助用户理解置换矩阵和置换向量之间的联系。即，下式成立，其中I = eye (n)为单位矩阵:

I(p,:) \* M = (I\*M) (p,:) = M(p,:)

相似，

M \* I(:,p) = (M\*I) (:,p) = M(:,p)

表达式I(p，:)和I(:，p)是置换矩阵。

置换矩阵可以被转置(或共轭转置，这是一样的，因为置换矩阵从来都不是复矩阵)，置换逆，或者等价地，将行置换矩阵变成列置换矩阵。对于置换矩阵，转置等于反转，因此P\M等于P'\*M。置换矩阵的转置(或逆)是一个常数时间操作，只在内部翻转一个标志，因此在上述两个等价的逆置换表达式之间的选择完全取决于用户的口味。

置换矩阵的乘法和除法在与稀疏矩阵结合时也能有效地工作，即P\*S，其中P为置换矩阵，S为稀疏矩阵，对稀疏矩阵的行进行排列，返回稀疏矩阵。表达式S\*P, P\S, S/P可以类比地工作。

两个排列矩阵可以相乘或相除(如果它们的大小匹配)，从而执行排列组合。排列矩阵也可以用一个(或两个)排列向量来索引，从而得到一个排列矩阵。任何其他操作通常不会产生置换矩阵，因此将触发隐式转换。

**21.3知道这些矩阵的函数**

本节列出了内置函数，这些函数在输入时可以识别对角矩阵和排列矩阵，或者可以将它们作为输出返回。将这些矩阵传递给其他函数，通常会触发隐式转换。(当然，用户定义的动态链接函数也可以使用对角矩阵或排列矩阵)。

**21.3.1对角矩阵函数**

Inv和pinv可以应用于一个对角矩阵，得到一个对角矩阵。当给定对角矩阵时，Det将使用一个有效的直接计算，以及cond。以下映射函数可以应用于对角矩阵，而无需将其转换为完整的对角矩阵:abs, real, image, conj, sqrt。对角矩阵也可以从平衡和svd函数返回。稀疏函数将对角矩阵有效地转换为稀疏矩阵。

**21.3.2置换矩阵函数**

Inv和pinv对置换矩阵进行逆，保持其特殊性。Det可以应用于置换矩阵，有效地计算置换的符号(等于行列式)。

如果请求主因子分解，也可以从内置函数lu和qr返回置换矩阵。

稀疏函数将置换矩阵有效地转换为稀疏矩阵。find函数还可以有效地处理置换矩阵，从而可以方便地获得置换索引。

**21.4用法示例**

对于线性系统a \*x = b，可以用主点LU分解求解:

[L, U, P] = lu (A); ## now L\*U = P\*A

x = U \ (L \ P) \* b;

这是将矩阵X的列归一化为单位范数的一种方法:

s = norm (X, "columns");

X /= diag (s);

广播也可以做到这一点(见广播):

s = norm (X, "columns");

X ./= s;

下面的表达式是一种有效地计算排列符号的方法，由排列向量p给出。它也可以在Octave的早期版本中工作，但速度很慢。

det (eye (length (p))(p, :))

最后，这里是如何解决一个线性系统a \*x = b与吉洪诺夫正则化(脊回归)使用SVD(仅骨架):

m = rows (A); n = columns (A);

[U, S, V] = svd (A);

## determine the regularization factor alpha

## alpha = …

## transform to orthogonal basis

b = U'\*b;

## Use the standard formula, replacing A with S.

## S is diagonal, so the following will be very fast and accurate.

x = (S'\*S + alpha^2 \* eye (n)) \ (S' \* b);

## transform to solution basis

x = V\*x;

**22稀疏矩阵**

**22.1稀疏矩阵的创建和操作**

在任何特定时间可以处理的数学问题的大小通常受到可用计算资源的限制。计算机的速度和可用内存都限制了问题的大小。

有许多数学问题会产生矩阵，其中有大量的元素为零。在这种情况下，有一个特殊的矩阵类型来处理这类只存储矩阵的非零元素的问题是有意义的。这不仅减少了存储矩阵的内存量，而且还意味着对这种类型的矩阵的操作可以利用对非零元素位置的先验知识来加速计算。

仅存储非零元素的矩阵类型通常称为稀疏矩阵。本文的目的是讨论稀疏矩阵的存储和创建的基础知识以及对它们的基本操作。

**22.1.1稀疏矩阵的存储**

严格来说，用户并不需要了解稀疏矩阵是如何存储的。然而，这样的理解将有助于理解稀疏矩阵的大小。对于希望创建自己的oct文件的用户来说，理解存储技术也是必要的。

有许多不同的方法来存储稀疏矩阵数据。所有这些方法的共同之处在于，它们都试图在给定要解决的特定问题类别的先验知识的情况下降低复杂性和存储空间。Saad 8很好地总结了存储稀疏矩阵的可用技术。对于满矩阵，矩阵中一个元素的点的知识是由它在计算机内存中的位置隐含的。然而，这不是稀疏矩阵的情况，因此矩阵的非零元素的位置必须相等

一种显而易见的方法是将矩阵的元素存储为三元组，其中两个元素是它们在数组中的位置(行和列)，第三个元素是数据本身。这在概念上很容易掌握，但需要比严格需要更多的存储空间。

Octave中使用的存储技术是压缩列格式。它与耶鲁大学的形式类似。在这种格式中，一行中每个元素的位置和数据都像以前一样存储。然而，如果我们假设同一列中的所有元素相邻地存储在计算机内存中，那么我们只需要存储每列中非零元素的数量信息，而不是它们的位置。因此，假设矩阵的非零元素多于矩阵中的列，我们在使用的内存量方面获胜。

实际上，列索引比列数多包含一个元素，第一个元素总是为零。这样做的好处是简化了代码，因为第一列或最后一列没有特殊情况。用C语言演示这一点的一个简短示例是。

for (j = 0; j < nc; j++)

for (i = cidx(j); i < cidx(j+1); i++)

printf ("nonzero element (%i,%i) is %d\n",

ridx(i), j, data(i));

通过考虑如何将上述方法应用于示例矩阵的示例，可能会有一个清晰的理解。考虑矩阵

1 2 0 0

0 0 0 3

0 0 0 4

这个矩阵的非零元素是

(1, 1) ⇒ 1

(1, 2) ⇒ 2

(2, 4) ⇒ 3

(3, 4) ⇒ 4

这将被存储为三个向量cidx, ridx和data，分别表示列索引，行索引和数据。上述矩阵的这三个向量的内容是

cidx = [0, 1, 2, 2, 4]

ridx = [0, 0, 1, 2]

data = [1, 2, 3, 4]

请注意，这是假设第一行和第一列从0开始的这些元素的表示，而在Octave本身中，行和列索引从1开始。因此，第i列中的元素个数由cidx (i + 1) - cidx (i)给出。

虽然Octave使用压缩列格式，但应该注意的是，压缩行格式也同样可行。然而，在混合稀疏矩阵和密集矩阵之间的混合运算环境中，稀疏矩阵的元素与密集矩阵的顺序相同是有意义的。八度音阶以列主序存储密集矩阵，因此稀疏矩阵也同样以这种方式存储。

对Octave使用的稀疏矩阵存储的进一步约束是，行中的所有元素都按照其行索引的递增顺序存储，这使得某些操作更快。但是，它在创建稀疏矩阵时需要对元素进行排序。拥有无序元素是一个潜在的优势，因为它使诸如将两个稀疏矩阵连接在一起这样的操作更容易和更快，但是它在其他地方增加了复杂性和速度问题。

**22.1.2创建稀疏矩阵**

有几种方法可以创建稀疏矩阵。

从函数返回

有许多函数直接返回稀疏矩阵。这些包括speye, spread, diag等。

由矩阵或向量构成的

函数稀疏允许从表示行、列和数据的三个向量构造稀疏矩阵。另外，函数spconvert使用三列矩阵格式，允许从其他地方轻松导入数据。

创建，然后填充

函数sparse或spalloc可用于创建一个空矩阵，然后由用户填充

从一个用户二进制程序

用户可以直接在oct文件中创建稀疏矩阵。

有几个基本函数可以返回特定的稀疏矩阵。例如稀疏单位矩阵，就是一种经常需要用到的矩阵。因此，它有自己的函数将其创建为speye (n)或speye (r, c)，这创建了一个n × n或r × c的稀疏单位矩阵。

另一个经常需要的典型稀疏矩阵是随机元素的随机分布。函数sprand和sprandn对元素的均匀和正态随机分布执行此操作。它们具有完全相同的调用约定，其中sprand (r, c, d)创建一个r × c的稀疏矩阵，其密度为d的填充元素。

其他直接产生稀疏矩阵的函数，是diag或者它的泛化spdiag，它可以取矩阵对角线的定义并产生对应于它的稀疏矩阵。例如,

s = diag (sparse (randn (1,n)), -1);

创建一个定义了单个对角线的稀疏(n+1) × -(n+1)稀疏矩阵。

: B = spdiags (A)

: [B, d] = spdiags (A)

: B = spdiags (A, d)

: A = spdiags (v, d, A)

: A = spdiags (v, d, m, n)

函数图的推广。

使用单个输入参数调用，提取a的非零对角线d。

有两个参数，要提取的对角线由向量d给出。

另外两种形式的对角线通过替换对角线来修改输入矩阵。它们用v的列来替换向量d所表示的对角线。如果定义了稀疏矩阵A，那么这个矩阵的对角线就被替换。否则一个m × n的矩阵由v的列给出的对角线组成。

负值d表示主对角线以下的对角线，正值d表示主对角线以上的对角线。

例如:

spdiags (reshape (1:12, 4, 3), [-1 0 1], 5, 4)

⇒ 5 10 0 0

1 6 11 0

0 2 7 12

0 0 3 8

0 0 0 4

参见:diag。

: s = speye (m, n)

: s = speye (m)

: s = speye (sz)

返回大小为mxn的稀疏单位矩阵。

由于没有构造完整的矩阵，因此实现比sparse (eye (m))明显更高效。

调用一个参数，创建一个大小为m × m的方阵。如果使用单个向量参数sz调用，则该参数将被视为要创建的矩阵的大小。

又见:稀疏的，散裂的，眼。

: r = spones (S)

将S中的非零项替换为1。

这创建了一个与S具有相同结构的稀疏矩阵。

参见:sparse, sprand, sprandn, sprandsym, spfun, spy。

: s = sprand (m, n, d)

: s = sprand (m, n, d, rc)

: s = sprand (s)

生成具有均匀分布的随机值的稀疏矩阵。

矩阵的大小为mxn，密度为d。d必须在0到1之间。值将均匀分布在区间(0,1)上。

如果使用单个矩阵参数调用，则在矩阵s非零的地方生成具有随机值的稀疏矩阵。

如果使用标量第四个参数rc调用，则生成一个具有互反条件数rc的随机稀疏矩阵。如果rc是一个向量，则它指定生成的矩阵(length (rc) <= min (m, n))的第一个奇异值。

参见:sprandn, sprandsym, rand。

: s = sprandn (m, n, d)

: s = sprandn (m, n, d, rc)

: s = sprandn (s)

生成一个具有正态分布随机值的稀疏矩阵。

矩阵的大小为mxn，密度为d。d必须在0到1之间。数值将呈正态分布，均值为0，方差为1。

如果使用单个矩阵参数调用，则在矩阵s非零的地方生成具有随机值的稀疏矩阵。

如果使用标量第四个参数rc调用，则生成一个具有互反条件数rc的随机稀疏矩阵。如果rc是一个向量，则它指定生成的矩阵(length (rc) <= min (m, n))的第一个奇异值。

参见:spread, sprandsym, randn。

: S = sprandsym (n, d)

: S = sprandsym (s)

生成一个对称随机稀疏矩阵。

矩阵的大小将是nxn，其密度由d给出。d必须在0到1之间。值将呈正态分布，均值为0，方差为1。

如果使用单个矩阵参数调用，则在矩阵s的下三角部分非零的地方生成一个随机稀疏矩阵。

参见:sprand, sprandn, spones, sparse。

用户创建稀疏矩阵的推荐方法是创建包含数据的行和列索引的两个向量，以及包含要存储的数据的相同大小的第三个向量。例如,

ri = ci = d = [];

for j = 1:c

ri = [ri; randperm(r,n)'];

ci = [ci; j\*ones(n,1)];

d = [d; rand(n,1)];

endfor

s = sparse (ri, ci, d, r, c);

创建一个r × c稀疏矩阵，每列随机分布n (<r)个元素。向量的元素不需要按照任何特定的顺序排序，因为Octave会在存储数据之前对它们进行排序。然而，对数据进行预排序将使稀疏矩阵的创建更快。

函数spconvert接受一个三列或四列实矩阵。前两列分别表示行索引和列索引，第三列和第四列分别表示稀疏矩阵的实部和虚部。矩阵可以包含零元素，并且元素可以按任意顺序排序。添加零元素是定义稀疏矩阵大小的一种方便方法。例如:

s = spconvert ([1 2 3 4; 1 3 4 4; 1 2 3 0]')

⇒ Compressed Column Sparse (rows=4, cols=4, nnz=3)

(1 , 1) -> 1

(2 , 3) -> 2

(3 , 4) -> 3

创建和填充矩阵的一个示例可能是

k = 5;

nz = r \* k;

s = spalloc (r, c, nz)

for j = 1:c

idx = randperm (r);

s (:, j) = [zeros(r - k, 1); ...

rand(k, 1)] (idx);

endfor

应该注意的是，由于Octave赋值函数的编写方式，赋值将在上述循环的每次迭代中重新分配稀疏矩阵使用的内存。因此，spalloc函数忽略nz参数，并且不预先为矩阵分配内存。因此，至关重要的是，使用上述结构的代码应该尽可能地向量化，以尽量减少分配的数量和减少内存分配的数量。

: FM = full (SM)

从稀疏矩阵、对角矩阵或排列矩阵或范围返回一个完整的存储矩阵。

参见:sparse, issparse。

: s = spalloc (m, n, nz)

创建一个m × n的稀疏矩阵，该矩阵具有最多nz个非零元素的预分配空间。

这对于通过索引赋值序列增量地构建矩阵非常有用。在空间分配之后的后续索引赋值将重用预分配的内存，前提是它们是一种简单的形式

* s(I:J) = x
* s(:,I:J) = x
* s(K:L,I:J) = x

并满足下列条件:

* 赋值不减少nnz (S)。
* 赋值后，nnz (S)不超过nz。
* 没有索引越界。

数据的部分移动可能仍然会发生，但通常在这种情况下，分配将更节省内存和时间。特别是，可以有效地从连续的列块中构建预分配的稀疏矩阵。

可以使用函数nzmax查询给定矩阵的预分配内存量。

编程注意:Octave总是为至少一个值保留内存，即使nz为0。

参见:nzmax, sparse。

: S = sparse (A)

: S = sparse (m, n)

: S = sparse (i, j, sv)

: S = sparse (i, j, sv, m, n)

: S = sparse (i, j, sv, m, n, "unique")

: S = sparse (i, j, sv, m, n, nzmax)

从一个完整矩阵a或行、列、值三元组创建一个稀疏矩阵。

如果A是一个满矩阵，则将其转换为稀疏矩阵表示，在此过程中删除所有零值。矩阵A应该是逻辑类型或双精度类型。

如果指定了两个输入m(行)和n(列)，则创建具有指定维数的空稀疏矩阵。

给定整数索引向量i和j，以及实数或复数值sv的1 × nnz向量，构造总体维数为m和n的稀疏矩阵S(i(k)，j(k)) = sv(k)。如果i, j或sv中的任何一个是标量，则将它们展开为一个公共大小。

如果m或n未指定，则它们的值由向量i和j中的最大索引导出，如m = max (i)， n = max (j)所给出的。

注:如果指定了多个具有相同i, j索引的值，则S中对应的值为重复位置的值之和。有关如何产生不同行为(例如取最小值)的示例，请参阅accumarray。

如果给出了“unique”选项，并且在相同的i, j索引处指定了多个值，则只使用最后指定的值。为了完整起见，可以给出选项"sum"，该选项将被忽略，因为默认行为是对重复位置的值求和。

稀疏(m, n)将创建一个空的MXN稀疏矩阵，等价于稀疏([]，[]，[]，m, n)

可选的final参数为稀疏数组中的nzmax值保留了空间，如果最终非零值的数量大于数组初始构造期间使用的sv值的数量，则很有用。有关更多信息和使用说明，请参阅空间分配。

示例1(将完整矩阵转换为稀疏矩阵以节省内存):

x = full (diag (1:1000));

sizeof (x)

⇒ 8000000

s = sparse (x);

sizeof (xs)

⇒ 24008

例2(重复索引求和):

i = [1 1 2]; j = [1 1 2]; sv = [3 4 5];

sparse (i, j, sv, 3, 4)

⇒

Compressed Column Sparse (rows = 3, cols = 4, nnz = 2 [17%])

(1, 1) -> 7

(2, 2) -> 5

示例3 ("unique"选项):

i = [1 1 2]; j = [1 1 2]; sv = [3 4 5];

sparse (i, j, sv, 3, 4, "unique")

⇒

Compressed Column Sparse (rows = 3, cols = 4, nnz = 2 [17%])

(1, 1) -> 4

(2, 2) -> 5

参见:full, accumarray, spalloc, spdiags, speye, spones, sprand, sprandn, sprandsym, spconvert, spfun。

: x = spconvert (m)

将其他程序容易生成的简单稀疏矩阵格式转换为Octave的内部稀疏格式。

输入m是一个3列或4列实矩阵，包含稀疏矩阵元素的行、列、实部和虚部。实部和虚部为零的元素可以用来强制一个特定的矩阵大小。

参见: sparse.

在oct文件中可以避免上述内存重新分配问题。然而，从oct文件构造稀疏矩阵要比这里讨论的复杂得多。有关所涉及的技术的完整描述，请参阅外部代码接口。

**22.1.3查找稀疏矩阵信息**

有许多函数允许获得关于稀疏矩阵的信息。其中最基本的是issparse，它可以识别一个特定的Octave对象是否实际上是一个稀疏矩阵。

另一个非常基本的函数是nnz，它返回稀疏矩阵中非零条目的数量，而函数nzmax返回分配给稀疏矩阵的存储量。注意，对于稀疏对象，Octave倾向于在第一次机会时裁剪未使用的内存。在某些情况下，用户创建的稀疏对象的nzmax返回的值与nnz不一样，但通常它们会给出相同的结果。函数spstats返回关于稀疏矩阵列的一些基本统计信息，包括元素的数量、每列的平均值和方差。

: tf = issparse (x)

如果x是稀疏矩阵则返回true。

参见:ismatrix。

: n = nnz (A)

返回A中非零元素的个数。

参见:nzmax，非零，find。

: v = nonzeros (A)

返回矩阵a的非零值的列向量。

参见:find, nnz。

: n = nzmax (SM)

返回分配给稀疏矩阵SM的存储量。

编程注意:对于稀疏对象，Octave倾向于在第一次机会时裁剪未使用的内存。因此，除了某些用户创建的稀疏对象外，通常nzmax的值将与nnz相同。

另外，注意Octave总是为至少一个值保留存储空间。因此，对于空矩阵，nnz将报告0，但nzmax将报告1。

参见:nnz, spalloc, sparse。

: [count, mean, var] = spstats (S)

: [count, mean, var] = spstats (S, j)

返回稀疏矩阵S的非零元素的统计信息。

Count是每列中非零的个数，mean是每列中非零的均值，var是每列中非零的方差。

使用两个输入参数调用，如果S是数据，j是数据的bin号，则计算每个bin的统计信息。在这种情况下，bin可以包含0的数据值，而使用spstats (S)时，0可能会消失。

当求解涉及稀疏矩阵的线性方程时，Octave根据矩阵的类型确定求解方程的方法(参见稀疏矩阵上的线性代数)。当div(/)或ldiv(\)运算符首次用于矩阵时，Octave探测矩阵类型，然后缓存该类型。但是，在使用div或ldiv操作符之前，可以使用matrix\_type函数来确定稀疏矩阵的类型。例如,

a = tril (sprandn (1024, 1024, 0.02), -1) ...

+ speye (1024);

matrix\_type (a);

ans = Lower

表明Octave正确地确定了下三角矩阵的矩阵类型。Matrix\_type也可用于强制矩阵的类型为特定类型。例如:

a = matrix\_type (tril (sprandn (1024, ...

1024, 0.02), -1) + speye (1024), "Lower");

这样就可以避免确定矩阵类型的成本。然而，不正确地定义矩阵类型将导致线性方程解的结果不正确，因此正确识别矩阵类型完全是用户的责任

有几种图形方法可以找到关于稀疏矩阵的信息。第一个是spy命令，它显示矩阵的非零元素的结构。图22.1是使用spy的一个示例。可以使用treeplot、etreeplot和gplot命令获得更高级的图形信息。

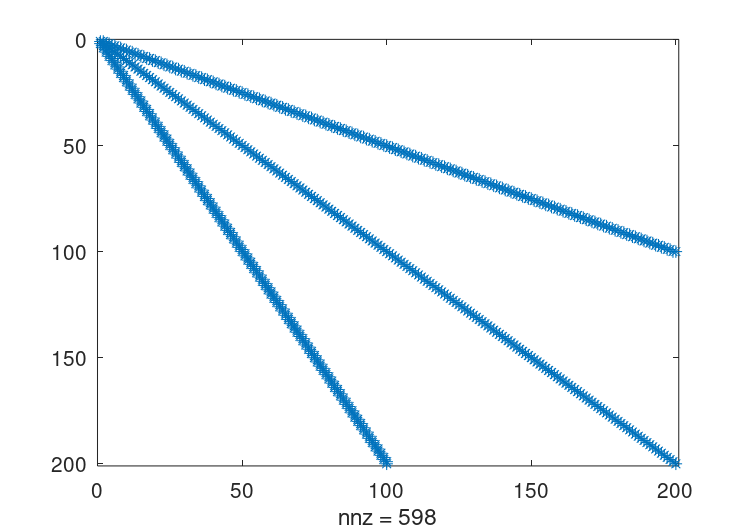


图22.1简单稀疏矩阵的结构。

稀疏矩阵的一个用途是在图论中，其中节点之间的互连被表示为邻接矩阵。也就是说，如果图中的第i个节点与第j个节点相连。那么稀疏邻接矩阵的第ij个节点(在无向图的情况下是第j个节点)是非零的。如果每个节点都与一组坐标相关联，那么可以使用gplot命令以图形方式显示节点之间的相互连接。

作为使用gplot的一个简单示例，请考虑以下示例:

A = sparse ([2,6,1,3,2,4,3,5,4,6,1,5],

[1,1,2,2,3,3,4,4,5,5,6,6],1,6,6);

xy = [0,4,8,6,4,2;5,0,5,7,5,7]';

gplot (A,xy)

它创建了一个邻接矩阵A，其中节点1与节点2和6相连，节点2与节点1和3相连，以此类推。节点的坐标在n × 2矩阵xy中给出。参见图22.2。

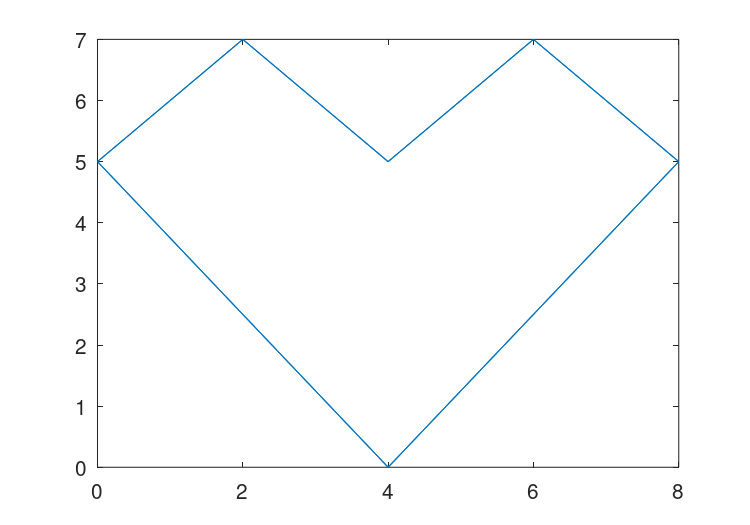


图22.2:gplot命令的简单使用

一个Cholesky分解的节点之间的依赖关系可以在线性时间内计算出来，而不需要通过etree命令显式地计算Cholesky分解。该命令返回矩阵的消去树，如果A是对称的，则可以通过命令treeplot (etree (A))或treeplot (etree (A+A'))以图形方式显示。

: spy (x)

: spy (…, markersize)

: spy (…, line\_spec)

绘制稀疏矩阵x的稀疏模式。

如果给出了可选的数字参数markersize，它将确定绘图中使用的标记的大小。

如果给出了可选字符串line\_spec，则将其传递给plot并确定plot的外观。

参见:plot, gplot。

: p = etree (S)

: p = etree (S, typ)

: [p, q] = etree (S, typ)

返回矩阵S的消去树。

默认情况下，假设S是对称的，并返回对称消去树。参数类型控制是返回对称树还是返回列消除树。type的有效值为“sym”或“col”，分别用于对称或列消除树。

使用第二个参数调用时，etree还返回树的后置排列。

: etreeplot (A)

: etreeplot (A, node\_style, edge\_style)

如果A不是对称的，画出矩阵A或A+A'的消去树。

可选参数node\_style和edge\_style定义了输出样式。

参见:treeplot, gplot。

: gplot (A, xy)

: gplot (A, xy, line\_style)

: [x, y] = gplot (A, xy)

画一个图论意义上由a和xy定义的图。

A是待绘制数组的邻接矩阵，xy是包含图中节点坐标的n × 2矩阵。

可选参数line\_style定义了绘图的输出样式。调用时不带输出参数，将直接绘制图形。否则，返回以x和y表示的绘图坐标。

参见:treeplot, etreeplot, spy。

: treeplot (tree)

: treeplot (tree, node\_style, edge\_style)

画一棵树或森林的图。

第一个参数是vector of predecessor。

可选参数node\_style和edge\_style定义了输出图的样式。

该算法在时间和内存需求方面的复杂度为0 (n)。

参见:etreeplot, gplot。

: [x, y] = treelayout (tree)

: [x, y] = treelayout (tree, permutation)

: [x, y, h, s] = treelayout (…)

Treelayout将树或森林进行布局。

第一个参数树是一个前辈向量。

可选参数permutation是后置排列。

该算法在时间和内存需求方面的复杂度为0 (n)。

参见:etreeplot, gplot, treeplot。

**22.1.4稀疏矩阵上的基本算子和函数**

**22.1.4.1稀疏函数**

许多Octave函数已经被重载以处理稀疏矩阵或满矩阵。当使用带有稀疏矩阵的重载函数时，调用约定没有区别，但是，也无法访问潜在的稀疏特定特性。任何时候都可以通过显式调用函数名来使用函数的稀疏矩阵特定版本。

下表列出了Octave的所有稀疏函数。请注意，函数的特定稀疏形式的名称通常与带有sp前缀的通用版本相同。在下表中以及本文的其余部分中，将使用函数的特定稀疏版本。

生成稀疏矩阵:

spalloc, spdiags, speye, sprand, sprandn, sprandsym

稀疏矩阵转换:

full, sparse, spconvert

操作稀疏矩阵

issparse, nnz, nonzeros, nzmax, spfun, spones, spy

图论：

etree, etreeplot, gplot, treeplot

稀疏矩阵重排序:

amd, ccolamd, colamd, colperm, csymamd, dmperm, symamd, randperm, symrcm

线性代数：

condest, eigs, matrix\_type, normest, normest1, sprank, spaugment, svds

迭代法：

ichol, ilu, pcg, pcr

其它：

spparms, symbfact, spstats

此外，所有标准的Octave映射函数(即，接受单个参数的基本数学函数)，如abs等，都可以接受稀疏矩阵。读者可以参考Octave本身提供的这些函数的文档了解更多细节。

**22.1.4.2操作符和函数的返回类型**

使用稀疏矩阵的两个基本原因是减少内存使用和不必对零元素进行计算。这两者密切相关，因为稀疏矩阵算子或函数的计算时间与非零元素的数量大致成线性关系。

因此，有一定密度的矩阵的非零元素，它不再是有意义的存储为稀疏矩阵，而是作为一个完整的矩阵。由于这个原因，具有高概率返回一个完整矩阵的算子和函数总是返回一个。例如，在稀疏矩阵中加入一个标量常数几乎总是会使它成为一个完整的矩阵，所以这个例子，

speye (3) + 0

⇒ 1 0 0

0 1 0

0 0 1

可以看到，返回一个完整的矩阵。

由于在满矩阵和稀疏矩阵之间的所有混合算子和函数都存在，因此通常不会引起任何问题。然而，它确实会引起问题的一个地方是，当一个稀疏矩阵被提升为一个完整矩阵时，后续的操作将使矩阵重新分配。这种情况很少，但可以人为地创建，例如(fliplr (speye (3)) + speye (3)) - speye(3)给出一个完整的矩阵，而它应该给出一个稀疏的矩阵。一般来说，当这种情况发生时，它们只会造成很小的内存损失。

然而，在一个已知的情况下，Octave的稀疏矩阵的这种行为会导致问题。这是在诊断函数的处理中。diag返回稀疏矩阵还是完整矩阵取决于其输入参数的类型。所以

a = diag (sparse ([1,2,3]), -1);

应该返回一个稀疏矩阵。为了确保这确实发生，稀疏函数以及基于它的其他函数(如speye)总是返回一个稀疏矩阵，即使所使用的内存将大于其完整表示。

**22.1.4.3数学考虑**

已经尝试使稀疏矩阵的行为方式与完全对应的矩阵完全相同。但是，存在一定的差异，特别是与其他产品的稀疏实现的差异。

首先，“。/"和"。^"操作符必须小心使用。考虑一下这些例子

s = speye (4);

a1 = s .^ 2;

a2 = s .^ s;

a3 = s .^ -2;

a4 = s ./ 2;

a5 = 2 ./ s;

a6 = s ./ s;

会给。第一个例子s的2次方没有问题。然而，从元素的角度来看，s包含了大量的项，也就是1。s ^ s是一个满矩阵。

同样s ^ -2也包含了0 ^ -2这是无穷，所以s ^ -2同样是一个满矩阵。

对于“。运算符s ./ 2没有问题，但2 ./ s也包含大量的无穷项，同样是一个满矩阵。s / s的情况涉及到像0 / 0这样的项，这是一个NaN所以这同样是一个完整的矩阵s的0个元素都是NaN值。

上述行为与全矩阵一致，但与其他产品中的稀疏实现不一致。

稀疏矩阵的一个特殊问题是由于零没有被存储，这些零的符号位也同样没有被存储。在某些情况下，0的符号位很重要。例如:

a = 0 ./ [-1, 1; 1, -1];

b = 1 ./ a

⇒ -Inf Inf

Inf -Inf

c = 1 ./ sparse (a)

⇒ Inf Inf

Inf Inf

要纠正这种行为，就意味着需要将带有负符号位的零元素存储在矩阵中，以确保它们的符号位得到尊重。由于效率的原因，此时没有这样做，因此警告用户，在零的符号位很重要的计算中，一定不能使用稀疏矩阵来完成。

通常，对稀疏矩阵使用任何函数或算子都会得到一个与原矩阵具有相同或更多非零元素的稀疏矩阵。这对于稀疏矩阵分解的重要情况尤其正确。通常解决这个问题的方法是对矩阵重新排序，这样它的分解就比原始矩阵的分解更稀疏。即L \* U = P \* S \* Q的分解比等价的L \* U = S的分解有更稀疏的L和U项。

根据要分解的矩阵的类型，有几个函数可用于重新排序。如果矩阵是对称正定的，则应使用符号或c符号。否则应使用amd、colamd或colamd。为了完整起见，还提供了重新排序函数colperm和randperm。

图22.3是一个简单正定矩阵的结构示例。

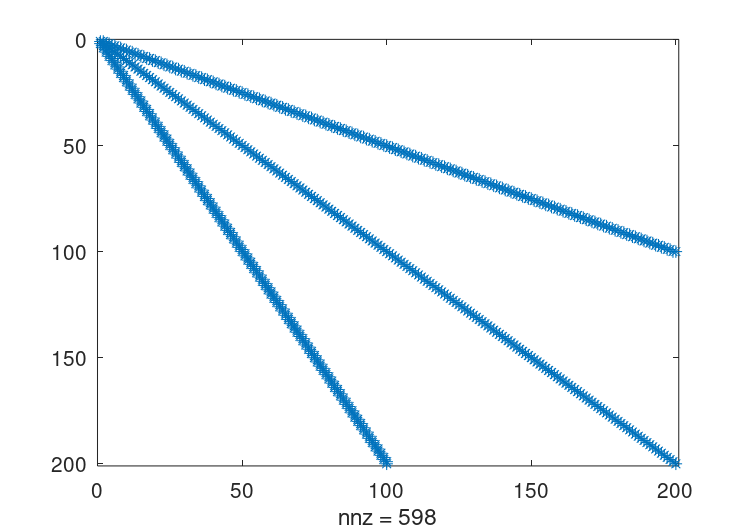


图22.3简单稀疏矩阵的结构。

这个矩阵的标准Cholesky分解可以用与完整矩阵相同的命令得到。这可以用命令r = chol (A)来可视化;间谍(r);。如图22.4所示。原始矩阵有598个非零项，而这个Cholesky分解有10200个，只存储了对称矩阵的一半。这是一个重要的填充级别，尽管对于这样一个小的测试用例来说不是问题，但在处理其他稀疏矩阵时可能会带来很大的开销。

原始矩阵的适当的保持稀疏性的排列由symamd给出，利用这种重排序的分解可以用命令q = symamd (A)来可视化;r = chol (A(q,q));spy (r)。这给出了399个非零项，这是一个显著的改进。

Cholesky分解本身可用于在分解过程中确定适当的保持稀疏性的矩阵重排序，在这种情况下，可以使用三个返回参数[r, p, q] = chol (A);间谍(r)。

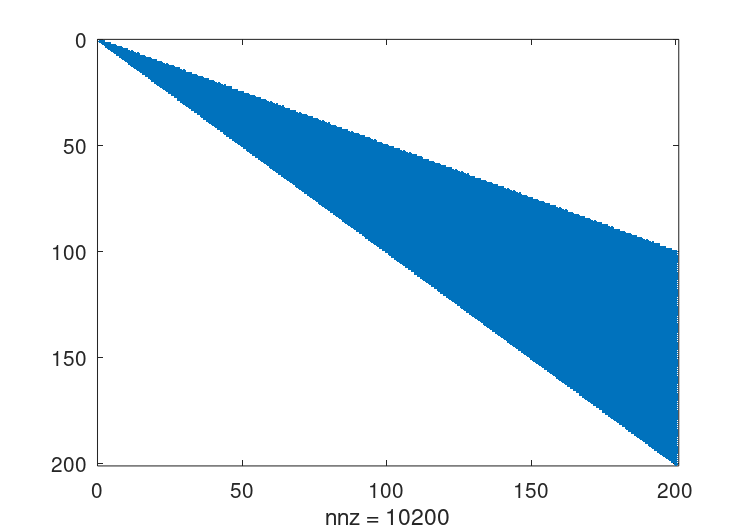


图22.4:上述矩阵的非置换Cholesky分解结构。

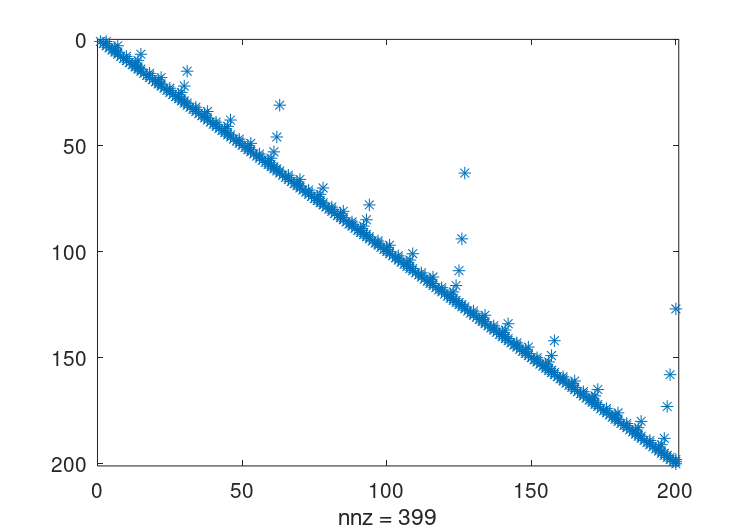


图22.5:上述矩阵的置换Cholesky分解结构。

在非对称矩阵的情况下，适当的保持稀疏性的排列是colamd，使用这种重排序的分解可以使用命令q = colamd (A)来可视化;[1, u, p] = lu (A(:，q));间谍(l + u)。

最后，在使用div(/)和ldiv(\)运算符时，Octave隐式地对矩阵重新排序，因此用户不需要显式地对矩阵重新排序以最大化性能。

: p = amd (S)

: p = amd (S, opts)

返回矩阵的近似最小度置换。

这个排列使得S (p, p)的Cholesky分解比S本身的Cholesky分解更稀疏。Amd通常比symamd快，但目的相似。

可选参数opts是一个控制amd行为的结构。结构的字段是

opts.dense

确定amd认为什么是输入矩阵的密集行或密集列。拥有超过max (16， (dense \* sqrt (n)))个条目的行或列(其中n是矩阵S的阶数)在计算排列期间被amd忽略。dense的值必须为正标量，默认值为10.0

opts.aggressive

如果该值是非零标量，则amd执行主动吸收。默认情况下不执行主动吸收。

代码的作者是Timothy A. Davis(参见http://faculty.cse.tamu.edu/davis/suitesparse.html)。

参见:符号，列。

: p = ccolamd (S)

: p = ccolamd (S, knobs)

: p = ccolamd (S, knobs, cmember)

: [p, stats] = ccolamd (…)

约束列近似最小度排列。

p = ccolamd (S)返回稀疏矩阵S的列近似最小度排列向量。对于非对称矩阵S, S(:， p)倾向于具有比S更稀疏的LU因子。chol (S(:， p)' \* S(:， p))也倾向于比chol (S' \* S)更稀疏。p = ccolamd (S, 1)优化了LU (S(:， p))的排序。排序之后是列消除树后排序。

Knobs是一个可选的1元素到5元素的输入向量，如果不存在或为空，默认值为[0 10 10 10 10 0]。不存在的条目被设置为默认值。

knobs(1)

如果非零，则为lu (S(:， p))优化排序。对于chol (S(:， p)' \* S(:， p))来说，这将是一个糟糕的排序。这是最重要的旋钮。

knobs(2)

如果S是m × n，则忽略超过max (16, knobs(2) \* sqrt (n))项的行。

knobs(3)

超过Max (16, knobs(3) \* SQRT (min (m, n)))项的列将被忽略，并在输出排列中排在最后(受cmember约束)。

knobs(4)

如果非零，则进行侵略性吸收。

knobs(5)

如果非零，则打印统计数据和旋钮。

cmember是长度为n的可选向量。它定义了列排序的约束。如果cmember(j) = c，则列j在约束集c中(c必须在1到n的范围内)。在输出排列p中，集合1中的所有列首先出现，然后是集合2中的所有列，以此类推。如果不存在或为空，Cmember = ones (1,n)。colamd (S， []， 1: n)返回1:n

p = colamd (S)与p = colamd (S)大致相同，但其默认值不同。colamd总是主动吸收，并且它找到一个适合于lu (S(:， p))和chol (S(:， p)' \* S(:， p))的顺序;它不能像colamd (S, 1)那样优化lu (S(:， p))的排序。

stats是一个可选的20元素输出向量，它提供了关于输入矩阵S的排序和有效性的数据。排序统计信息在stats(1: 3)中。stats(1)和stats(2)是CCOLAMD忽略的密集或空行和列的数量，stats(3)是在CCOLAMD使用的内部数据结构上执行的垃圾收集的数量(大小大约为2.2 \* nnz (S) + 4 \* m + 7 \* n个整数)。

stats(4:7)提供了CCOLAMD是否能够继续的信息。如果stats(4)为零，则矩阵为OK，如果无效则为1。Stats(5)是最右边未排序或包含重复条目的列索引，如果不存在这样的列，则为零。Stats(6)是Stats(5)给出的列索引中最后一次看到的重复或乱序的行索引，如果不存在这样的行索引，则为零。Stats(7)是重复或乱序的行索引的数量。stats(8:20)在当前版本的CCOLAMD中始终为零(保留以供将来使用)。

代码本身的作者是S. Larimore, T. Davis和S. Rajamanickam与J. Bilbert和E. Ng合作编写的。由美国国家科学基金会(DMS-9504974, DMS-9803599, CCR-0203270)和桑迪亚国家实验室资助。请参阅http://faculty.cse.tamu.edu/davis/suitesparse.html了解colamd、csymamd、amd、colamd、symamd以及其他相关排序。

参见:colamd, csymamd。

: p = colamd (S)

: p = colamd (S, knobs)

: [p, stats] = colamd (S)

: [p, stats] = colamd (S, knobs)

计算列近似最小度排列。

p = colamd (S)返回稀疏矩阵S的列近似最小度排列向量。对于非对称矩阵S, S(:，p)倾向于具有比S更稀疏的LU因子。S(:，p)' \* S(:，p)的Cholesky分解也倾向于比S' \* S更稀疏。

Knobs是一个可选的一到三元素输入向量。如果S是m × n，那么超过max(16,knobs(1)\*sqrt(n))项的行将被忽略。大于max (16,knobs(2)\*sqrt(min(m,n)))项的列在排序之前被删除，并且在输出排列p中排序最后。如果knobs(1)和knobs(2)分别< 0，则只删除完全密集的行或列。如果knobs(3)不为零，则打印状态和旋钮。默认值是knobs =[10 10 0]。请注意，旋钮与早期版本的colamd不同。

stats是一个可选的20个元素的输出向量，它提供了关于输入矩阵s的排序和有效性的数据。排序统计信息在stats(1:3)中。stats(1)和stats(2)是COLAMD忽略的密集或空行和列的数量，stats(3)是COLAMD使用的内部数据结构上执行的垃圾收集的数量(大小大致为2.2 \* nnz(S) + 4 \* m + 7 \* n个整数)。

Octave内置函数旨在生成有效的稀疏矩阵，没有重复的条目，每列中非零的行索引是升序的，每列中有非负数的条目(!)等等。如果矩阵无效，则COLAMD可能无法继续执行。如果存在重复条目(一个行索引在同一列中出现两次或两次以上)，或者如果列中的行索引无序，那么COLAMD可以通过忽略重复条目并对矩阵S的内部副本的每个列进行排序来纠正这些错误(但是，输入矩阵S没有修复)。如果矩阵在其他方面无效，则COLAMD无法继续，打印错误消息，并且不返回输出参数(p或stats)。因此，COLAMD是一种检查稀疏矩阵是否有效的简单方法。

stats(4:7)提供COLAMD是否能够继续的信息。如果stats(4)为零，则矩阵为OK，如果无效则为1。Stats(5)是最右边未排序或包含重复条目的列索引，如果不存在这样的列，则为零。Stats(6)是Stats(5)给出的列索引中最后一次看到的重复或乱序的行索引，如果不存在这样的行索引，则为零。Stats(7)是重复或乱序的行索引的数量。stats(8:20)在COLAMD的当前版本中始终为零(保留以供将来使用)。

排序之后是列消除树后排序。

代码本身的作者是Stefan I. Larimore和Timothy A. Davis。该算法是与施乐帕洛阿尔托研究中心的约翰·吉尔伯特和橡树岭国家实验室的埃斯蒙德·吴恩达合作开发的。(参见http://faculty.cse.tamu.edu/davis/suitesparse.html)

参见:colperm, symamd, colcolamd。

: p = colperm (s)

返回列排列，使得s(:， p)的列按照非零元素的数量递增进行排序。

如果s是对称的，则选择p，使得s(p, p)的行和列的顺序随着非零元素数量的增加而增加。

: p = csymamd (S)

: p = csymamd (S, knobs)

: p = csymamd (S, knobs, cmember)

: [p, stats] = csymamd (…)

对于对称正定矩阵S，返回排列向量p，使得S(p,p)趋向于具有比S更稀疏的Cholesky因子。

有时c符号也适用于对称不定矩阵。假设矩阵S是对称的;只有严格的下三角形部分被引用。S一定是平方的。排序之后是消去树后排序。

Knobs是一个可选的1元素到3元素的输入向量，默认值为[10 10 0]。不存在的条目被设置为默认值。

knobs(1)

如果S是n × n，那么包含超过max(16，旋门(1)\*sqrt(n))项的行和列将被忽略，并在输出排列中排在最后(受cmember约束)。

knobs(2)

如果非零，则进行侵略性吸收。

knobs(3)

如果非零，则打印统计数据和旋钮。

cmember是长度为n的可选向量。它定义了排序的约束。如果cmember(j) = S，则行/列j在约束集c中(c必须在1到n的范围内)。在输出排列p中，集合1中的行/列首先出现，然后是集合2中的所有行/列，以此类推。如果不存在或为空，Cmember = ones (1,n)。csymamd (S，[]，1:n)返回1:n。

p = csymamd (S)与p = symamd (S)大致相同，但其默认值不同。

stats(4:7)提供了CCOLAMD是否能够继续的信息。如果stats(4)为零，则矩阵为OK，如果无效则为1。Stats(5)是最右边未排序或包含重复条目的列索引，如果不存在这样的列，则为零。Stats(6)是Stats(5)给出的列索引中最后一次看到的重复或乱序的行索引，如果不存在这样的行索引，则为零。Stats(7)是重复或乱序的行索引的数量。stats(8:20)在当前版本的CCOLAMD中始终为零(保留以供将来使用)。

代码本身的作者是S. Larimore, T. Davis和S. Rajamanickam与J. Bilbert和E. Ng合作编写的。由美国国家科学基金会(DMS-9504974, DMS-9803599, CCR-0203270)和桑迪亚国家实验室资助。请参阅http://faculty.cse.tamu.edu/davis/suitesparse.html了解colamd、colamd、csymamd、amd、colamd、symamd以及其他相关顺序。

参见:symamd, colamd。

: p = dmperm (A)

: [p, q, r, s, cc, rr] = dmperm (A)

对稀疏矩阵a进行Dulmage-Mendelsohn置换。

使用单个输出参数dmperm，返回一个最大匹配p，如果j列与第i行匹配，则p(j) = i，如果j列不匹配，则p(j) = 0。如果A是方阵且是全结构秩，则p是行排列，并且A(p，:)具有无零对角线。A的结构秩为sprank(A) = sum(p>0)。

用两个或多个输出参数调用，返回a的Dulmage-Mendelsohn分解。p和q是置换向量。Cc和rr是长度为5的向量。c = A(p,q)分成4 × 4的粗块集:

A11 A12 A13 A14

0 0 A23 A24

0 0 0 A34

0 0 0 A44

其中A12, A23和A34是零对角线的正方形。A11的列为未匹配的列，A44的行为未匹配的行。这些块中的任何一个都可以是空的。在“粗”分解中，第(i,j)块是C(rr(i):rr(i+1)-1,cc(j):cc(j+1)-1)。对于线性系统，[A11 A12]是系统的欠定部分(它总是矩形的，有更多的列和行，或0 × 0)， A23是系统的确定部分(它总是正方形的)，[A34;A44]是系统的过度确定部分(它总是矩形，行多于列，或0 × 0)。

A的结构秩为sprank(A) = rr(4)-1，这是A的数值秩的上界。在精确算术中，sprank(A) = rank(full(sprand(A)))的概率为1。

通过“精细”分解(A23的强连接分量)，将A23子矩阵进一步细分为块状上三角形。如果A是方阵且结构上非奇异，则A23是整个矩阵。

C(r(i):r(i+1)-1,s(j):s(j+1)-1)是精细分解的(i,j) -1块。(1,1)块是矩形块[A11 A12]，除非该块是0 × 0。(b,b)块为矩形块[A34;A44]，除非该块是0 × 0，其中b = length(r)-1。所有其他形式的块C(r(i):r(i+1)-1,s(i):s(i+1)-1)是A23的对角线块，并且是无零对角线的正方形。

所采用的方法见:A. Pothen & C.-J。粉丝。计算稀疏矩阵的块三角形式。ACM反式。数学。软件学报，16(4):303-324,1990。

参见:colamd, colamd。

: p = symamd (S)

: p = symamd (S, knobs)

: [p, stats] = symamd (S)

: [p, stats] = symamd (S, knobs)

对于对称正定矩阵S，返回使S(p, p)趋向于具有比S更稀疏的Cholesky因子的排列向量p。

有时候符号也适用于对称不定矩阵。假设矩阵S是对称的;只有严格的下三角形部分被引用。S一定是平方的。

Knobs是一个可选的一到两元素输入向量。如果S是n × n，则在排序之前删除大于max (16,knobs(1)\*sqrt(n))项的行和列，并在输出排列p中最后排序。如果knobs(1) < 0，则不删除任何行/列。如果knobs(2)不为零，则打印状态和旋钮。默认值是knobs =[10 0]。请注意，knobs与symamd的早期版本不同。

stats是一个可选的20个元素的输出向量，它提供了关于输入矩阵s的排序和有效性的数据。排序统计信息在stats(1:3)中。stats(1) = stats(2)是SYMAMD忽略的密集或空行和列的数量，stats(3)是SYMAMD使用的内部数据结构(大小大约为8.4 \* nnz (tril (S， -1)) + 9 \* n个整数)上执行的垃圾收集的数量。

Octave内置函数旨在生成有效的稀疏矩阵，没有重复的条目，每列中非零的行索引是升序的，每列中有非负数的条目(!)等等。如果矩阵无效，则SYMAMD可能无法继续运行。如果存在重复条目(一个行索引在同一列中出现两次或两次以上)，或者如果列中的行索引无序，那么SYMAMD可以通过忽略重复条目并对矩阵S的内部副本的每个列进行排序来纠正这些错误(但是，输入矩阵S不会被修复)。如果矩阵在其他方面无效，则SYMAMD无法继续，打印错误消息，并且不返回输出参数(p或stats)。因此，SYMAMD是一种检查稀疏矩阵是否有效的简单方法。

stats(4:7)提供了SYMAMD是否能够继续的信息。如果stats(4)为零，则矩阵为OK，如果无效则为1。Stats(5)是最右边未排序或包含重复条目的列索引，如果不存在这样的列，则为零。Stats(6)是Stats(5)给出的列索引中最后一次看到的重复或乱序的行索引，如果不存在这样的行索引，则为零。Stats(7)是重复或乱序的行索引的数量。stats(8:20)在SYMAMD的当前版本中始终为零(保留以供将来使用)。

排序之后是列消除树后排序。

代码本身的作者是Stefan I. Larimore和Timothy A. Davis。该算法是与施乐帕洛阿尔托研究中心的约翰·吉尔伯特和橡树岭国家实验室的埃斯蒙德·吴恩达合作开发的。(参见http://faculty.cse.tamu.edu/davis/suitesparse.html)

参见:colperm, colamd。

: p = symrcm (S)

返回S的对称反向Cuthill-McKee排列。

p是一个排列向量，使得S(p, p)的对角线元素趋向于比S更接近对角线。这对于来自“长而细”问题的矩阵的LU或Cholesky分解来说是一个很好的预排序。它对对称S和非对称S都有效。

该算法是求解np完全带宽最小化问题的一种启发式方法。实现基于中找到的描述

E.卡希尔，J.麦基。减少稀疏对称矩阵的带宽。第24届ACM全国会议论文集，157-172 1969年，布兰登出版社，新泽西州。

A.乔治，刘j.w.h.。大型稀疏正定系统的计算机解，计算数学中的Prentice Hall系列，ISBN 0-13-165274- 5,1981。

参见:colperm, colamd, symamd。

**22.2稀疏矩阵上的线性代数**

Octave包含一个稀疏矩阵的多态求解器，其中用于分解矩阵的精确求解器取决于稀疏矩阵本身的性质。一般来说，确定矩阵类型的成本相对于分解矩阵本身的成本来说是很小的，但是在任何情况下，一旦计算出矩阵类型，就会缓存它，这样就不必每次在线性方程中使用它时都重新确定它。

线性方程解的选择树是

1. 如果矩阵是对角线的，直接解到8
2. 如果矩阵是一个排列对角线，直接求解考虑到排列。转到8
3. 如果矩阵是方形的，带状的，如果带密度小于spparms给出的(“bandden”)，继续，否则转到4。
4. 如果矩阵是三对角线的，并且右边不是稀疏的，继续，否则是3b。
5. 如果矩阵是厄米矩阵，具有正实对角线，则尝试使用LAPACK xPTSV进行Cholesky分解。
6. 如果上述方法失败或矩阵不是具有正实对角线的厄米矩阵，则使用使用LAPACK xGTSV进行高斯消去和旋转，并转到8。
7. 如果矩阵是具有正实对角线的厄米矩阵，则尝试使用LAPACK xPBTRF进行Cholesky分解。
8. 如果上述方法失败或矩阵不是具有正实对角线的厄米矩阵，则使用高斯消去并使用LAPACK xGBTRF进行旋转，并转到8。
   1. 如果矩阵是上三角形或下三角形，则执行稀疏的正向或向后替换，并转到8
   2. 如果矩阵是具有列排列的上三角矩阵或具有行排列的下三角矩阵，则执行稀疏的向前或向后替换，并转到8
   3. 如果矩阵是正方形的，厄密矩阵具有正对角线，尝试使用CHOLMOD进行稀疏Cholesky分解。
   4. 如果稀疏Cholesky分解失败或矩阵不是具有实正对角线的hermite，并且矩阵是平方的，则使用UMFPACK进行分解，求解并执行一次细化迭代。
   5. 如果矩阵不是平方的，或者前面的任何求解器标记了一个奇异或接近奇异的矩阵，那么使用CXSPARSE10找到最小范数解。

频带密度定义为频带内非零值的个数除以整个频带内值的总数。通过使用spparms将bandden设置为1(即spparms(“bandden”，1))，可以完全禁用带状矩阵求解器。

QR求解器使用Dulmage-Mendelsohn分解来分解问题，将问题分解为可被视为过度确定的块、多个良好确定的块和最终的过度确定块。对于具有强连接节点块的矩阵，这是一个巨大的胜利，因为LU分解可以用于许多块。它还显著提高了为过度确定的问题找到解决方案的机会，而不仅仅是返回NaN的向量。

以上所有的求解器，都可以计算出条件数的估计值。这可用于检测解中的数值稳定性问题，并强制使用最小范数解。然而，对于窄带状、三角形或对角矩阵，计算条件数的成本是显著的，并且实际上可能超过对矩阵进行因式分解的成本。因此，在这些情况下不计算条件数，并且Octave依赖于更简单的技术来检测奇异矩阵或在带状矩阵的情况下的底层LAPACK代码。

用户可以使用matrix\_type函数强制设置矩阵的类型。这克服了发现矩阵类型的代价。但是，应该注意的是，不正确地识别矩阵的类型将导致不可预测的结果，因此应该谨慎使用matrix\_type。

: nest = normest (A)

: nest = normest (A, tol)

: [nest, iter] = normest (…)

用幂级数分析估计矩阵A的2范数。

这通常用于大型矩阵，其中计算范数(A)的成本令人望而却步，并且近似于2范数是可以接受的。

Tol是计算2范数的公差。缺省值为1e-6。

可选输出iter返回normest收敛所需的迭代次数。

参见:normst1, norm, cond, condest。

: nest = normest1 (A)

: nest = normest1 (A, t)

: nest = normest1 (A, t, x0)

: nest = normest1 (Afcn, t, x0, p1, p2, …)

: [nest, v] = normest1 (A, …)

: [nest, v, w] = normest1 (A, …)

: [nest, v, w, iter] = normest1 (A, …)

用块算法估计矩阵A的1范数。

Normest1最适合只需要范数估计的大型稀疏矩阵。对于中小型矩阵，考虑使用norm (A, 1)。此外，normst1可以用于估计线性算子A的1-范数，当矩阵向量乘积A \* x和A' \* x可以便宜地计算时。在这种情况下，使用的不是矩阵A，而是函数Afcn (flag, x);它必须返回:

* the dimension n of A, if flag is "dim"
* true if A is a real operator, if flag is "real"
* the result A \* x, if flag is "notransp"
* the result A' \* x, if flag is "transp"

典型的例子是由b ^ m定义的A，其中结果A \* x可以通过以下方式计算，甚至不需要显式地形成b ^ m:

y = x;

for i = 1:m

y = b \* y;

endfor

形参p1, p2，…是Afcn (flag, x, p1, p2，…)的实参。

t的默认值是2。该算法需要大小为n × n和n × t的矩阵-矩阵乘积。

初始矩阵x0应该有单位1范数的列。默认初始矩阵x0的第一列是1 (n, 1) / n，如果t > 1，其余的列是随机元素-1 / n, 1 / n，除以n。

在输出中，nest是期望的估计，v和w是使w = A \* v的向量，其中norm (w, 1) = c \* norm (v, 1)。iter在iter(1)中包含迭代次数(最大值硬编码为5)，在iter(2)中包含算法执行的乘积A \* x或A' \* x的总数。

算法说明:normst1在求值时使用随机数。因此，如果需要一致的结果，则应该在调用normst1之前固定随机生成器的“状态”。

参考文献:N. J. Higham和F. Tisseur。矩阵1-范数估计的块算法及其在1-范数伪谱中的应用。达成。， 2000年第21卷第4期，第1185-1201页。

参见normest, norm, cond, condest。

: cest = condest (A)

: cest = condest (A, t)

: cest = condest (A, Ainvfcn)

: cest = condest (A, Ainvfcn, t)

: cest = condest (A, Ainvfcn, t, p1, p2, …)

: cest = condest (Afcn, Ainvfcn)

: cest = condest (Afcn, Ainvfcn, t)

: cest = condest (Afcn, Ainvfcn, t, p1, p2, …)

: [cest, v] = condest (…)

使用t检验向量和随机1-范数估计方阵a的1-范数条件数。

可选输入t指定测试向量的数量(默认为5)。

输入可以是矩阵a(该算法特别适用于大型稀疏矩阵)。或者，矩阵的行为可以由函数隐式地定义。当使用隐式定义时，condest需要下列函数:

-Afcn (flag, x) which must return

* the dimension n of A, if flag is "dim"
* true if A is a real operator, if flag is "real"
* the result A \* x, if flag is "notransp"
* the result A' \* x, if flag is "transp"

-Ainvfcn (flag, x) which must return

* the dimension n of inv (A), if flag is "dim"
* true if inv (A) is a real operator, if flag is "real"
* the result inv (A) \* x, if flag is "notransp"
* the result inv (A)' \* x, if flag is "transp"

任何形参p1, p2，…都是Afcn (flag, x, p1, p2，…)和Ainvfcn (flag, x, p1, p2，…)的附加参数。

主要输出是1范数条件估计检验。

可选的第二个输出v是一个近似的零向量;它满足方程norm (A\*v, 1) == norm (A, 1) \* norm (v, 1) / cest。

算法说明:condest使用随机化算法来近似1-norm。因此，如果需要一致的结果，则应该在调用condest之前固定随机生成器的“状态”。

引用:

* nj . highham和F. Tisseur，矩阵1-范数估计的块算法及其在1-范数伪谱中的应用。SIMAX vol 21, no 4, pp 1185-1201。https://dx.doi.org/10.1137/S0895479899356080
* nj . highham和F. Tisseur，矩阵1-范数估计的块算法及其在1-范数伪谱中的应用。https://citeseer.ist.psu.edu/223007.html

参见:cond, rd, norm, normst1, normest。

: spparms ()

: vals = spparms ()

: [keys, vals] = spparms ()

: val = spparms (key)

: spparms (vals)

: spparms ("default")

: spparms ("tight")

: spparms (key, val)

查询或设置稀疏求解器和分解函数使用的参数。

上面的前四个调用获取有关当前设置的信息，而其他调用则更改当前设置。参数以键和值对的形式存储，其中值都是浮点数，键是以下字符串之一:

‘spumoni’

求解器调试信息的打印级别(默认为0)

‘ths\_rel’

包括兼容性。不习惯。(默认为1)

‘ths\_abs’

包括兼容性。不习惯。(默认为1)

‘exact\_d’

包括兼容性。不习惯。0(默认)

‘supernd’

包括兼容性。不习惯。(默认3)

‘rreduce’

包括兼容性。不习惯。(默认3)

‘wh\_frac’

包括兼容性。不习惯。0.5(默认)

‘autommd’

标记LU/QR和' \ '和' / '操作符是否会自动使用保持稀疏性的mmd函数(默认为1)

‘autoamd’

标记LU和' \ '和' / '操作符是否会自动使用保持稀疏性的amd函数(默认为1)

‘piv\_tol’

UMFPACK解算器的枢轴公差(默认0.1)

‘sym\_tol’

UMFPACK对称解算器的枢轴公差(默认为0.001)

‘bandden’

在被LAPACK带状解算器处理之前，带状矩阵中非零元素的密度(默认为0.5)

‘umfpack’

标记UMFPACK或mmd解算器是否用于LU， ' \ '和' / '操作(默认为1)

可以使用spparms (key, val)设置单个键的值。可以使用特殊关键字“default”恢复默认值。可以使用特殊关键字“tight”来设置mmd求解器，以尝试更稀疏的解决方案，但可能会以更长的运行时间为代价。

参见:chol, colamd, lu, qr, symamd。

: p = sprank (S)

计算稀疏矩阵S的结构秩。

请注意，在这个基于Dulmage-Mendelsohn排列到块三角形形式的计算中，只使用矩阵的结构。因此，矩阵S的数值秩以sprank (S) >= rank (S)为界。忽略浮点误差，sprank (S) == rank (S)。

参见:dperm。

: [count, h, parent, post, R] = symbfact (S)

: […] = symbfact (S, typ)

: […] = symbfact (S, typ, mode)

执行稀疏矩阵S的符号分解分析。

输入变量是

S

S是一个实或复稀疏矩阵。

typ

因子分解的类型是否可以是其中之一

"sym" (default)

假设S是对称的，并使用矩阵的上三角部分。

"col"

分解S' \* S。

"row"

分解S \* S'。

"lo"

因式分解年代”。假设S是对称的并且使用矩阵的下三角部分。

mode

当mode未指定时，返回R的Cholesky分解。如果mode为“lower”或“L”，则返回共轭转置R'，这是一个较低的三角形因子。共轭转置版本更快，使用更少的内存，但仍然为所有其他输出返回相同的值:count, h, parent和post。

输出变量为:

count

按类型确定的Cholesky分解的行数。使用chol进行真正因子分解的计算难度是sum (count .^ 2)。

h

消去树的高度。

parent

消去树本身。

post

一种稀疏布尔矩阵，其结构是由类型决定的乔列斯基分解矩阵。

参见:chol, tree, treelayout。

对于非方阵，用户也可以利用spaugment函数来求线性方程的最小二乘解。

: s = spaugment (A, c)

创建A的增广矩阵。

这是由

[c \* eye(m, m), A;

A', zeros(n, n)]

它与A \ b的最小二乘解有关

s \* [ r / c; x] = [ b, zeros(n, columns(b)) ]

r是残差

r = b - A \* x

由于矩阵s是对称不定的，它可以被lu分解，因此可以找到最小范数解，而不需要qr分解。由于残差将为0 (m, m)，对于待定问题，示例可以

m = 11; n = 10; mn = max (m, n);

A = spdiags ([ones(mn,1), 10\*ones(mn,1), -ones(mn,1)],

[-1, 0, 1], m, n);

x0 = A \ ones (m,1);

s = spaugment (A);

[L, U, P, Q] = lu (s);

x1 = Q \* (U \ (L \ (P \* [ones(m,1); zeros(n,1)])));

x1 = x1(end - n + 1 : end);

求过定问题的解需要残差r的估计，因此用spaugment函数来表述最小范数解比较复杂。

一般来说，左除法运算符比使用扩展函数更稳定、更快。

参见:mldivide。

最后，函数eigs可以用来计算基于选择标准的有限数量的特征值和特征向量，同样地，svds可以计算有限数量的奇异值和向量。

: d = eigs (A)

: d = eigs (A, k)

: d = eigs (A, k, sigma)

: d = eigs (A, k, sigma, opts)

: d = eigs (A, B)

: d = eigs (A, B, k)

: d = eigs (A, B, k, sigma)

: d = eigs (A, B, k, sigma, opts)

: d = eigs (Af, n)

: d = eigs (Af, n, k)

: d = eigs (Af, n, k, sigma)

: d = eigs (Af, n, k, sigma, opts)

: d = eigs (Af, n, B)

: d = eigs (Af, n, B, k)

: d = eigs (Af, n, B, k, sigma)

: d = eigs (Af, n, B, k, sigma, opts)

: [V, D] = eigs (…)

: [V, D, flag] = eigs (…)

根据选择标准计算有限数量的特征值和特征向量。

默认情况下，eigs求解方程，其中为相应的特征向量。如果给定正定矩阵B，则eigs解一般特征值方程

输入A是一个n × n维的方阵。通常，A也是大而稀疏的。

广义特征值问题的输入B是一个与a (n × n)大小相同的方阵。通常，B也是大而稀疏的。

要计算的特征值和特征向量的数量由k给出，默认为6。

参数决定返回哪些特征值。可以是标量也可以是字符串。当σ是标量时，返回最接近σ的k个特征值。如果sigma是字符串，则它必须是下列值之一。

"lm"

最大幅度(默认值)。

"sm"

最小的大小。

"la"

最大代数(仅对实对称问题有效)。

"sa"

最小代数(仅适用于实对称问题)。

"be"

两端，如果k是奇数(仅对实对称问题有效)，则从高端再加一个。

"lr"

最大实部(仅适用于复杂或非对称问题)。

"sr"

最小实部(仅适用于复杂或非对称问题)。

"li"

最大虚部(仅对复杂或非对称问题有效)。

"si"

最小虚部(仅适用于复杂或非对称问题)。

如果给出了options，则它是一个定义egg应该使用的可能选项的结构。opts结构的字段是:

issym

如果给出了Af，那么这个标志(true/false)决定函数Af是否定义了一个对称问题。如果矩阵a是给定的，则忽略它。默认为false。

isreal

如果给出了Af，那么这个标志(true/false)决定函数Af是否定义了一个真正的问题。如果矩阵a是给定的，则忽略它。默认为true。

tol

定义所需的收敛容差，以tol \*范数(A)计算。默认值为eps。

maxit

迭代的最大次数。默认值是300。

p

要使用的Lanczos基向量的个数。更多的向量会导致更快的收敛，但会占用更多的内存。p的最优值与问题有关，应该在k + 1到n的范围内。默认值是2 \* k。

v0

算法的起始向量。初始向量接近最终向量将加快收敛速度。默认情况下，ARPACK随机生成一个起始向量。如果指定，v0必须是n × 1向量，其中n =行(A)。

disp

诊断打印输出的级别(0|1|2)。如果disp为0，则禁用诊断。缺省值为0。

cholB

如果正在计算广义特征值问题，则该标志(true/false)指定B输入是代表chol (B)还是仅仅代表矩阵B。默认为false。

permB

若cholB为真，则B的Cholesky分解的排列向量。由[R， ~， permB] = chol (B，“向量”)得到。缺省值为1:n。

也可以用一个表示为Af的函数来表示A。Af必须后跟一个标量参数n，该参数定义了Af接受的向量参数的长度。Af可以是函数句柄、内联函数或字符串。当Af是字符串时，它保存要使用的函数的名称。

Af是一个形式为y = Af (x)的函数，其中所需的Af返回值由sigma的值决定。四种可能的形式是

A \* x

如果sigma没有给出或者是除“sm”以外的字符串。

A \ x

如果是0或sm。

(A - sigma \* I) \ x

如果是一个不等于0的标量;I是和A一样大小的单位矩阵。

(A - sigma \* B) \ x

对于一般特征值问题。

返回参数及其形式取决于所请求的返回参数的数量。对于单个返回参数，返回长度为k的列向量d，其中包含已找到的k个特征值。对于两个返回参数，V是一个n × k矩阵，它的列是对应于返回的特征值的k个特征向量。特征值本身在D中以k × k矩阵的形式返回，其中对角线上的元素是特征值。

第三个返回参数标志返回收敛的状态。如果标志为0，则所有特征值都收敛。任何其他值表示收敛失败。

编程注意事项:对于n < 500的小问题，考虑使用eig (full (A))。

如果ARPACK不能收敛，可以考虑增加Lanczos向量的数量(opt.p)，增加迭代次数(opt.maxiter)，或者减少容差(opt.tol)。

参考:该函数基于ARPACK包，由R. Lehoucq, K. Maschhoff, D. Sorensen和C. Yang编写。欲了解更多信息，请参阅http://www.caam.rice.edu/software/ARPACK/。

参见: eig,svds。

: s = svds (A)

: s = svds (A, k)

: s = svds (A, k, sigma)

: s = svds (A, k, sigma, opts)

: [u, s, v] = svds (…)

: [u, s, v, flag] = svds (…)

找出矩阵a的几个奇异值。

奇异值的计算使用

[m, n] = size (A);

s = eigs ([sparse(m, m), A;

A', sparse(n, n)])

eg返回的特征值对应于a的奇异值。要计算的奇异值的个数由k给出，默认为6。

参数sigma指定要查找的奇异值。当sigma是字符串'L'时，默认情况下，找到A的最大奇异值。否则，必须是一个实标量，并且找到最接近的奇异值。作为推论，sigma = 0找到最小的奇异值。请注意，对于相对较小的sigma值，有可能找不到所请求的奇异值的数量。在这种情况下应该增加。

Opts是一个定义SVDS将传递给egg的选项的结构。该结构的可能字段以egg形式记录。缺省情况下，svds设置了以下三个字段:

tol

奇异值所要求的收敛容限。默认值为1e-10。Eigs通过了l / SQRT(2)。

maxit

迭代的最大次数。默认值是300。

disp

诊断打印输出的级别(0|1|2)。如果disp为0，则禁用诊断。缺省值为0。

如果请求多个输出，那么svds将返回A的奇异值分解的近似值

A\_approx = u\*s\*v'

其中a\_ax是一个大小为a但秩为k的矩阵。

如果算法成功收敛，Flag返回0，否则返回1。收敛性的检验是

norm (A\*v - u\*s, 1) <= tol \* norm (A, 1)

SVDS最适合从一个大的稀疏矩阵中只找到几个奇异值。否则，svd (full (A))可能会更有效。

参见:svd, eigs。

**22.3迭代技术在稀疏矩阵中的应用**

前一节讨论的左除法\和右除法/运算符使用直接求解器来求解形式为x = a \ b或x = b / a的线性方程。Octave还包括一些使用迭代技术求解稀疏线性方程的函数。

: x = pcg (A, b, tol, maxit, m1, m2, x0, …)

: x = pcg (A, b, tol, maxit, M, [], x0, …)

: [x, flag, relres, iter, resvec, eigest] = pcg (A, b, …)

用预条件共轭梯度迭代法求解线性方程组A \* x = b。

输入参数是:

* A是线性系统的矩阵它必须是平方的。A可以作为矩阵、函数句柄或内联函数Afcn传递，这样Afcn(x) = A \* x。Afcn的附加参数可以在x0之后传递。

A必须是厄米正定(HPD)如果pcg检测到A不是正确定的，则打印警告并设置标志输出。

* B是右边的向量。
* tol是残余误差b - A \* x所需的相对容差。如果norm (b - A \* x)≤tol \* norm (b)，则迭代停止。如果tol省略或为空，则使用1e-6的容差。
* Maxit是允许的最大迭代次数;如果maxit省略或为空，则使用值20。
* m为HPD预处理矩阵。对于任意分解m = p1 \* p2，使得inv (p1) \* A \* inv (p2)为HPD，将共轭梯度法正式应用于线性系统inv (p1) \* A \* inv (p2) \* y = inv (p1) \* b, x = inv (p2) \* y(分裂预处理)。在实际应用中，共轭梯度法的每次迭代都是用mldivide法求解矩阵为m的线性系统。如果一个特定的因式分解m = m1 \* m2是可用的(例如，a的不完全Cholesky因式分解)，则可以传递两个矩阵m1和m2，并使用mldivide算子求解相对的线性系统。请注意，正确选择前置条件可能会显著提高该方法的整体性能。用户可以传递两个函数，而不是矩阵m1和m2，它们返回将m1和m2的逆应用于向量的结果。如果m1省略或为空[]，则不应用预处理。如果无法对m进行因式分解，则可以省略或保留m2[]，使用输入变量m1传递预条件m。
* X0是最初的猜测。如果x0省略或为空，则该函数默认将x0设置为零向量。

x0后面的参数被视为参数，并以适当的方式传递给已给定给pcg的任何函数(A或m1或m2)。有关更多细节，请参阅下面的示例。

输出参数是:

* x为A \* x = b解的计算近似值，如果算法不收敛，则x为残差最小的迭代。
* 关于收敛的标志报告:
* 0:算法收敛到规定公差内。
* 1:算法不收敛，达到最大迭代次数。
* 2:预条件矩阵是奇异的。
* 3:算法停滞，即当前迭代x与前一次迭代差值的绝对值小于eps \* norm (x,2)。
* 4:算法检测到输入(预置)矩阵不是HPD。
* relres是最终残差与其初始值的比值，用欧几里得范数表示。
* Iter表示它被计算的x的迭代。由于输出x对应于最小剩余解，因此该方法执行的迭代总数由length(resvec) - 1给出。
* Resvec描述了该方法的收敛历史。resvec (i, 1)为残差的欧氏范数，resvec (i, 2)为预条件残差范数，经过(i-1)次迭代后，i = 1,2，…，iter+1。预条件残差范数定义为r' \* (m \ r)，其中r = b - A \* x，参见m的描述。如果不需要eigest，则只返回resvec(:， 1)。
* eigest返回预条件矩阵P = m \ A的最小(1)和最大(2)特征值的估计。特别地，如果不使用预条件，则返回A的极值特征值的估计。eigest(1)为高估值，eigest(2)为低估值，因此eigest(2) / eigest(1)为cond (P, 2)的下界，但在极限情况下理论上应等于条件数的实际值。

让我们考虑一个关于三对角矩阵的平凡问题

n = 10;

A = toeplitz (sparse ([1, 1], [1, 2], [2, 1], 1, n));

b = A \* ones (n, 1);

M1 = ichol (A); # in this tridiagonal case it corresponds to chol (A)'

M2 = M1';

M = M1 \* M2;

Afcn = @(x) A \* x;

Mfcn = @(x) M \ x;

M1fcn = @(x) M1 \ x;

M2fcn = @(x) M2 \ x;

例1:pcg最简单的用法

x = pcg (A, b)

例2:pcg带有一个计算a \* x的函数

x = pcg (Afcn, b)

例3:pcg带有一个预条件矩阵M

x = pcg (A, b, A -06, 100, M)

例4:用函数作为前置条件的pcg

x = pcg (Afcn, b, 1e-6, 100, Mfcn)

例5:pcg与预条件矩阵M1和M2

x = pcg (A, b, 1e-6, 100, M1, M2)

例6:pcg具有前置功能

x = pcg (Afcn, b, 1e-6, 100, M1fcn, M2fcn)

例7:pcg输入一个需要参数的函数

function y = Ap (A, x, p) # compute A^p \* x

y = x;

for i = 1:p

y = A \* y;

endfor

endfunction

Apfcn = @(x, p) Ap (A, x, p);

x = pcg (Apfcn, b, [], [], [], [], [], 2);

例8:显示pcg使用分割前置条件的显式示例

M1 = ichol (A + 0.1 \* eye (n)); # factorization of A perturbed

M2 = M1';

M = M1 \* M2;

## reference solution computed by pcg after two iterations

[x\_ref, fl] = pcg (A, b, [], 2, M)

## split preconditioning

[y, fl] = pcg ((M1 \ A) / M2, M1 \ b, [], 2)

x = M2 \ y # compare x and x\_ref

引用:

1. C.T. Kelley，线性和非线性方程的迭代方法，计算机工程，1995。(基本PCG算法)
2. 李志强，稀疏线性系统的迭代方法，计算机工程学报，1996。(从PCG的条件数字估计)这本书的修订版可在https://www-users.cs.umn.edu/~saad/books.html网上找到

参见:sparse, pcr, gmres, bicg, bigstab, cgs。

: x = pcr (A, b, tol, maxit, m, x0, …)

: [x, flag, relres, iter, resvec] = pcr (…)

用预条件共轭残差迭代法求解线性方程组A \* x = b。

输入参数是

* A可以是正方形矩阵(最好是稀疏矩阵)，也可以是函数句柄、内联函数或包含计算A \* x的函数名称的字符串。原则上，A应该是对称且非奇异的;如果pcr发现A在数字上是单数，您将得到一条警告消息，并设置标志输出参数。
* B是右边的向量。
* tol是残余误差b - A \* x所需的相对容差。如果norm (b - A \* x) <= tol \* norm (b - A \* x0)，迭代停止。如果tol为空或省略，则函数默认设置tol = 1e-6。
* Maxit是允许的最大迭代次数;如果为maxit提供了[]，或者PCR的参数较少，则使用默认值为20。
* m是(左)预处理矩阵，因此迭代(理论上)相当于用pcr求解P \* x = m \ b，其中P = m \ a。注意，正确选择预条件可能会显著提高该方法的整体性能。用户可以传递一个函数，而不是矩阵m，该函数返回将m的逆应用于向量的结果(通常这是使用前置条件的首选方式)。如果为m提供[]，或者省略m，则不应用预处理。
* X0是最初的猜测。如果x0为空或省略，则该函数默认将x0设置为零向量。

x0后面的参数被视为参数，并以适当的方式传递给传递给pcr的任何函数(a或m)。有关更多细节，请参阅下面的示例。

输出参数是

* x是A \* x = b解的计算近似值。
* Flag报告收敛情况。Flag = 0表示解收敛，满足tol给出的公差准则。Flag = 1表示达到了迭代计数的最大限制。Flag = 3表示PCR故障，详见[1]。
* relres是最终残差与其初始值的比值，用欧几里得范数表示。
* Iter是执行迭代的实际次数。
* resvec描述了方法的收敛历史，使得resvec (i)包含了(i-1)次迭代后残差的欧氏范数，i = 1,2，…，iter+1。

让我们考虑一个关于对角矩阵的平凡问题(我们利用a的稀疏性)

n = 10;

A = sparse (diag (1:n));

b = rand (N, 1);

例1:pcr的最简单应用

x = pcr (A, b)

例2:pcr与计算a \* x的函数。

function y = apply\_a (x)

y = [1:10]' .\* x;

endfunction

x = pcr ("apply\_a", b)

例3:预置迭代，具有完整的诊断。预条件(很奇怪，因为即使是原始矩阵A也是平凡的)被定义为一个函数

function y = apply\_m (x)

k = floor (length (x) - 2);

y = x;

y(1:k) = x(1:k) ./ [1:k]';

endfunction

[x, flag, relres, iter, resvec] = ...

pcr (A, b, [], [], "apply\_m")

semilogy ([1:iter+1], resvec);

例4:最后，一个依赖于参数k的预条件。

function y = apply\_m (x, varargin)

k = varargin{1};

y = x;

y(1:k) = x(1:k) ./ [1:k]';

endfunction

[x, flag, relres, iter, resvec] = ...

pcr (A, b, [], [], "apply\_m"', [], 3)

参考:

W. Hackbusch，大型稀疏方程组的迭代解，第9.5.4节;施普林格1994

参见:稀疏，pcg。

迭代求解器收敛到解的速度可以通过使用预处理矩阵M来加快。在这种情况下，线性方程M^-1 \* x = M^-1 \* a \ b被求解。典型的预处理矩阵是原始矩阵的部分分解。

: L = ichol (A)

: L = ichol (A, opts)

计算稀疏方阵A的不完全Cholesky分解。

默认情况下，ichol只使用A的下三角形，并产生一个下三角形因子L，使L\*L'近似于A。

该例程给出的因子可作为用PCG(预条件共轭梯度)等迭代方法求解的线性方程组的预条件。

可以通过在结构options中传递选项来修改分解。选项名是结构的一个字段，设置是field的值。名称和说明符区分大小写。

类型

因子分解的类型。

"nofill" (default)

无填充的不完全Cholesky分解(IC(0))。

"ict"

阈值下降的不完全Cholesky分解。

diagcomp

A的不完全Cholesky分解的非负标量α + α \* diag (diag (A))代替A。这在A不是正定时很有用。缺省值为0。

droptol

一个非负的标量，指定在执行ICT时因式分解的容差。默认值为0，生成完整的Cholesky分解。

L的非对角元素被设为0，除非

abs (L(i,j)) >= droptol \* norm (A(j:end, j), 1).

michol

改进的不完全Cholesky分解:

"off" (default)

行和列和不一定保留。

"on"

修改了L的对角线，以便即使在分解过程中删除了元素，也保留行(和列)和。保留的关系是:A \* e = L \* L' \* e，其中e是1的向量。

shape

"lower" (default)

只使用A的下三角形并返回下三角形因子L，使得L\*L'近似于A。

"upper"

只使用A的上三角形并返回上三角形因子U，使得U'\*U近似于A。

例子

下面的问题演示了如何用完全Cholesky分解和不完全Cholesky分解分解一个样本对称正定矩阵。

A = [ 0.37, -0.05, -0.05, -0.07;

-0.05, 0.116, 0.0, -0.05;

-0.05, 0.0, 0.116, -0.05;

-0.07, -0.05, -0.05, 0.202];

A = sparse (A);

nnz (tril (A))

ans = 9

L = chol (A, "lower");

nnz (L)

ans = 10

norm (A - L \* L', "fro") / norm (A, "fro")

ans = 1.1993e-16

opts.type = "nofill";

L = ichol (A, opts);

nnz (L)

ans = 9

norm (A - L \* L', "fro") / norm (A, "fro")

ans = 0.019736

分解的另一个例子是用有限差分矩阵来解决单位方框上的边值问题。

nx = 400; ny = 200;

hx = 1 / (nx + 1); hy = 1 / (ny + 1);

Dxx = spdiags ([ones(nx, 1), -2\*ones(nx, 1), ones(nx, 1)],

[-1 0 1 ], nx, nx) / (hx ^ 2);

Dyy = spdiags ([ones(ny, 1), -2\*ones(ny, 1), ones(ny, 1)],

[-1 0 1 ], ny, ny) / (hy ^ 2);

A = -kron (Dxx, speye (ny)) - kron (speye (nx), Dyy);

nnz (tril (A))

ans = 239400

opts.type = "nofill";

L = ichol (A, opts);

nnz (tril (A))

ans = 239400

norm (A - L \* L', "fro") / norm (A, "fro")

ans = 0.062327

实现算法参考:

[1]王晓明。“预处理技术。”稀疏线性系统的迭代方法，PWS出版社，1996。

[2]张晓明，张晓明。一种改进的不完全Cholesky分解方法，2007。

参见:chol, ilu, pcg。

: LUA = ilu (A)

: LUA = ilu (A, opts)

: [L, U] = ilu (…)

: [L, U, P] = ilu (…)

计算稀疏方阵A的不完全LU分解。

ilu返回一个单位下三角矩阵L，一个上三角矩阵U，以及一个可选的置换矩阵P，使得L\*U近似于P\* a。

该例程给出的因子可以作为用BICG(双共轭梯度)或GMRES(广义最小残差法)等迭代方法求解线性方程组的前置条件。

可以通过在结构options中传递选项来修改分解。选项名是结构的一个字段，设置是field的值。名称和说明符区分大小写。

type

分解类型。

"nofill" (default)

没有填充的ILU分解(ILU(0))。

其他支持的选项:milu。

"crout"

崩溃版本的ILU分解(ILUC)。

其他支持的选项:milu, droptool。

"ilutp"

带有阈值和枢轴的ILU分解。

其他支持的选项:milu, droptool, using, thresh。

droptol

一个非负标量，指定因式分解的掉落公差。默认值为0，生成完整的LU分解。

U的非对角元素被设为0，除非

abs (U(i,j)) >= droptol \* norm (A(:,j)).

L的非对角元素被设为0，除非

abs (L(i,j)) >= droptol \* norm (A(:,j))/U(j,j).

milu

改进的不完全LU分解:

"row"

行和修正的不完全LU分解。因式分解保留行和:A \* e = L \* U \* e，其中e是1的向量。

"col"

列和修正的不完全LU分解。因式分解保留列和:e' \* A = e' \* L \* U。

"off" (default)

行和列和不一定保留。

udiag

如果为真，则上三角因子对角线上的任何零都被替换为局部掉落公差droptol \*范数(A(:，j))/U(j,j)。默认为false。

thresh

分解的枢轴阈值。它的范围可以在0(对角线旋转)和1(默认)之间，其中选择列中的最大大小条目作为枢轴。

如果只使用一个输出调用ilu，则返回的矩阵是L + U - speye (size (A))，其中L是单位下三角形，U是上三角形。

对于两个输出，ilu返回一个单位下三角矩阵L和一个上三角矩阵u。键入== "ilutp"，其中一个因素根据options .milu的值进行排列。当options .milu == "row"时，U是列排列的上三角因子。否则，L是一个按行排列的单位下三角因子。

如果有三个已命名的输出并且options .milu != "row"，则返回P，使得L和U是P\*A的不完全因子。当options .milu == "row"时，返回P，使得L和U是A\*P的不完全因子。

示例

A = gallery ("neumann", 1600) + speye (1600);

opts.type = "nofill";

nnz (A)

ans = 7840

nnz (lu (A))

ans = 126478

nnz (ilu (A, opts))

ans = 7840

这表明，A有7840个非零，完全LU分解有126478个非零，不完全LU分解有7840个非零，填充级别为0，与A相同。摘自:https://www.mathworks.com/help/matlab/ref/ilu.html

A = gallery ("wathen", 10, 10);

b = sum (A, 2);

tol = 1e-8;

maxit = 50;

opts.type = "crout";

opts.droptol = 1e-4;

[L, U] = ilu (A, opts);

x = bicg (A, b, tol, maxit, L, U);

norm (A \* x - b, inf)

本例使用ILU作为条件数较大的随机有限元矩阵的预条件。如果没有L和U, BICG就不会收敛。

参见:lu, ichol, bicg, gmres。

**22.4使用稀疏矩阵的现实生活示例**

稀疏矩阵的一个常见应用是求解有限元模型。有限元模型允许不具有封闭形式解的偏微分方程的数值解，通常是因为域的复杂形状。

为了激发这种应用，我们考虑了边值拉普拉斯方程。该系统可以模拟标量势场，如热势或电势。给定一个中值，边界是。在所有点上边界条件都是已知的，我们希望计算的势。边界条件可以指定势(狄利克雷边界条件)，其越过边界的法向导数(诺伊曼边界条件)，或势及其导数的加权和(柯西边界条件)。

在热模型中，我们想要计算温度，并知道边界温度(狄利克雷条件)或热流密度(从中我们可以通过除以边界的热导率来计算诺伊曼条件)。同样，在电模型中，我们想计算ω中的电压，并知道边界电压(狄利克雷)或电流(通过电导率潜水后的诺伊曼条件)。在电模型中，边界的大部分通常是电隔离的;这是电流为零的诺伊曼边界条件。

最简单的有限元模型会将Omega分成简单体(2D的三角形，3D的金字塔)。我们以EIDORS项目中的一个装有小不导电球的圆柱形储液罐为例11。该模型旨在反映电阻抗层析成像的应用，将电流模式应用于这样的水箱，以便对内部电导率分布进行成像。为了描述有限元几何结构，我们有一个由顶点、节点和简单元素组成的矩阵。

下面的例子创建了一个简单的矩形二维导电介质，两侧分别施加10v和20v(狄利克雷边界条件)。所有其他边缘都是电隔离的。

node\_y = [1;1.2;1.5;1.8;2]\*ones(1,11);

node\_x = ones(5,1)\*[1,1.05,1.1,1.2, ...

1.3,1.5,1.7,1.8,1.9,1.95,2];

nodes = [node\_x(:), node\_y(:)];

[h,w] = size (node\_x);

elems = [];

for idx = 1:w-1

widx = (idx-1)\*h;

elems = [elems; ...

widx+[(1:h-1);(2:h);h+(1:h-1)]'; ...

widx+[(2:h);h+(2:h);h+(1:h-1)]' ];

endfor

E = size (elems,1); # No. of simplices

N = size (nodes,1); # No. of vertices

D = size (elems,2); # dimensions+1

这将创建一个n × 2矩阵节点和一个带有值的e × 3矩阵元素，它们定义了有限元三角形:

nodes(1:7,:)'

1.00 1.00 1.00 1.00 1.00 1.05 1.05 …

1.00 1.20 1.50 1.80 2.00 1.00 1.20 …

elems(1:7,:)'

1 2 3 4 2 3 4 …

2 3 4 5 7 8 9 …

6 7 8 9 6 7 8 …

使用一阶有限元法，我们将ω中的电导率分布近似为每个单纯形上的常数(由矢量电导率表示)。基于有限元几何，我们首先计算每个单纯形的系统(或刚度)矩阵(在元素系统矩阵SE的对角线上表示为3 × 3元素)。基于SE和n × de连接矩阵C(表示简单点与顶点之间的连接)，计算全局连接矩阵S。

## Element conductivity

conductivity = [1\*ones(1,16), ...

2\*ones(1,48), 1\*ones(1,16)];

## Connectivity matrix

C = sparse ((1:D\*E), reshape (elems', ...

D\*E, 1), 1, D\*E, N);

## Calculate system matrix

Siidx = floor ([0:D\*E-1]'/D) \* D \* ...

ones(1,D) + ones(D\*E,1)\*(1:D) ;

Sjidx = [1:D\*E]'\*ones (1,D);

Sdata = zeros (D\*E,D);

dfact = factorial (D-1);

for j = 1:E

a = inv ([ones(D,1), ...

nodes(elems(j,:), :)]);

const = conductivity(j) \* 2 / ...

dfact / abs (det (a));

Sdata(D\*(j-1)+(1:D),:) = const \* ...

a(2:D,:)' \* a(2:D,:);

endfor

## Element-wise system matrix

SE = sparse(Siidx,Sjidx,Sdata);

## Global system matrix

S = C'\* SE \*C;

系统矩阵的作用类似于欧姆定律S \* V = i中的电导率S。基于狄利克雷和诺伊曼边界条件，我们能够求解每个顶点V处的电压。

## Dirichlet boundary conditions

D\_nodes = [1:5, 51:55];

D\_value = [10\*ones(1,5), 20\*ones(1,5)];

V = zeros (N,1);

V(D\_nodes) = D\_value;

idx = 1:N; # vertices without Dirichlet

# boundary condns

idx(D\_nodes) = [];

## Neumann boundary conditions. Note that

## N\_value must be normalized by the

## boundary length and element conductivity

N\_nodes = [];

N\_value = [];

Q = zeros (N,1);

Q(N\_nodes) = N\_value;

V(idx) = S(idx,idx) \ ( Q(idx) - ...

S(idx,D\_nodes) \* V(D\_nodes));

最后，为了显示解，我们在z轴上为每个单纯形顶点显示每个解出的电压值。如图22.6所示。

elemx = elems(:,[1,2,3,1])';

xelems = reshape (nodes(elemx, 1), 4, E);

yelems = reshape (nodes(elemx, 2), 4, E);

velems = reshape (V(elemx), 4, E);

plot3 (xelems,yelems,velems,"k");

print "grid.eps";

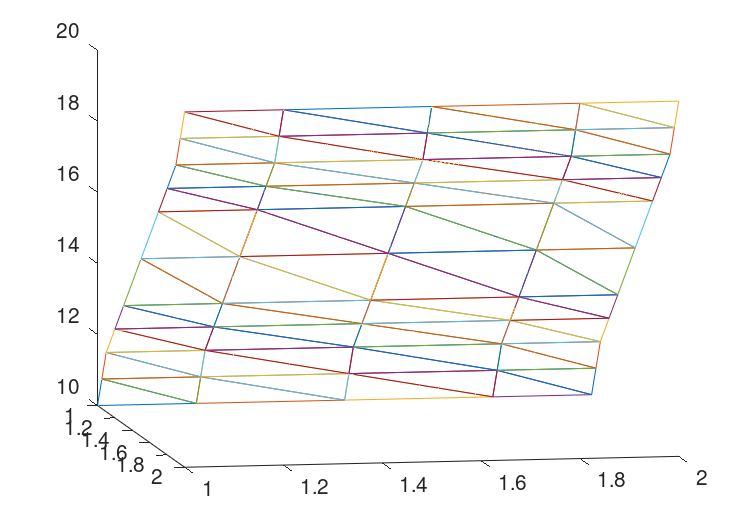


图22.6:显示三角形元素的有限元模型示例。每个顶点的高度对应于解值。

**23数值积分**

Octave带有几个内置函数，用于数值计算函数的积分(称为正交)。这些函数都解决一维积分问题。

**23.1单变量函数**

Octave支持五种不同的自适应正交算法，用于计算函数f在从a到b的区间内的积分

quad

基于高斯正交的数值积分。

quadv

使用自适应矢量化辛普森规则的数值积分。

quadl

使用自适应Lobatto规则的数值积分。

quadgk

采用自适应高斯-康罗德规则的数值积分。

quadcc

基于自适应克伦肖-柯蒂斯规则的数值积分。

此外，还提供以下功能:

integral

一个兼容性包装器函数，它将根据所选的被积函数和选项在quadv和quadgk之间进行选择。

trapz, cumtrapz

用梯形法对数据进行数值积分。

最好的正交算法取决于被积函数。如果您有经验数据，而不是函数，则选择trapz或cumtrapz。如果你不确定被积函数的特性，那么quadcc将是最鲁棒的，因为它可以处理不连续、奇点、振荡函数和无限区间。当被积函数为光滑时，quadgk可能是最快的算法。

|  | **Function** | **Characteristics** |
| --- | --- | --- |
|  | quad | Low accuracy with nonsmooth integrands |
|  | quadv | Medium accuracy with smooth integrands |
|  | quadl | Medium accuracy with smooth integrands. Slower than quadgk. |
|  | quadgk | Medium accuracy (1e-6 – 1e-9) with smooth integrands. |
|  |  | Handles oscillatory functions and infinite bounds |
|  | quadcc | Low to High accuracy with nonsmooth/smooth integrands |
|  |  | Handles oscillatory functions, singularities, and infinite bounds |

这是一个使用四轴函数对函数进行积分的例子

f(x) = x \* sin (1/x) \* sqrt (abs (1 - x))

从x = 0到x = 3。

这是一个相当困难的积分(在积分范围内绘制函数来了解原因)。

第一步是定义函数:

function y = f (x)

y = x .\* sin (1./x) .\* sqrt (abs (1 - x));

endfunction

注意操作符的“点”形式的使用。这对于四积分器来说不是必需的，但是对于其他积分器来说是必需的。在任何情况下，它都使生成一组用于绘图的点变得容易得多，因为可以调用带有矢量参数的函数来生成矢量结果。

第二步是调用具有积分限制的四边形:

[q, ier, nfun, err] = quad ("f", 0, 3)

⇒ 1.9819

⇒ 1

⇒ 5061

⇒ 1.1522e-07

虽然quad返回ier的非零值，但结果相当准确(要了解原因，请检查如果将下界移动到0.1，然后是0.01，然后是0.001，等等，结果会发生什么)。

函数"f"可以是函数的字符串名，也可以是函数句柄。这些选项使集成变得非常容易，而不必在m文件中完全定义函数。例如:

# Verify gamma function = (n-1)! for n = 4

f = @(x) x.^3 .\* exp (-x);

quadcc (f, 0, Inf)

⇒ 6.0000

: q = quad (f, a, b)

: q = quad (f, a, b, tol)

: q = quad (f, a, b, tol, sing)

: [q, ier, nfev, err] = quad (…)

使用QUADPACK中的Fortran例程对f从a到b的积分进行数值计算。

F是一个函数句柄、内联函数或包含要求值的函数名的字符串。函数的形式必须是y = f (x)其中y和x是标量。

A和b分别是积分的下限和上限。其中一个或两个都可以是无限的。

可选参数tol是一个向量，用于指定所需的结果精度。向量的第一个元素是期望的绝对公差，第二个元素是期望的相对公差。若要仅选择相对测试，请将绝对公差设置为零。若要仅选择绝对测试，请将相对容差设置为零。这两个公差默认为sqrt (eps)或大约1.5e-8。

可选参数sing是一个已知被积函数为奇异值的值向量。

积分的结果在q中返回。

Ier包含一个整数错误码(0表示集成成功)。

Nfev表示进行的函数求值的次数。

Err包含解决方案中误差的估计值。

函数quad\_options可以为quad设置其他可选参数。

注意:因为quad是用Fortran编写的，所以它不能被递归调用。这防止了它在通过例程dblquad和triplequad对多个变量进行积分时的使用。

参见:quad\_options, quadv, quadl, quadgk, quadcc, trapz, dblquad, triplequad。

: quad\_options ()

: val = quad\_options (opt)

: quad\_options (opt, val)

查询或设置函数组的选项。

当不带参数调用时，将显示所有可用选项的名称及其当前值。

给定一个参数，返回选项opt的值。

当带两个参数调用时，quad\_options将选项opt设置为value val。

选项包括

"absolute tolerance"

绝对的宽容;对于纯相对误差测试，可能为零。

"relative tolerance"

非负相对容忍度。如果绝对公差为零，则相对公差必须大于或等于max (50\*eps, 0.5e-28)。

"single precision absolute tolerance"

单精度绝对公差;对于纯相对误差测试，可能为零。

"single precision relative tolerance"

单精度的非负相对公差。如果绝对公差为零，则相对公差必须大于或等于max (50\*eps, 0.5e-28)。

: q = quadv (f, a, b)

: q = quadv (f, a, b, tol)

: q = quadv (f, a, b, tol, trace)

: q = quadv (f, a, b, tol, trace, p1, p2, …)

: [q, nfev] = quadv (…)

用自适应辛普森法则计算f从a到b的积分。

F是一个函数句柄、内联函数或包含要求值的函数名的字符串。Quadv是quad的矢量化版本，由f定义的函数必须接受标量或向量作为输入，并返回标量、向量或数组作为输出。

A和b分别是积分的下限和上限。两个极限都必须是有限的。

可选参数tol定义用于停止适应过程的绝对容忍度。默认值为1e-6。

quadv的算法是递归细分积分区间，并对每个子区间应用辛普森规则。如果trace为真，则在计算这些部分积分后显示:(1)函数评估的总数，(2)子区间的左端，(3)子区间的长度，(4)子区间上积分的近似值。

额外的参数p1等直接传递给函数f。要使用tol和trace的默认值，可以传递空矩阵([])。

积分的结果在q中返回。

可选输出nfev表示执行的函数求值的总数。

注意:quadv是用Octave的脚本语言编写的，可以在dblquad和triplequad中递归使用，不像quad函数。

参见:quad, quadl, quadgk, quadcc, trapz, dblquad, triplequad, integral, integral2, integral3。

: q = quadl (f, a, b)

: q = quadl (f, a, b, tol)

: q = quadl (f, a, b, tol, trace)

: q = quadl (f, a, b, tol, trace, p1, p2, …)

: [q, nfev] = quadl (…)

利用自适应Lobatto规则对f从a到b的积分进行数值计算。

F是一个函数句柄、内联函数或包含要求值的函数名的字符串。函数f必须被向量化，当给定一个输入值的向量时，它返回一个输出值的向量。

A和b分别是积分的下限和上限。两个极限都必须是有限的。

可选参数tol定义执行积分的绝对容忍度。默认值为1e-6。

quadl使用的算法是递归细分积分区间。如果定义了trace，则对于每个子区间显示:(1)函数求值的总次数，(2)子区间的左端，(3)子区间的长度，(4)子区间上的积分的近似。

额外的参数p1等直接传递给函数f。要使用tol和trace的默认值，可以传递空矩阵([])。

积分的结果在q中返回。

可选输出nfev表示执行的函数求值的总数。

参考文献:W. Gander和W. Gautschi，《自适应正交——重访》，BIT Vol. 40, No. 1, 2000年3月，第84-101页。https://www.inf.ethz.ch/personal/gander/

参见:quad, quadv, quadgk, quadcc, trapz, dblquad, triplequad, integral, integral2, integral3。

: q = quadgk (f, a, b)

: q = quadgk (f, a, b, abstol)

: q = quadgk (f, a, b, abstol, trace)

: q = quadgk (f, a, b, "prop", val, …)

: [q, err] = quadgk (…)

利用自适应高斯-克朗罗德积分法对f从a到b的积分进行数值计算。

F是一个函数句柄、内联函数或包含要求值的函数名的字符串。函数f必须被向量化，并且在给定一个输入值的向量时返回一个输出值的向量(参见属性“arrayvalue”了解此规则的例外)。

A和b分别是积分的下限和上限。其中一个或两个极限可以是无穷大或包含弱端奇点。用变量变换来处理任意无限区间并减弱奇异性。例如:

quadgk (@(x) 1 ./ (sqrt (x) .\* (x + 1)), 0, Inf)

请注意，被积函数的公式使用逐元素运算符。/，并且所有quadgk的用户函数都应该使用相同的操作符。

可选参数abstol定义用于停止积分过程的绝对容差。默认值为1e-10 (single为1e-5)。

quadgk使用的算法包括细分积分区间并对每个子区间求值。如果trace为真，则在计算这些部分积分后显示:(1)该步骤的子区间数，(2)误差err的当前估计，(3)积分q的当前估计。

算法的行为可以通过将参数传递给quadgk作为对“prop”，val来配置。有效的属性是

AbsTol

定义正交的绝对误差容限。默认的绝对公差是1e-10(单个为1e-5)。

RelTol

定义正交的相对误差容限。默认的相对容差为1e-6(单个为1e-4)。

ArrayValued

当设置为true时，函数f为标量输入生成数组输出。默认值为false，它要求f生成与输入大小相同的输出。例如,

quadgk (@(x) x .^ (1:5), 0, 2, "ArrayValued", 1)

对x积分。^1, x.^2, x.^3, x.^4, x.^5]在一个函数调用中，而不必重复定义单个匿名函数并使用普通的quadgk调用。

WayPoints

在算法中指定将成为子区间端点的点，这可以显著改善积分误差估计，加快计算速度，或两者兼而有之。在被积函数快速变化的区域周围指定更多的子区间或标记函数一阶导数不连续的位置是有用的。例如，sgum函数在x == 0处具有不连续，并且指定了一个路径点

quadgk (@(x) sign (x), -0.5, 1, "Waypoints", [0])

误差范围从4e-7减小到1e-13。

如果函数在积分区域内有奇点，则不应该用路点来处理。相反，整个积分应该被分解成几个较小积分的和，这样奇点就会作为调用quadgk的积分界之一出现。

如果任何一个路点是复杂的，那么轮廓积分执行如下文件。

MaxIntervalCount

quadgk最初将执行正交的间隔细分为10个间隔，或者，如果给定了路径点，则在每个路径点上细分。具有不可接受错误的子区间将被细分并重新求值。如果在任意一点上的子区间数超过650个子区间，则表示收敛性差，并返回当前的积分估计。属性“MaxIntervalCount”可用于更改退出前可以存在的子间隔的数量。

Trace

如果逻辑上为真，则quadgk在每次迭代时打印有关正交收敛的信息。

如果a, b或路径点中的任何一个是复的，则将正交视为沿分段线性路径的轮廓积分，该路径由[a，路径点(1)，路径点(2)，…，b]定义。在这种情况下，积分假定没有边奇点。例如,

quadgk (@(z) log (z), 1+1i, 1+1i, "WayPoints",

[-1+1i, -1-1i, +1-1i])

对log (z)沿[1+1i， -1+1i， -1-1i， +1-1i]定义的平方积分。

积分的结果在q中返回。

err是积分abs (q - I)误差的近似界，其中I是积分的确切值。如果自适应集成不收敛，err的值将大于要求的容差。如果只请求一个输出，那么当请求的容差不满足时将发出警告。如果请求第二个输出错误，则不会发出警告，程序员有责任检查并确定结果是否令人满意。

参考文献:L.F. Shampine，“MATLAB中的矢量自适应正交”，计算与应用数学学报，第211卷，第2期，2008年2月，第131-140页。

参见:quad, quadv, quadl, quadcc, trapz, dblquad, triplequad, integral, integral2, integral3。

: q = quadcc (f, a, b)

: q = quadcc (f, a, b, tol)

: q = quadcc (f, a, b, tol, sing)

: [q, err, nr\_points] = quadcc (…)

用双自适应克伦肖-柯蒂斯正交法对f从a到b的积分进行数值计算。

F是一个函数句柄、内联函数或包含要求值的函数名的字符串。函数f必须被向量化，如果给定一个输入值的向量，它必须返回一个输出值的向量。例如,

f = @(x) x .\* sin (1./x) .\* sqrt (abs (1 - x));

它对所有操作符使用逐元素的“点”形式。

A和b分别是积分的下限和上限。其中一个或两个极限都可以是无限的。Quadcc通过用x = tan (pi/2\*u)代替积分变量来处理无穷极限。

可选参数tol是一个包含1或2个元素的向量，用于指定所需结果的精度。向量的第一个元素是期望的绝对公差，第二个元素是期望的相对公差。若要仅选择相对测试，请将绝对公差设置为零。若要仅选择绝对测试，请将相对容差设置为零。默认的绝对公差是1e-10(单个为1e-5)，默认的相对公差是1e-6(单个为1e-4)。

可选参数sing包含一个点的列表，在积分区间内，被积函数具有已知的奇点，或者在其任何导数中具有不连续点。对于上面的示例，它在x=1处具有不连续，对quadcc的调用将如下所示

int = quadcc (f, a, b, [], [ 1 ]);

积分的结果在q中返回。

Err是对绝对积分误差的估计。

Nr\_points是计算被积函数的点数。

如果自适应集成不收敛，err的值将大于要求的容差。如果只请求一个输出，那么当请求的容差不满足时将发出警告。如果请求第二个输出错误，则不会发出警告，程序员有责任检查并确定结果是否令人满意。

quadcc能够处理被积数的非数值，如NaN或Inf。如果积分发散，并且quadcc检测到这一点，则发出警告并返回Inf -Inf。

注意:quadcc是一种通用的正交算法，因此，对于光滑或其他表现良好的被积函数，可能比其他方法(如quadgk)效率低。

该算法在每个区间内采用度递增的克伦肖-柯蒂斯积分规则，如果函数不光滑或达到最大度规则，则对区间进行等分。误差估计是根据被积函数在各自正交规则的节点上两次连续插值之间的差的l2范数计算的。

参考文献:P. Gonnet，利用显式插值提高自适应正交的可靠性，中国计算机学会数学软件学报，第37卷，第3期，第3期，2010。

参见:quad, quadv, quadl, quadgk, trapz, dblquad, triplequad。

: q = integral (f, a, b)

: q = integral (f, a, b, prop, val, …)

: [q, err] = integral (…)

用自适应求积分法数值计算f从a到b的积分。

integral是quadcc(一般实值，标量积分和极限)和quadgk(具有指定积分路径和数组值积分的积分)的包装器，旨在提供MATLAB兼容性。通过直接调用各种正交函数，可以实现对数值积分的更多控制。

F是一个函数句柄、内联函数或包含要求值的函数名的字符串。函数f必须被向量化，当给定一个输入值的向量时，它返回一个输出值的向量。

A和b分别是积分的下限和上限。其中一个或两个极限可以是无穷大或包含弱端奇点。如果一个或两个极限都是复数，积分将执行直线路径积分。或者，可以使用“Waypoints”选项指定复杂的域路径(见下文)。

其他可选参数可以使用"property"、" value "对指定。有效的属性有:

Waypoints

指定用于定义正交算法的子区间的点，或者如果a、b或路点是复点，则沿分段连续路径计算正交作为轮廓积分。有关更多详细信息，请参见quadgk。

ArrayValued

Integral期望f返回一个标量值，除非arrayvalue被指定为true。此选项将导致integral对整个数组执行积分，并返回与f返回的维度相同的q。有关详细信息，请参阅quadgk。

AbsTol

定义正交的绝对误差容限。默认的绝对公差是1e-10(单个为1e-5)。

RelTol

定义正交的相对误差容限。默认的相对容差为1e-6(单个为1e-4)。

可选输出err包含被调用的积分器使用的绝对误差估计。

使用自适应正交来最小化误差估计，直到满足以下条件:

error <= max (AbsTol, RelTol\*|q|).

已知的MATLAB不兼容性:

1. 如果公差未指定，并且任何积分限制或路径点都是single类型，则Octave的积分函数会自动减少上述指定的默认绝对和相对误差公差。如果需要更严格的公差，则必须指定。无论积分极限的类别如何，MATLAB都为双输入保留了更严格的公差。

参见:integral2, integral3, quad, quadgk, quadv, quadl, quadcc, trapz, dblquad, triplequad。

有时没有函数，但只有原始的(x, y)点来执行积分。在实验中收集数据时可能会出现这种情况。trapz函数可以对这些值进行积分，如下面的示例所示，其中“数据”是在余弦函数上收集的，范围为[0,pi/2)。

x = 0:0.1:pi/2; # Uniformly spaced points

y = cos (x);

trapz (x, y)

⇒ 0.99666

答案非常接近1的确切值。普通正交对被积函数的特性很敏感。经验积分不仅取决于被积函数，还取决于用来表示函数的特定点。用正弦函数在[0,pi/2]范围内重复上面的例子，结果会差得多。

x = 0:0.1:pi/2; # Uniformly spaced points

y = sin (x);

trapz (x, y)

⇒ 0.92849

然而，数据点的选择稍有不同就会显著改变结果。同样的积分，同样的点数，但间隔不同，会得到更准确的答案。

x = linspace (0, pi/2, 16); # Uniformly spaced, but including endpoint

y = sin (x);

trapz (x, y)

⇒ 0.99909

一般来说，没有办法提前知道最佳的点分布。或者这些点可能来自于一个没有选择最佳分布的自由的实验。无论如何，在使用trapz时必须注意这个问题。

: q = trapz (y)

: q = trapz (x, y)

: q = trapz (…, dim)

用梯形法对y点积分进行数值计算。

Trapz (y)沿第一个非单维计算y的积分。当参数x被省略时，假设一个单位间距为(1)的等间距x向量。trapz (x, y)计算关于x中的间隔和y中的值的积分。如果y中的点采样不均匀，这很有用。

如果给出了可选的dim参数，则沿着这个维度操作。

应用注意:如果没有指定x，则使用单位间距。要将积分缩放到正确的值，必须乘以实际的间距值(deltaX)。例如，x^3在[0,1]范围内的积分是x^4/4或0.25。下面的代码使用trapz以三种不同的方式计算积分。

x = 0:0.1:1;

y = x.^3;

## No scaling

q = trapz (y)

⇒ q = 2.5250

## Approximation to integral by scaling

q \* 0.1

⇒ 0.25250

## Same result by specifying x

trapz (x, y)

⇒ 0.25250

参见:cumtrapz。

: q = cumtrapz (y)

: q = cumtrapz (x, y)

: q = cumtrapz (…, dim)

用梯形法对点y进行累积数值积分。

Cumtrapz (y)沿第一个非单维计算y的累积积分。在trapz只报告整体的积分和的地方，cumtrapz报告当前在y的每个点的部分和值。

当参数x被省略时，假设一个单位间距为(1)的等间距x向量。cumtrapz (x, y)计算关于x中的间隔和y中的值的积分。如果y中的点采样不均匀，这很有用。

如果给出了可选的dim参数，则沿着这个维度操作。

应用注意:如果没有指定x，则使用单位间距。要将积分缩放到正确的值，必须乘以实际的间距值(deltaX)。

参见:trapz, cumsum。

**23.2正交搭配**

: [r, amat, bmat, q] = colloc (n, "left", "right")

计算正交配置的导数和积分权矩阵。

参考文献:J. Villadsen, M. L. Michelsen，用多项式近似解微分方程模型。

下面是一个使用colloc生成权矩阵的示例，用于求解具有边界条件u(0) = 0和u(1) = 1的二阶微分方程u ' - alpha \* u " = 0。

首先，我们可以生成n个点(包括区间的端点)的权重矩阵，并将边界条件合并到右侧(针对特定的alpha值)。

n = 7;

alpha = 0.1;

[r, a, b] = colloc (n-2, "left", "right");

at = a(2:n-1,2:n-1);

bt = b(2:n-1,2:n-1);

rhs = alpha \* b(2:n-1,n) - a(2:n-1,n);

那么根r处的解是

u = [ 0; (at - alpha \* bt) \ rhs; 1]

⇒ [ 0.00; 0.004; 0.01 0.00; 0.12; 0.62; 1.00 ]

**23.3多变量函数**

Octave包括几个函数，用于计算多变量函数的积分。这个过程通常可以通过创建一个函数对f对x积分，然后对y积分来完成。这个过程可以用下面的例子来手动完成:

f(x, y) = sin(pi\*x\*y) \* sqrt(x\*y)

x和y在0和1之间。

在下面的示例中使用quadgk，可以执行二重积分。(请注意，任何一维正交函数都可以以这种方式使用，除了quad，因为它是用Fortran编写的，不能递归调用。)

function q = g(y)

q = ones (size (y));

for i = 1:length (y)

f = @(x) sin (pi\*x.\*y(i)) .\* sqrt (x.\*y(i));

q(i) = quadgk (f, 0, 1);

endfor

endfunction

I = quadgk ("g", 0, 1)

⇒ 0.30022

上述算法在函数dblquad中实现，用于两个变量上的积分。这个过程的3-D等效是用triplequad实现的，用于三个变量的积分。例如，可以通过调用dblquad复制上面的结果，如下所示。

I = dblquad (@(x, y) sin (pi\*x.\*y) .\* sqrt (x.\*y), 0, 1, 0, 1)

⇒ 0.30022

: q = dblquad (f, xa, xb, ya, yb)

: q = dblquad (f, xa, xb, ya, yb, tol)

: q = dblquad (f, xa, xb, ya, yb, tol, quadf)

: q = dblquad (f, xa, xb, ya, yb, tol, quadf, …)

用数值计算f的二重积分。

F是一个函数句柄、内联函数或包含要求值的函数名的字符串。函数f必须具有z = f(x,y)的形式，其中x是矢量，y是标量。它应该返回一个与x相同长度和方向的向量。

Xa, ya和xb, yb分别是x和y的积分下限和上限。底层积分器决定是否接受无限边界。

可选参数tol定义用于积分每个子积分的绝对容差。默认值为1e-6。

可选参数quadf指定要使用哪个底层积分器函数。除了quad之外的任何选择都是可用的，默认是quadcc。

其他参数直接传递给f。若要使用tol或quadf的默认值，可以传递':'或空矩阵([])。

参见:integral2, integral3, triplequad, quad, quadv, quadl, quadgk, quadcc, trapz。

: q = triplequad (f, xa, xb, ya, yb, za, zb)

: q = triplequad (f, xa, xb, ya, yb, za, zb, tol)

: q = triplequad (f, xa, xb, ya, yb, za, zb, tol, quadf)

: q = triplequad (f, xa, xb, ya, yb, za, zb, tol, quadf, …)

用数值计算f的三重积分。

F是一个函数句柄、内联函数或包含要求值的函数名的字符串。函数f必须具有w = f(x,y,z)的形式，其中x或y是向量，其余输入是标量。它应该返回一个长度和方向与x或y相同的向量。

Xa, ya, za和xb, yb, zb分别是x, y, z的积分下限和上限。底层积分器决定是否接受无限边界。

可选参数tol定义用于积分每个子积分的绝对容差。默认值为1e-6。

可选参数quadf指定要使用哪个底层积分器函数。除了quad之外的任何选择都是可用的，默认是quadcc。

其他参数直接传递给f。若要使用tol或quadf的默认值，可以传递':'或空矩阵([])。

参见:integral3, integral2, dblquad, quad, quadv, quadl, quadgk, quadcc, trapz。

上面提出的求正交的递归算法称为“迭代”。在函数quad2d中实现了一种单独的二维积分方法。该函数通过将积分域细分为矩形区域并在这些域上执行单独的积分来执行“平铺”积分。这些域被进一步细分为需要细化的区域，以达到所需的数值精度。对于某些函数，这种方法可以比上述其他函数中使用的二维迭代更快。

: q = quad2d (f, xa, xb, ya, yb)

: q = quad2d (f, xa, xb, ya, yb, prop, val, …)

: [q, err, iter] = quad2d (…)

利用自适应正交在由xa, xb, ya, yb定义的二维域上用平铺积分对f的二维积分进行数值计算。另外，ya和yb可以是x的标量函数，允许在非矩形域上积分。

F是一个函数句柄、内联函数或包含要求值的函数名的字符串。函数f必须具有z = f(x,y)的形式，并且所有操作必须向量化，以便x和y接受数组输入并返回相同大小的数组输出。(可以假设x和y是相同大小的数组，或者其中一个是标量。) 底层积分器将把积分点数组输入f和/或使用内部向量展开来加快计算速度，如果f不限于元素操作，则可能产生不可预测的结果。对于不可避免的积分，下面描述的(“矢量化”)选项可能会产生更可靠的结果。

其他可选参数可以使用"property"、" value "对指定。有效的属性有:

AbsTol

定义正交的绝对误差容限。默认值为1e-10 (single为1e-5)。

RelTol

定义正交的相对误差容限。默认值为1e-6 (single为1e-4)。

MaxFunEvals

向量化函数f的最大函数调用次数。默认值为5000。

Singular

启用/禁用转换以削弱集成域边缘的奇异性。默认值为true。

Vectorized

启用或禁用矢量化集成。false值强制Octave在调用被积函数时只使用标量输入，这使得未向量化的被积函数f(x,y)或只接受x或y的标量值。默认值为true。注意，这是通过用函数arrayfun包装f(x,y)来实现的，这可能会显著降低计算速度。

FailurePlot

如果在达到MaxFunEvals之前，quad2d未能收敛到期望的误差容限，则创建一个仍然需要改进的区域的图。默认值为false。

使用自适应正交来最小化误差估计，直到满足以下条件:

error <= max (AbsTol, RelTol\*|q|)

可选输出err是积分abs (q - I)误差的近似边界，其中I是积分的确切值。可选输出iter是对函数f所使用的向量化函数调用的次数。

例1:在x-y平面上对一个矩形区域进行积分

f = @(x,y) 2\*ones (size (x));

q = quad2d (f, 0, 1, 0, 1)

⇒ q = 2

结果是一个体积，对于这个常值被积函数，它只是长\*宽\*高。

例2:在x-y平面上对一个三角形区域进行积分

f = @(x,y) 2\*ones (size (x));

ymax = @(x) 1 - x;

q = quad2d (f, 0, 1, 0, ymax)

⇒ q = 1

结果是一个体积，对于这个常值被积函数f = 2，它等于三角形面积x高或1/2 \*底\*宽\*高。

例3:在一个方形区域上对一个非矢量化函数积分

f = @(x,y) sinc (x) \* sinc (y));

q = quad2d (f, -1, 1, -1, 1)

⇒ q = 12.328 (incorrect)

q = quad2d (f, -1, 1, -1, 1, "Vectorized", false)

⇒ q = 1.390 (correct)

f = @(x,y) sinc (x) .\* sinc (y);

q = quad2d (f, -1, 1, -1, 1)

⇒ q = 1.390 (correct)

第一个结果是不正确的，因为f中的sinc函数之间的非元素运算符在quad2d使用的内部集成数组之间创建了意想不到的矩阵乘法。在第二种结果中，将“Vectorized”设置为false强制quad2d执行标量内部运算来计算积分，从而以增加约20倍的计算时间为代价获得正确的数值结果。在第三种结果中，使用元素乘法算子对被积函数进行矢量化，在不增加计算时间的情况下得到了正确的结果。

编程注意事项:如果积分区域内存在奇异点，最好将积分拆分，并将奇异点置于边界上。

已知的MATLAB不兼容性:如果公差未指定，并且任何积分限制类型为single，则Octave的积分函数会自动减少上述指定的默认绝对和相对误差公差。如果需要更严格的公差，则必须指定。无论积分极限的类别如何，MATLAB都为双输入保留了更严格的公差。

参考文献:L.F. Shampine，二维正交的MATLAB程序，应用数学与计算，pp. 266-274, Vol . 1, 2008。

参见:integral2, dblquad, integral, quad, quadgk, quadv, quadl, quadcc, trapz, integral3, triplequad。

最后，给出了函数integral2和integral3作为一般的二维和三维积分函数。它们将在迭代和平铺集成方法之间自动选择，并且与双四边形和三四边形不同，它们将处理非矩形集成域。

: q = integral2 (f, xa, xb, ya, yb)

: q = integral2 (f, xa, xb, ya, yb, prop, val, …)

: [q, err] = integral2 (…)

在由xa, xb, ya, yb(标量可以是有限的或无限的)定义的二维域上，用自适应正交对f的二维积分进行数值计算。另外，ya和yb可以是x的标量函数，允许在非矩形域上积分。

F是一个函数句柄、内联函数或包含要求值的函数名的字符串。函数f必须具有z = f(x,y)的形式，并且所有操作必须向量化，以便x和y接受数组输入并返回相同大小的数组输出。(可以假设x和y是相同大小的数组，或者其中一个是标量。) 底层积分器将把积分点数组输入f和/或使用内部向量展开来加快计算速度，如果f不限于元素操作，则可能产生不可预测的结果。对于不可避免的积分，下面描述的(“矢量化”)选项可能会产生更可靠的结果。

其他可选参数可以使用"property"、" value "对指定。有效的属性有:

AbsTol

定义正交的绝对误差容限。默认值为1e-10 (single为1e-5)。

RelTol

定义正交的相对误差容限。默认值为1e-6 (single为1e-4)。

Method

指定要使用的二维集成方法，有效的选项是“auto”(默认)、“titile”或“iterate”。当使用“自动”时，Octave将选择“平铺”方法，除非任何积分限制是无限的。

Vectorized

启用或禁用矢量化集成。false值强制Octave在调用被积函数时只使用标量输入，这使得未向量化的被积函数f(x,y)或只接受x或y的标量值。默认值为true。注意，这是通过用函数arrayfun包装f(x,y)来实现的，这可能会显著降低计算速度。

使用自适应正交来最小化误差估计，直到满足以下条件:

error <= max (AbsTol, RelTol\*|q|)

err是积分abs (q - I)误差的近似界，其中I是积分的确切值。

例1:在x-y平面上对一个矩形区域进行积分

f = @(x,y) 2\*ones (size (x));

q = integral2 (f, 0, 1, 0, 1)

⇒ q = 2

结果是一个体积，对于这个常值被积函数，它只是长\*宽\*高。

例2:在x-y平面上对一个三角形区域进行积分

f = @(x,y) 2\*ones (size (x));

ymax = @(x) 1 - x;

q = integral2 (f, 0, 1, 0, ymax)

⇒ q = 1

结果是一个体积，对于这个常值被积函数f = 2，它等于三角形面积x高或1/2 \*底\*宽\*高。

例3:在一个方形区域上对一个非矢量化函数积分

f = @(x,y) sinc (x) \* sinc (y));

q = integral2 (f, -1, 1, -1, 1)

⇒ q = 12.328 (incorrect)

q = integral2 (f, -1, 1, -1, 1, "Vectorized", false)

⇒ q = 1.390 (correct)

f = @(x,y) sinc (x) .\* sinc (y);

q = integral2 (f, -1, 1, -1, 1)

⇒ q = 1.390 (correct)

第一个结果是不正确的，因为f中的sinc函数之间的非元素运算符在integral2使用的内部积分数组之间创建了意想不到的矩阵乘法。在第二种结果中，将“Vectorized”设置为false强制integral2执行标量内部运算来计算积分，从而以增加约20倍的计算时间为代价获得正确的数值结果。在第三种结果中，使用元素乘法算子对被积函数进行矢量化，在不增加计算时间的情况下得到了正确的结果。

编程注意事项:如果积分区域内存在奇异点，最好将积分拆分，并将奇异点置于边界上。

已知的MATLAB不兼容性:如果公差未指定，并且任何积分限制类型为single，则Octave的积分函数会自动减少上述指定的默认绝对和相对误差公差。如果需要更严格的公差，则必须指定。无论积分极限的类别如何，MATLAB都为双输入保留了更严格的公差。

参考文献:L.F. Shampine，二维正交的MATLAB程序，应用数学与计算，pp. 266-274, Vol . 1, 2008。

参见:quad2d, dblquad, integral, quad, quadgk, quadv, quadl, quadcc, trapz, integral3, triplequad。

: q = integral3 (f, xa, xb, ya, yb, za, zb)

: q = integral3 (f, xa, xb, ya, yb, za, zb, prop, val, …)

在由xa, xb, ya, yb, za, zb(标量可以是有限的或无限的)定义的三维域上，用自适应正交对f的三维积分进行数值计算。另外，ya和yb可能是x和za的标量函数，zb可能是x和y的标量函数，允许在非矩形域上积分。

F是一个函数句柄、内联函数或包含要求值的函数名的字符串。函数f必须具有z = f(x,y,z)的形式，并且所有操作必须向量化，以便x,y和z接受数组输入并返回相同大小的数组输出。(可以假设x, y和z将是相同大小的数组或标量。)底层积分器将把积分点数组输入f和/或使用内部向量展开来加快计算速度，如果f不限于元素操作，则可能产生不可预测的结果。对于不可避免的积分，下面描述的(“矢量化”)选项可能会产生更可靠的结果。

其他可选参数可以使用"property"、" value "对指定。有效的属性有:

AbsTol

定义正交的绝对误差容限。默认值为1e-10 (single为1e-5)。

RelTol

定义正交的相对误差容限。默认值为1e-6 (single为1e-4)。

Method

指定要使用的二维集成方法，有效的选项是“auto”(默认)、“titile”或“iterate”。当使用“自动”时，Octave将选择“平铺”方法，除非任何积分限制是无限的。

Vectorized

启用或禁用矢量化集成。false值强制Octave在调用被积函数时只使用标量输入，这使得未向量化的被积函数f(x,y,z)或只接受x,y或z的标量值。默认值为true。注意，这是通过用函数arrayfun包装f(x,y,z)来实现的，这可能会显著降低计算速度。

使用自适应正交来最小化误差估计，直到满足以下条件:

error <= max (AbsTol, RelTol\*|q|)

err是积分abs (q - I)误差的近似界，其中I是积分的确切值。

例1:对一个矩形体积积分

f = @(x,y,z) ones (size (x));

q = integral3 (f, 0, 1, 0, 1, 0, 1)

⇒ q = 1.00000

对于这个常值被积函数，结果是一个长度\*宽度\*高度的体积。

例2:对球面体积积分

f = @(x,y) ones (size (x));

ymax = @(x) sqrt (1 - x.^2);

zmax = @(x,y) sqrt (1 - x.^2 - y.^2);

q = integral3 (f, 0, 1, 0, ymax, 0, zmax)

⇒ q = 0.52360

对于这个常值被积函数，得到的体积是单位球体的1/8或者1/8 \* 4/3 \* pi。

例3:在一个立方体积上对一个非矢量化函数积分

f = @(x,y) sinc (x) \* sinc (y), \* sinc (z);

q = integral3 (f, -1, 1, -1, 1, -1, 1)

⇒ q = 14.535 (incorrect)

q = integral3 (f, -1, 1, -1, 1, -1, 1, "Vectorized", false)

⇒ q = 1.6388 (correct)

f = @(x,y,z) sinc (x) .\* sinc (y), .\* sinc (z);

q = integral3 (f, -1, 1, -1, 1, -1, 1)

⇒ q = 1.6388 (correct)

第一个结果是不正确的，因为f中的sinc函数之间的非元素运算符在integral3使用的内部积分数组之间创建了意想不到的矩阵乘法。在第二种结果中，将“Vectorized”设置为false强制integral3执行标量内部运算来计算积分，从而得到正确的数值结果，但代价是计算时间增加了约30倍。在第三种结果中，使用元素乘法算子对被积函数进行矢量化，在不增加计算时间的情况下得到了正确的结果。

编程注意事项:如果积分区域内存在奇异点，最好将积分拆分，并将奇异点置于边界上。

已知的MATLAB不兼容性:如果公差未指定，并且任何积分限制类型为single，则Octave的积分函数会自动减少上述指定的默认绝对和相对误差公差。如果需要更严格的公差，则必须指定。无论积分极限的类别如何，MATLAB都为双输入保留了更严格的公差。

参考文献:L.F. Shampine，二维正交的MATLAB程序，应用数学与计算，pp. 266-274, Vol . 1, 2008。

参见:triplequad, integral, quad, quadgk, quadv, quadl, quadcc, trapz, integral2, quad2d, dblquad。

上面的积分可能相当慢，而且这个问题随着积分的维数呈指数增长。二维集成的另一个可能的解决方案是使用上一节中描述的正交配置(请参阅正交配置)。函数f(x,y)对x和y在0和1之间的积分可以用n个点来近似，即i=1:n和j=1:n对q(i)\*q(j)\*f(r(i)，r(j))的求和，其中q和r是由colc (n)返回的。对两个以上变量的推广是直接的。下面的代码使用n=8个点计算所研究的积分。

f = @(x,y) sin (pi\*x\*y') .\* sqrt (x\*y');

n = 8;

[t, ~, ~, q] = colloc (n);

I = q'\*f(t,t)\*q;

⇒ 0.30022

应该注意的是，点的数量决定了近似的质量。如果需要在a和b之间进行积分，而不是在0和1之间，那么就需要改变变量。

**24微分方程**

Octave具有用于求解常微分方程(ode)和微分代数方程(DAEs)的内置函数。

**24.1常微分方程**

函数lsode可用于求解表单的ode

dx

-- = f (x, t)

dt

使用Hindmarsh的ODE求解器LSODE。

: [x, istate, msg] = lsode (fcn, x\_0, t)

: [x, istate, msg] = lsode (fcn, x\_0, t, t\_crit)

常微分方程(ODE)求解器。

要解的微分方程组是

dx

-- = f (x, t)

dt

with

x(t\_0) = x\_0

在矩阵x中返回解，每一行对应向量t的一个元素。t的第一个元素应为t\_0，对应于系统x\_0的初始状态，因此输出的第一行为x\_0。

第一个参数fcn是一个字符串、内联或函数句柄，用于命名要调用的函数f，以计算一组方程右侧的向量。函数必须有这样的形式

xdot = f (x, t)

其中x导和x是向量t是标量。

如果fcn是由字符串、内联函数或函数句柄组成的双元素字符串数组或双元素单元数组，则第一个元素命名上述函数f，第二个元素命名用于计算f的雅可比矩阵的函数。雅可比矩阵函数必须具有这样的形式

jac = j (x, t)

其中jac是偏导数矩阵

| df\_1 df\_1 df\_1 |

| ---- ---- ... ---- |

| dx\_1 dx\_2 dx\_N |

| |

| df\_2 df\_2 df\_2 |

| ---- ---- ... ---- |

df\_i | dx\_1 dx\_2 dx\_N |

jac = ---- = | |

dx\_j | . . . . |

| . . . . |

| . . . . |

| |

| df\_M df\_M df\_M |

| ---- ---- ... ---- |

| dx\_1 dx\_2 dx\_N |

第二个参数指定系统x\_0的初始状态。第三个参数是一个向量t，指定了寻找解的时间值。

第四个参数是可选的，可用于指定ODE求解器不应该集成过去的一组时间。它有助于避免导数中存在不连续点和奇异点的困难。

计算成功后，state的值将为2(与LSODE的Fortran版本一致)。

如果计算不成功，则state将不是2，并且msg将包含其他信息。

可以使用lsode\_options函数为lsode设置可选参数。

参见Alan C. Hindmarsh, ODEPACK, ODE求解器的系统化集合，在科学计算中，R. S. Stepleman，编辑，(1983)或https://computing.llnl.gov/projects/odepack获取关于lsode内部工作的更多信息。

例:解范德波尔方程

fvdp = @(y,t) [y(2); (1 - y(1)^2) \* y(2) - y(1)];

t = linspace (0, 20, 100);

y = lsode (fvdp, [2; 0], t);

参见:daspk, dassl, dasst。

: lsode\_options ()

: val = lsode\_options (opt)

: lsode\_options (opt, val)

查询或设置lsode函数的选项。

当不带参数调用时，将显示所有可用选项的名称及其当前值。

给定一个参数，返回选项opt的值。

当使用两个参数调用时，lsode\_options将选项opt设置为value val。

选项包括

"absolute tolerance"

绝对宽容。可以是向量也可以是标量。如果是向量，它必须与状态向量的维数相匹配。

"relative tolerance"

相对公差参数。与绝对容差不同，此参数只能是标量。

在每个集成步骤中应用的本地错误测试为

abs (local error in x(i)) <= ...

rtol \* abs (y(i)) + atol(i)

"integration method"

一个字符串，指定用于求解ODE系统的集成方法。有效值是

"adams"

"non-stiff"

没有使用雅可比矩阵(即使它可用)。

"bdf"

"stiff"

采用硬后向微分公式(BDF)方法。如果没有提供计算雅可比矩阵的函数，那么lsode将计算雅可比矩阵的有限差分近似。

"initial step size"

在第一步中尝试的步长(默认值是自动确定的)。

"maximum order"

限制解法的最大阶数。如果使用Adams方法，该选项必须在1到12之间。否则，必须在1到5之间(包括5)。

"maximum step size"

设置最大步长将避免传递非常大的区域(默认值未指定)。

"minimum step size"

允许的最小绝对步长(默认为0)。

"step limit"

允许的最大步数(默认为100000)。

"jacobian type"

一个字符串指定雅可比矩阵的类型,它使用的是硬向后微分公式(BDF)集成方法。有效值

"full"

默认值。所有偏导数都是由用户提供的雅可比函数近似或使用的。

"banded"

只有“下雅可比子对角线”和“上雅可比子对角线”选项分别指定的对角线和上下亚对角线的数量是由用户提供的雅可比函数近似或使用的。用户提供的雅可比函数可以将所有其他偏导数设置为任意值。

"diagonal"

如果用户提供了雅可比函数，则此设置不起作用。由lsode近似的雅可比矩阵被限制在对角线上，其中每个偏导数是通过对状态的所有元素施加有限变化来计算的;如果真实的雅可比矩阵确实总是对角的，这与只对状态的各自元素应用有限变化具有相同的效果，但效率更高。

"lower jacobian subdiagonals"

如果选项“雅可比类型”设置为“带状”，则使用的较低次对角线的数量。默认值为零。

"upper jacobian subdiagonals"

如果选项“雅可比类型”设置为“带状”，则使用的上亚对角线数量。默认值为零。

下面是一个使用lsode求解一组三个微分方程的例子。给定函数

## oregonator differential equation

function xdot = f (x, t)

xdot = zeros (3,1);

xdot(1) = 77.27 \* (x(2) - x(1)\*x(2) + x(1) ...

- 8.375e-06\*x(1)^2);

xdot(2) = (x(3) - x(1)\*x(2) - x(2)) / 77.27;

xdot(3) = 0.161\*(x(1) - x(3));

endfunction

初始条件x0 = [4;1.1;4]，可使用命令对方程组进行积分

t = linspace (0, 500, 1000);

y = lsode ("f", x0, t);

如果您尝试这样做，您将看到结果的值在t = 0和5之间以及在t = 305附近发生了巨大的变化。一组更有效的输出点可能是

t = [0, logspace(-1, log10(303), 150), ...

logspace(log10(304), log10(500), 150)];

上面使用的微分方程的m文件包含在示例目录中的Octave分布中，其名称为oregonator.m。

**24.2微分代数方程**

函数daspk可以用来求解表单的DAEs

0 = f (x-dot, x, t), x(t=0) = x\_0, x-dot(t=0) = x-dot\_0

式中x导为x的导数，用Petzold的DAE求解器DASPK求解方程。

: [x, xdot, istate, msg] = daspk (fcn, x\_0, xdot\_0, t, t\_crit)

解一组微分代数方程。

Daspk解出了这组方程

0 = f (x, xdot, t)

with

x(t\_0) = x\_0, xdot(t\_0) = xdot\_0

解在矩阵x和xdot中返回，结果矩阵中的每一行对应向量t中的一个元素。t的第一个元素应为t\_0，对应系统x\_0及其导数xdot\_0的初始状态，因此输出x的第一行为x\_0，输出xdot的第一行为xdot\_0。

第一个参数fcn是一个字符串、内联或函数句柄，用于命名要调用的函数f，以计算一组方程的残差向量。它必须有这个形式

res = f (x, xdot, t)

其中x x导和res是向量，t是标量。

如果fcn是由字符串、内联函数或函数句柄组成的双元素字符串数组或双元素单元数组，则第一个元素命名上述函数f，第二个元素命名用于计算修改后的雅可比矩阵的函数

df df

jac = -- + c ------

dx d xdot

修正后的雅可比函数必须有这样的形式

jac = j (x, xdot, t, c)

daspk的第二个和第三个参数指定状态及其导数的初始条件，第四个参数指定期望解的输出时间向量，包括与初始条件对应的时间。

初始状态和导数的集合并不严格要求一致。如果它们不一致，则必须使用daspk\_options函数提供额外的信息，以便daspk可以计算一致的起点。

第五个参数是可选的，可用于指定DAE求解器不应集成过去的一组时间。它有助于避免导数中存在不连续点和奇异点的困难。

计算成功后，state的值将大于零(与DASPK的Fortran版本一致)。

如果计算不成功，state的值将小于零，msg将包含额外的信息。

可以使用daspk\_options函数设置daspk的可选参数。

参见:dassl。

: daspk\_options ()

: val = daspk\_options (opt)

: daspk\_options (opt, val)

查询或设置daspk函数的选项。

当不带参数调用时，将显示所有可用选项的名称及其当前值。

给定一个参数，返回选项opt的值。

当带两个参数调用时，daspk\_options将选项opt设置为value val。

选项包括

"absolute tolerance"

绝对宽容。可以是向量也可以是标量。如果是矢量，则必须与状态矢量的维度相匹配，并且相对容差也必须是相同长度的矢量。

"relative tolerance"

相对宽容。可以是向量也可以是标量。如果是矢量，则必须与状态矢量的尺寸相匹配，并且绝对公差也必须是相同长度的矢量。

在每个集成步骤中应用的本地错误测试为

abs (local error in x(i))

<= rtol(i) \* abs (Y(i)) + atol(i)

"compute consistent initial condition"

用Y\_d表示状态向量中的微分变量，用Y\_a表示代数变量，ddaspk可以解决以下两个初始化问题之一:

1. Given Y\_d, calculate Y\_a and Y’\_d
2. Given Y’, calculate Y.

在任何一种情况下，给定组件的初始值都是输入，未知组件的初始猜测也必须作为输入提供。将该选项设置为1解决第一个问题，或设置为2解决第二个问题(默认值为0，因此必须提供一组一致的初始条件)。

如果将此选项设置为非零值，则还必须设置“代数变量”选项，以声明问题中的哪些变量是代数的。

"use initial condition heuristics"

设置为非零值以使用下面描述的初始条件启发式选项。

"initial condition heuristics"

下列参数的矢量，可用于控制初始条件的计算。

MXNIT

Newton迭代的最大次数(默认为5)。

MXNJ

雅可比矩阵计算的最大次数(默认为6)。

MXNH

如果“计算一致初始条件”选项设置为1(默认为5)，则要尝试的人工步长参数的最大数值。

注意，如果“计算一致初始条件”选项设置为1，则允许的Newton迭代的最大总数为MXNIT\*MXNJ\*MXNH，如果设置为2，则为MXNIT\*MXNJ。

LSOFF

设置一个非零值来禁用行搜索算法(默认为0)。

STPTOL

线性研究算法中的最小缩放步长(默认为eps^(2/3))。

EPINIT

牛顿迭代收敛检验中的摇摆因子。测试应用于残差向量，预乘近似雅可比矩阵。为了收敛，该向量的加权RMS范数(由误差权重缩放)必须小于EPINIT\*EPCON，其中EPCON = 0.33是时间步长中使用的类似测试常数。默认值为EPINIT = 0.01。

"print initial condition info"

将此选项设置为非零值，以显示有关初始条件计算的详细信息(默认为0)。

"exclude algebraic variables from error test"

设置为非零值以从错误测试中排除代数变量。您还必须设置“algebraic variables”选项，以声明问题中的哪些变量是代数的(默认为0)。

"algebraic variables"

与状态向量长度相同的向量。非零元素表示状态向量对应的元素是一个代数变量(即它的导数没有显式出现在方程集中)。

“计算一致的初始条件”和“从错误测试中排除代数变量”选项需要此选项。

"enforce inequality constraints"

设置为以下值之一，以强制执行由“不等式约束类型”选项指定的不等式约束(默认为0)。

1. 仅在初始条件计算中进行约束验算。
2. 在集成期间强制约束检查。
3. 执行选项1和2。

"inequality constraint types"

与指定不等式约束类型的状态具有相同长度的向量。向量的每个元素对应于状态的一个元素，并应分配以下代码之一

-2

小于零。

-1

小于或等于零的。

0

没有限制。

1

大于或等于零的。

2

大于零。

此选项仅在“强制不等式约束”选项非零时有效。

"initial step size"

微分代数问题有时会在第一步遇到严重的标度困难。如果您非常了解问题的可伸缩性，那么可以通过指定初始步长(默认步长是自动计算的)来帮助缓解这个问题。

"maximum order"

限制解法的最大阶数。该选项必须在1到5之间，包括(默认为5)。

"maximum step size"

设置最大步长将避免传递非常大的区域(默认值未指定)。

Octave还包括DASSL (DASPK的早期版本)和DASRT，它们可用于解决带有约束(停止条件)的DAEs。

: [x, xdot, istate, msg] = dassl (fcn, x\_0, xdot\_0, t, t\_crit)

解一组微分代数方程。

Dassl解出了这组方程

0 = f (x, xdot, t)

with

x(t\_0) = x\_0, xdot(t\_0) = xdot\_0

解在矩阵x和xdot中返回，结果矩阵中的每一行对应向量t中的一个元素。t的第一个元素应为t\_0，对应系统x\_0及其导数xdot\_0的初始状态，因此输出x的第一行为x\_0，输出xdot的第一行为xdot\_0。

第一个参数fcn是一个字符串、内联或函数句柄，用于命名要调用的函数f，以计算一组方程的残差向量。它必须有这个形式

res = f (x, xdot, t)

其中x x导和res是向量，t是标量。

如果fcn是由字符串、内联函数或函数句柄组成的双元素字符串数组或双元素单元数组，则第一个元素命名上述函数f，第二个元素命名用于计算修改后的雅可比矩阵的函数

df df

jac = -- + c ------

dx d xdot

修正后的雅可比函数必须有这样的形式

jac = j (x, xdot, t, c)

dassl的第二个和第三个参数指定状态及其导数的初始条件，第四个参数指定期望解的输出时间向量，包括与初始条件对应的时间。

初始状态和导数的集合并不严格要求一致。然而，在实践中，DASSL并不擅长为您确定一个一致的集合，因此最好确保初始值导致函数求值为零。

第五个参数是可选的，可用于指定DAE求解器不应集成过去的一组时间。它有助于避免导数中存在不连续点和奇异点的困难。

计算成功后，state的值将大于零(与DASSL的Fortran版本一致)。

如果计算不成功，state的值将小于零，msg将包含额外的信息。

可以使用dassl\_options函数设置dassl的可选参数。

参见:daspk, daspt, lsode。

: dassl\_options ()

: val = dassl\_options (opt)

: dassl\_options (opt, val)

查询或设置dassl函数的选项。

当不带参数调用时，将显示所有可用选项的名称及其当前值。

给定一个参数，返回选项opt的值。

当带两个参数调用时，dassl\_options将选项opt设置为value val。

选项包括

"absolute tolerance"

绝对宽容。可以是向量也可以是标量。如果是矢量，则必须与状态矢量的维度相匹配，并且相对容差也必须是相同长度的矢量。

"relative tolerance"

相对宽容。可以是向量也可以是标量。如果是矢量，则必须与状态矢量的尺寸相匹配，并且绝对公差也必须是相同长度的矢量。

在每个集成步骤中应用的本地错误测试为

abs (local error in x(i))

<= rtol(i) \* abs (Y(i)) + atol(i)

"compute consistent initial condition"

如果非零，dassl将尝试计算一组一致的初始条件。这通常是不可靠的，因此最好提供一个一致的集合，并将此选项设置为零。

"enforce nonnegativity constraints"

如果您知道方程的解总是非负的，则将此参数设置为非零值可能会有所帮助。但是，最好先尝试将此选项设置为零，只有在效果不佳时才将其设置为非零值。

"initial step size"

微分代数问题有时会在第一步遇到严重的标度困难。如果您非常了解问题的可伸缩性，则可以通过指定初始步长来帮助缓解这个问题。

"maximum order"

限制解法的最大阶数。该选项必须在1到5之间，包括1和5。

"maximum step size"

设置最大步长将避免传递非常大的区域(默认值未指定)。

"step limit"

对底层Fortran代码的单个调用尝试的最大集成步骤数。

: [x, xdot, t\_out, istat, msg] = dasrt (fcn, g, x\_0, xdot\_0, t)

: … = dasrt (fcn, g, x\_0, xdot\_0, t, t\_crit)

: … = dasrt (fcn, x\_0, xdot\_0, t)

: … = dasrt (fcn, x\_0, xdot\_0, t, t\_crit)

解一组微分代数方程。

dasrt解方程组

0 = f (x, xdot, t)

with

x(t\_0) = x\_0, xdot(t\_0) = xdot\_0

具有功能性停止准则(解根)。

解以矩阵x和xdot的形式返回，结果矩阵中的每一行对应于向量t\_out中的一个元素。t的第一个元素应为t\_0，对应于系统x\_0及其导数xdot\_0的初始状态，因此输出x的第一行为x\_0，输出xdot的第一行为xdot\_0。

向量t给出了积分长度的上限。如果停止条件满足，则向量t\_out将小于t，并且t\_out的最后一个元素将是满足停止条件的点，并且可以不对应于向量t的任何元素。

第一个参数fcn是一个字符串、内联或函数句柄，用于命名要调用的函数f，以计算一组方程的残差向量。它必须有这个形式

res = f (x, xdot, t)

其中x x导和res是向量，t是标量。

如果fcn是由字符串、内联函数或函数句柄组成的双元素字符串数组或双元素单元数组，则第一个元素命名上述函数f，第二个元素命名用于计算修改后的雅可比矩阵的函数

df df

jac = -- + c ------

dx d xdot

修正后的雅可比函数必须有这样的形式

jac = j (x, xdot, t, c)

可选的第二个参数命名了一个函数，该函数定义了在集成期间需要其根的约束函数。这个函数必须有这样的形式

g\_out = g (x, t)

返回约束函数值的向量。如果任何约束函数的值改变了符号，DASRT将尝试在符号改变的点上停止积分。

如果省略约束函数的名称，则dasrt解决的问题与daspk或dassl相同。

请注意，由于四舍五入和积分误差导致约束函数中的数值误差，DASRT可能返回假根，或者在两个或多个接近相等的t上返回相同的根。如果怀疑存在这种假根，则用户应考虑在约束函数的求值中使用更小的误差容忍度或更高的精度。

如果某个约束函数的根定义了问题的结束，那么DASRT的输入应该允许积分到稍微超过该根的一点，以便DASRT可以通过插值来定位根。

dasrt的第三和第四个参数指定状态及其导数的初始条件，第四个参数指定期望解的输出时间向量，包括与初始条件对应的时间。

初始状态和导数的集合并不严格要求一致。然而，在实践中，DASSL并不擅长为您确定一个一致的集合，因此最好确保初始值导致函数求值为零。

第六个参数是可选的，可用于指定DAE求解器不应集成过去的一组时间。它有助于避免导数中存在不连续点和奇异点的困难。

计算成功后，state的值将大于零(与DASSL的Fortran版本一致)。

如果计算不成功，state的值将小于零，msg将包含额外的信息。

可以使用dasrt\_options函数为dasrt设置可选参数。

参见:dasrt\_options, daspk, dasrt, lsode。

: dasrt\_options ()

: val = dasrt\_options (opt)

: dasrt\_options (opt, val)

查询或设置函数dasrt的选项。

当不带参数调用时，将显示所有可用选项的名称及其当前值。

给定一个参数，返回选项opt的值。

当带两个参数调用时，dasrt\_options将选项opt设置为值val。

选项包括

"absolute tolerance"

绝对宽容。可以是向量也可以是标量。如果是矢量，则必须与状态矢量的维度相匹配，并且相对容差也必须是相同长度的矢量。

"relative tolerance"

相对宽容。可以是向量也可以是标量。如果是矢量，则必须与状态矢量的尺寸相匹配，并且绝对公差也必须是相同长度的矢量。

在每个集成步骤中应用的本地错误测试为

abs (local error in x(i)) <= ...

rtol(i) \* abs (Y(i)) + atol(i)

"initial step size"

微分代数问题有时会在第一步遇到严重的标度困难。如果您非常了解问题的可伸缩性，则可以通过指定初始步长来帮助缓解这个问题。

"maximum order"

限制解法的最大阶数。该选项必须在1到5之间，包括1和5。

"maximum step size"

设置最大步长将避免经过非常大的区域。

"step limit"

最大数量的集成步骤,以尝试单个调用底层的Fortran代码。

见K. E. Brenan,等。,n -代数方程组中初始值问题的数值解,DOI:https://doi.org/10.1137/1.9781611971224,了解更多关于DASSL的实现的信息。

**24.3与matlab兼容的求解器**

Octave还提供了一组具有matlab兼容接口的常微分方程(ode)初值问题的求解器。这类方法的选项是使用函数设置的。

目前实现的求解器有:

* 龙格-库塔方法
* ode45利用高阶变步长Dormand-Prince方法集成了一组非刚性微分代数方程或指数-1微分代数方程(DAEs)。它每个积分步骤需要6个函数计算，但在平滑问题上可能比ode23采取更大的步骤:在更小的公差下可能提供更高的效率。
* ode23集成了一个非刚性ode或(或index-1 dae)系统。它使用三阶Bogacki-Shampine方法，并调整局部步长以满足用户指定的公差。求解器在每个积分步骤中需要三个函数求值。
* ode23s使用改进的二阶Rosenbrock方法集成了一个刚性ode(或index-1 DAEs)系统。
* 线性多步法
* ode15s使用一种基于后向差分公式(BDF)的变步长、变阶方法集成了一个刚性ode(或index-1 DAEs)系统。
* ode15i集成了一个全隐式ode(或index-1 dae)系统，使用与ode15s相同的可变步长、可变顺序方法。Decic可以用来计算ode15i的一致初始条件。

关于求解器的详细信息在L. F. Shampine和M. W. Reichelt中给出，MATLAB ODE套件，SIAM科学计算杂志，1997年第18卷，第1-22页，DOI: https://doi.org/10.1137/S1064827594276424。

: [t, y] = ode45 (fcn, trange, init)

: [t, y] = ode45 (fcn, trange, init, ode\_opt)

: [t, y, te, ye, ie] = ode45 (…)

: solution = ode45 (…)

: ode45 (…)

用众所周知的4阶显式Dormand-Prince方法求解一组非刚性常微分方程。

fcn是一个函数句柄、内联函数或包含定义ODE的函数名的字符串:y' = f(t,y)。函数必须接受两个输入，第一个是时间t，第二个是未知的列向量y。

range指定计算ODE的时间间隔。通常，它是一个双元素向量，指定初始和最终时间([tinit, tfinal])。如果有两个以上的元素，那么解决方案也将在这些中间时间实例中进行评估。

默认情况下，ode45使用integrate\_adaptive算法的自适应时间步长。时间步长计算的容差可以通过使用“RelTol”和“AbsTol”选项来改变。

Init包含未知的初始值。如果它是一个行向量，那么解y将是一个矩阵，其中每一列都是init中对应初始值的解。

可选的第四个参数ode\_opt为ODE求解器指定非默认选项。它是由odeset生成的结构。

该函数通常返回两个输出。变量t是一个列向量，它包含了找到解的时间。输出y是一个矩阵，其中每一列表示问题的一个不同的未知数，每一行对应于t中的一个时间。

输出也可以作为结构解决方案返回，其中字段x包含求解时间的行向量，字段y包含解决方案矩阵，以便每列对应x中的时间。使用fieldnames (solution)查看返回的其他字段和其他信息。

如果没有请求输出参数，并且在ode\_opt中没有指定“OutputFcn”，则将“OutputFcn”设置为odeploy，并立即绘制求解器的结果。

如果使用“Events”选项，则可能返回三个额外的输出。保存Event函数返回零的时间。Ye表示时刻t的解的值。ie包含一个索引，指示在多个Event函数的情况下哪个Event函数被触发。

例:解范德波尔方程

fvdp = @(t,y) [y(2); (1 - y(1)^2) \* y(2) - y(1)];

[t,y] = ode45 (fvdp, [0, 20], [2, 0]);

参见:odeset, odeget, ode23, ode15。

: [t, y] = ode23 (fcn, trange, init)

: [t, y] = ode23 (fcn, trange, init, ode\_opt)

: [t, y, te, ye, ie] = ode23 (…)

: solution = ode23 (…)

: ode23 (…)

用众所周知的3阶显式bogacki - shammpine方法求解一组非刚性常微分方程(non-stiff ODEs)。

fcn是一个函数句柄、内联函数或包含定义ODE的函数名的字符串:y' = f(t,y)。函数必须接受两个输入，第一个是时间t，第二个是未知的列向量y。

range指定计算ODE的时间间隔。通常，它是一个双元素向量，指定初始和最终时间([tinit, tfinal])。如果有两个以上的元素，那么解决方案也将在这些中间时间实例中进行评估。

默认情况下，ode23使用integrate\_adaptive算法的自适应时间步长。时间步长计算的容差可以通过使用“RelTol”和“AbsTol”选项来改变。

Init包含未知的初始值。如果它是一个行向量，那么解y将是一个矩阵，其中每一列都是init中对应初始值的解。

可选的第四个参数ode\_opt为ODE求解器指定非默认选项。它是由odeset生成的结构。

该函数通常返回两个输出。变量t是一个列向量，它包含了找到解的时间。输出y是一个矩阵，其中每一列表示问题的一个不同的未知数，每一行对应于t中的一个时间。

输出也可以作为结构解决方案返回，其中字段x包含求解时间的行向量，字段y包含解决方案矩阵，以便每列对应x中的时间。使用fieldnames (solution)查看返回的其他字段和其他信息。

如果没有请求输出参数，并且在ode\_opt中没有指定“OutputFcn”，则将“OutputFcn”设置为odeploy，并立即绘制求解器的结果。

如果使用“Events”选项，则可能返回三个额外的输出。保存Event函数返回零的时间。Ye表示时刻t的解的值。ie包含一个索引，指示在多个Event函数的情况下哪个Event函数被触发。

例:解范德波尔方程

fvdp = @(t,y) [y(2); (1 - y(1)^2) \* y(2) - y(1)];

[t,y] = ode23 (fvdp, [0, 20], [2, 0]);

参考:关于此方法的定义，请参见https://en.wikipedia.org/wiki/List\_of\_Runge%E2%80%93Kutta\_methods。

参见:odeset, odeget, ode45, ode15。

: [t, y] = ode23s (fcn, trange, init)

: [t, y] = ode23s (fcn, trange, init, ode\_opt)

: [t, y] = ode23s (…, par1, par2, …)

: [t, y, te, ye, ie] = ode23s (…)

: solution = ode23s (…)

用(2,3)阶的Rosenbrock方法求解一组刚性常微分方程(stiff ode)。

fcn是一个函数句柄、内联函数或包含定义ODE的函数名的字符串:M y' = f(t,y)。函数必须接受两个输入，其中第一个是时间t，第二个是未知的列向量y。M是一个恒定质量矩阵，非奇异并且可能是稀疏的。在odeopts中设置字段“Mass”，使用odeset指定质量矩阵。

range指定计算ODE的时间间隔。通常，它是一个双元素向量，指定初始和最终时间([tinit, tfinal])。如果有两个以上的元素，那么也将在这些中间时间实例中使用与求解器相同阶次的插值程序来评估解。

默认情况下，ode23使用integrate\_adaptive算法的自适应时间步长。时间步长计算的容差可以通过使用“RelTol”和“AbsTol”选项来改变。

Init包含未知的初始值。如果它是一个行向量，那么解y将是一个矩阵，其中每一列都是init中对应初始值的解。

可选的第四个参数ode\_opt为ODE求解器指定非默认选项。它是由odeset生成的结构。ode23将忽略以下选项:“BDF”，“InitialSlope”，“MassSingular”，“MStateDependence”，“MvPattern”，“MaxOrder”，“Non-negative”。

该函数通常返回两个输出。变量t是一个列向量，它包含了找到解的时间。输出y是一个矩阵，其中每一列表示问题的不同未知数，每行对应于t中的一个时间。如果range指定了中间时间步长，则只返回这些中间时间步长。

输出也可以作为结构解决方案返回，其中字段x包含求解时间的行向量，字段y包含解决方案矩阵，以便每列对应x中的时间。使用fieldnames (solution)查看返回的其他字段和其他信息。

如果使用“Events”选项，则可能返回三个额外的输出。保存Event函数返回零的时间。Ye表示时刻t的解的值。ie包含一个索引，指示在多个Event函数的情况下哪个Event函数被触发。

例:解僵硬的范德波尔方程

f = @(t,y) [y(2); 1000\*(1 - y(1)^2) \* y(2) - y(1)];

opt = odeset ('Mass', [1 0; 0 1], 'MaxStep', 1e-1);

[vt, vy] = ode23s (f, [0 2000], [2 0], opt);

参见:odeset, daspk, dassl。

: [t, y] = ode15s (fcn, trange, y0)

: [t, y] = ode15s (fcn, trange, y0, ode\_opt)

: [t, y, te, ye, ie] = ode15s (…)

: solution = ode15s (…)

: ode15s (…)

求解一组刚性常微分方程(ode)或刚性半显式指标1微分代数方程(DAEs)。

ode15s使用可变步长、可变阶数的BDF(向后微分公式)方法，其阶数从1到5不等。

fcn是一个函数句柄、内联函数或包含定义ODE的函数名的字符串:y' = f(t,y)。函数必须接受两个输入，第一个是时间t，第二个是未知的列向量y。

range指定计算ODE的时间间隔。通常，它是一个双元素向量，指定初始和最终时间([tinit, tfinal])。如果有两个以上的元素，那么解决方案也将在这些中间时间实例中进行评估。

Init包含未知的初始值。如果它是一个行向量，那么解y将是一个矩阵，其中每一列都是init中对应初始值的解。

可选的第四个参数ode\_opt为ODE求解器指定非默认选项。它是由odeset生成的结构。

该函数通常返回两个输出。变量t是一个列向量，它包含了找到解的时间。输出y是一个矩阵，其中每一列表示问题的一个不同的未知数，每一行对应于t中的一个时间。

输出也可以作为结构解决方案返回，其中字段x包含求解时间的行向量，字段y包含解决方案矩阵，以便每列对应x中的时间。使用fieldnames (solution)查看返回的其他字段和其他信息。

如果没有请求输出参数，并且在ode\_opt中没有指定“OutputFcn”，则将“OutputFcn”设置为odeploy，并立即绘制求解器的结果。

如果使用“Events”选项，则可能返回三个额外的输出。保存Event函数返回零的时间。Ye表示时刻t的解的值。ie包含一个索引，指示在多个Event函数的情况下哪个Event函数被触发。

例子:解罗伯逊方程:

function r = robertson\_dae (t, y)

r = [ -0.04\*y(1) + 1e4\*y(2)\*y(3)

0.04\*y(1) - 1e4\*y(2)\*y(3) - 3e7\*y(2)^2

y(1) + y(2) + y(3) - 1 ];

endfunction

opt = odeset ("Mass", [1 0 0; 0 1 0; 0 0 0], "MStateDependence", "none");

[t,y] = ode15s (@robertson\_dae, [0, 1e3], [1; 0; 0], opt);

参见:decic, odeset, odeget, ode23, ode45。

: [t, y] = ode15i (fcn, trange, y0, yp0)

: [t, y] = ode15i (fcn, trange, y0, yp0, ode\_opt)

: [t, y, te, ye, ie] = ode15i (…)

: solution = ode15i (…)

: ode15i (…)

求解一组全隐式常微分方程(ode)或指标1微分代数方程(DAEs)。

ode15i使用了一个变步长、变阶的BDF(反向微分公式)方法，其阶数从1到5不等。

fcn是一个函数句柄、内联函数或包含定义ODE的函数名的字符串:0 = f(t,y,yp)。函数必须接受三个输入，其中第一个是时间t，第二个是函数值y(列向量)，第三个是导数值yp(列向量)。

range指定计算ODE的时间间隔。通常，它是一个双元素向量，指定初始和最终时间([tinit, tfinal])。如果有两个以上的元素，那么解决方案也将在这些中间时间实例中进行评估。

Y0和yp0包含未知y和yp的初始值。如果它们是行向量，那么解y就是一个矩阵，其中每一列都是y0和yp0中对应初值的解。

Y0和yp0必须是一致的初始条件，即满足f(t, Y0,yp0) = 0。函数decic可以用来计算给定初始猜测的一致初始条件。

可选的第五个参数ode\_opt为ODE求解器指定非默认选项。它是由odeset生成的结构。

该函数通常返回两个输出。变量t是一个列向量，它包含了找到解的时间。输出y是一个矩阵，其中每一列表示问题的一个不同的未知数，每一行对应于t中的一个时间。

输出也可以作为结构解决方案返回，其中字段x包含求解时间的行向量，字段y包含解决方案矩阵，以便每列对应x中的时间。使用fieldnames (solution)查看返回的其他字段和其他信息。

如果没有请求输出参数，并且在ode\_opt中没有指定“OutputFcn”，则将“OutputFcn”设置为odeploy，并立即绘制求解器的结果。

如果使用“Events”选项，则可能返回三个额外的输出。保存Event函数返回零的时间。Ye表示时刻t的解的值。ie包含一个索引，指示在多个Event函数的情况下哪个Event函数被触发。

例子:解罗伯逊方程:

function r = robertson\_dae (t, y, yp)

r = [ -(yp(1) + 0.04\*y(1) - 1e4\*y(2)\*y(3))

-(yp(2) - 0.04\*y(1) + 1e4\*y(2)\*y(3) + 3e7\*y(2)^2)

y(1) + y(2) + y(3) - 1 ];

endfunction

[t,y] = ode15i (@robertson\_dae, [0, 1e3], [1; 0; 0], [-1e-4; 1e-4; 0]);

参见:decic, odeset, odeget。

: [y0\_new, yp0\_new] = decic (fcn, t0, y0, fixed\_y0, yp0, fixed\_yp0)

: [y0\_new, yp0\_new] = decic (fcn, t0, y0, fixed\_y0, yp0, fixed\_yp0, options)

: [y0\_new, yp0\_new, resnorm] = decic (…)

给定初始猜测值y0和yp0，计算一致的隐式ODE初始条件y0\_new和yp0\_new。

fixed\_y0和fixed\_yp0之间的最大长度(y0)分量可以选择为固定值。

FCN是函数句柄。函数必须接受三个输入，第一个是时间t，第二个是未知的列向量y，第三个是未知的列向量yp。

T0是fcn(T0, y0\_new, yp0\_new) = 0的初始时间，以标量形式指定。

Y0是一个向量，用作y的初始猜测。

Fixed\_y0是一个向量，它指定y0的组成部分保持固定。选择fixed\_y0和fixed\_yp0之间长度(y0)个分量的最大值作为固定值。如果需要固定y0(i)的值，则将fixed\_y0(i)组件设置为1。如果您希望允许更改y0(i)的值，则将fixed\_y0(i)组件设置为0。

Yp0是一个向量，用作yp的初始猜测值。

Fixed\_yp0是一个向量，它指定yp0中保持固定的分量。选择fixed\_y0和fixed\_yp0之间的最大长度(yp0)个组件作为固定值。如果需要固定yp0(i)的值，将fixed\_yp0(i) component设置为1。如果希望允许修改yp0(i)的值，则将fixed\_yp0(i)组件设置为0。

可选的第七个参数options是一个结构数组。使用odeset来生成这个结构。相关的选项是RelTol和AbsTol，它们指定用于计算初始条件的错误阈值。

该函数通常返回两个输出。变量y0\_new为列向量，包含y的一致初始值。输出yp0\_new为列向量，包含yp的一致初始值。

可选的第三个输出resnorm是残差向量的范数。如果resnorm很小，则表示decic成功地计算了初始条件。如果resnorm值较大，请使用RelTol和AbsTol进行调整。

例:计算罗伯逊方程的初始条件:

function r = robertson\_dae (t, y, yp)

r = [ -(yp(1) + 0.04\*y(1) - 1e4\*y(2)\*y(3))

-(yp(2) - 0.04\*y(1) + 1e4\*y(2)\*y(3) + 3e7\*y(2)^2)

y(1) + y(2) + y(3) - 1 ];

endfunction

[y0\_new,yp0\_new] = decic (@robertson\_dae, 0, [1; 0; 0], [1; 1; 0],

[-1e-4; 1; 0], [0; 0; 0]);

参见:ode15i, odeset。

: odestruct = odeset ()

: odestruct = odeset ("field1", value1, "field2", value2, …)

: odestruct = odeset (oldstruct, "field1", value1, "field2", value2, …)

: odestruct = odeset (oldstruct, newstruct)

: odeset ()

创建或修改ODE选项结构。

在没有输入参数和一个输出参数的情况下调用时，返回一个新的ODE选项结构，其中包含初始化为其默认值的所有可能字段。如果未请求输出参数，则显示常见ODE求解器选项列表及其默认值。

如果使用名称-值输入参数对“field1”，“value1”，“field2”，“value2”，…调用，则返回一个新的ODE选项结构，其中初始化了所有最常见的选项字段，并将字段“field1”，“field2”，…的值设置为值value1, value2， ....

如果使用输入结构oldstruct调用，则用新值value1, value2，…覆盖选项"field1"， "field2"，…的值，并返回修改后的结构。

当使用两个输入ODE选项结构oldstruct和newstruct调用时，用结构newstruct中的新值覆盖结构oldstruct中的所有值。newstruct中的空值不会覆盖oldstruct中的值。

最常用的ODE选项如下，它们总是由odeset赋值:

AbsTol: positive scalar | vector, def. 1e-6

绝对误差容忍。

BDF: {"off"} | "on"

在隐式多步骤方法中使用BDF公式。注意:此选项尚未实现。

Events: function\_handle

事件函数。事件函数必须具有如下形式[value, terminal, direction] = my\_events\_f (t, y)

InitialSlope: vector

DAE解算器的一致初始斜率向量。

InitialStep: positive scalar

初始时间步长。

Jacobian: matrix | function\_handle

雅可比矩阵，指定为常数矩阵或时间和状态的函数。

JConstant: {"off"} | "on"

指定雅可比矩阵是常数矩阵还是依赖于状态。

JPattern: sparse matrix

如果雅可比矩阵是稀疏且非恒定的，但保持恒定的稀疏模式，则指定稀疏模式。

Mass: matrix | function\_handle

质量矩阵，指定为常数矩阵或时间和状态的函数。

MassSingular: {"maybe"} | "yes" | "on"

指定质量矩阵是否奇异。

MaxOrder: {5} | 4 | 3 | 2 | 1

公式的最大阶数。

MaxStep: positive scalar

最大时间步长值。

MStateDependence: {"weak"} | "none" | "strong"

指定质量矩阵是依赖于状态还是仅仅依赖于时间。

MvPattern: sparse matrix

如果质量矩阵是稀疏且非恒定的，但保持恒定的稀疏模式，则指定稀疏模式。注意:此选项尚未实现。

NonNegative: scalar | vector

指定状态向量中在模拟期间保持非负的元素。

NormControl: {"off"} | "on"

相对于解的2范数的控制误差，而不是它的绝对值。

OutputFcn: function\_handle

在模拟过程中监控状态的功能。有关要使用的函数的形式，请参阅odeploy。

OutputSel: scalar | vector

要传递给输出监视函数的状态向量元素的索引。

Refine: positive scalar

指定是只在每个时间步骤结束时返回输出，还是在中间时间实例时也返回输出。该值应该是一个标量，指示在返回输出的每个时间步长中使用的等间隔时间点的数量。

RelTol: positive scalar

相对容错。

Stats: {"off"} | "on"

模拟后打印求解器统计信息。

Vectorized: {"off"} | "on"

指定odefcn是否可以一次传递多个状态值。

不在上述列表中的字段名也被接受并添加到结果结构中。

参见:odeget。

: val = odeget (ode\_opt, field)

: val = odeget (ode\_opt, field, default)

查询ODE选项结构ode\_opt中属性字段的值。

如果用两个输入参数调用，第一个输入参数ode\_opt是一个ODE选项结构，第二个输入参数字段是一个指定选项名称的字符串，那么返回与ode\_opt字段对应的选项值val。

如果使用可选的第三个输入参数调用，并且字段没有在ode\_opt结构中设置，则返回默认值default。

参见odeset。

: stop\_solve = odeplot (t, y, flag)

打开一个新的图形窗口，并在集成期间的每个时间步骤绘制一个代码问题的解决方案。

输入参数t和y的类型和值取决于字符串类型的输入标志。flag的有效值为:

"init"

输入t必须是长度为2的列向量，其第一个和最后一个时间步长为[tfirst tlast]。输入y包含ode问题(y0)的初始条件。

""

输入t必须是标量双精度或矢量，指定计算输入y中的解的时间(s)。

"done"

输入应该为空，但如果存在则忽略它们。

odeploy总是返回false，也就是说，不要停止ode求解器。

示例:求解“Van der Pol”方程的匿名实现，并在求解时显示结果。

fvdp = @(t,y) [y(2); (1 - y(1)^2) \* y(2) - y(1)];

opt = odeset ("OutputFcn", @odeplot, "RelTol", 1e-6);

sol = ode45 (fvdp, [0 20], [2 0], opt);

背景信息:如果在用odeset创建的选项结构的“OutputFcn”属性中指定了该函数，则该函数由ode求解器函数调用。ode求解器最初将使用odeploy ([tfirst, tlast]， y0， "init")语法调用该函数。该函数初始化内部变量，创建一个新的图形窗口，并设置绘图的x限制。随后，在集成期间的每个时间步，代码求解器调用odeploy (t, y，[])。在解决方案的最后，代码求解器调用odeploy([]，[]，“done”)，以便odeploy可以执行所需的任何清理操作。

参见:odeset, odeget, ode23, ode45。

**25优化**

Octave支持解决各种优化问题。具体来说，Octave可以解决线性规划、二次规划、非线性规划和线性最小二乘最小化等问题。

**25.1线性规划**

Octave可以使用glpk函数解决线性规划问题。也就是说，Octave可以解决

min C'\*x

服从线性约束A\*x = b，其中x≥0。

glpk函数也支持这个问题的变体。

: [xopt, fmin, errnum, extra] = glpk (c, A, b, lb, ub, ctype, vartype, sense, param)

使用GNU GLPK库求解一个线性程序。

给定三个参数，glpk解决以下标准LP:

min C'\*x

受制于

A\*x = b

x >= 0

但也可能解决问题的形式

[ min | max ] C'\*x

受制于

A\*x [ "=" | "<=" | ">=" ] b

x >= LB

x <= UB

输入参数

c

包含目标函数系数的列数组。

A

包含约束系数的矩阵。

b

包含约束矩阵中每个约束的右侧值的列数组。

lb

包含每个变量的下界的数组。如果未提供lb，则变量的默认下界为零。

ub

包含每个变量的上界的数组。如果未提供ub，则假定默认上界为无穷大。

ctype

包含约束矩阵中每个约束意义的字符数组。数组的每个元素可以是下列值之一

“F”

自由(无界)约束(该约束被忽略)。

“U”

一个有上界的不等式约束(A(i，:)\*x <= b(i))。

“S”

等式约束(A(i，:)\*x = b(i))。

“L”

一个有下界的不等式(a (i，:)\*x >= b(i))。

“D”

具有上界和下界的不等式约束(A(i，:)\*x >= -b(i))和(A(i，:)\*x <= b(i))。

vartype

包含变量类型的列数组。

“C”

一个连续变量。

“I”

整数变量。

sense

如果sense为1，问题就是最小化。如果sense是-1，问题就是最大化。缺省值为1。

param

包含以下参数的结构，用于定义求解器的行为。结构中缺少的元素采用默认值，因此您只需要设置希望从默认值更改的元素。

整数参数:

msglev (default: 1)

求解器例程输出的消息级别:

0 (GLP\_MSG\_OFF)

没有输出。

1 (GLP\_MSG\_ERR)

只有错误和警告消息。

2 (GLP\_MSG\_ON)

正常输出。

3 (GLP\_MSG\_ALL)

完整输出(包括信息消息)。

scale (default: 16)

扩展选项。这些值可以与位或运算符组合，可以是:

1 (GLP\_SF\_GM)

几何平均缩放。

16 (GLP\_SF\_EQ)

平衡扩展。

32 (GLP\_SF\_2N)

将比例因子四舍五入到2的次方。

64 (GLP\_SF\_SKIP)

如果问题规模很好，则跳过。

或者，也可以指定值128 (GLP\_SF\_AUTO)，在这种情况下，例程会自动选择缩放选项。

dual (default: 1)

单纯形方法选项:

1 (GLP\_PRIMAL)

使用两相原始单纯形。

2 (GLP\_DUALP)

使用两相双单纯形，如果失败，切换到原始单纯形。

3 (GLP\_DUAL)

采用两相双单形。

price (default: 34)

定价选项(对于原始和双单形):

17 (GLP\_PT\_STD)

教科书定价。

34 (GLP\_PT\_PSE)

最陡边缘定价。

itlim (default: intmax)

单纯形迭代限制。当执行一次单纯形迭代时，它每次减少1，并且达到零值表示求解器停止搜索。

outfrq (default: 200)

输出频率，以迭代为单位。此参数指定求解器将有关解决方案的信息发送到标准输出的频率。

branch (default: 4)

分支技术选项(仅适用于MIP):

1 (GLP\_BR\_FFV)

第一个分数变量。

2 (GLP\_BR\_LFV)

最后一个分数变量。

3 (GLP\_BR\_MFV)

大多数分数变量。

4 (GLP\_BR\_DTH)

Driebeck和Tomlin的启发式。

5 (GLP\_BR\_PCH)

混合伪成本启发式。

btrack (default: 4)

回溯技术选项(仅针对MIP):

1 (GLP\_BT\_DFS)

深度优先搜索。

2 (GLP\_BT\_BFS)

广度优先搜索。

3 (GLP\_BT\_BLB)

最佳局部边界。

4 (GLP\_BT\_BPH)

最佳投影启发式。

presol (default: 1)

如果设置了此标志，则单纯形解算器使用内置的LP解算器。否则不使用LP解析器。

lpsolver (default: 1)

选择要使用的求解器。如果问题是MIP问题，这个标志将被忽略。

1

修正单纯形法。

2

内点法。

rtest (default: 34)

比值测试技术:

17 (GLP\_RT\_STD)

标准(“教科书”)。

34 (GLP\_RT\_HAR)

哈里斯的两通比率测试。

tmlim (default: intmax)

搜索时间限制，以毫秒为单位。

outdly (default: 0)

输出延迟，以秒为单位。此参数指定求解器应该延迟多长时间将有关解决方案的信息发送到标准输出。

save (default: 0)

如果此参数为非零，则以CPLEX LP格式保存问题的副本到文件"outpb.lp"。目前没有办法更改输出文件的名称。

真正的参数:

tolbnd (default: 1e-7)

相对公差用于检查当前基本解是否基本可行。不建议更改此参数，除非您已经详细了解了其目的。

toldj (default: 1e-7)

绝对容差用来检查当前基本解是否对偶可行。不建议更改此参数，除非您已经详细了解了其目的。

tolpiv (default: 1e-10)

相对公差用于选择单纯形表中合适的关键元素。不建议更改此参数，除非您已经详细了解了其目的。

objll (default: -DBL\_MAX)

目标函数的下限。如果目标函数达到这个极限并继续减小，求解器停止搜索。该参数仅在双单纯形法中使用。

objul (default: +DBL\_MAX)

目标函数的上限。如果目标函数达到这个极限并继续增加，求解器停止搜索。该参数仅在双单纯形中使用。

tolint (default: 1e-5)

相对容差用于检查当前基本解是否整数可行。不建议更改此参数，除非您已经详细了解了其目的。

tolobj (default: 1e-7)

相对容差用于检查目标函数的值是否优于已知的最佳整数可行解。不建议更改此参数，除非您已经详细了解了其目的。

输出值：

xopt

优化器(决策变量的最优值)。

fopt

目标函数的最优值。

errnum

错误代码。

0

没有错误。

1 (GLP\_EBADB)

无效的基础。

2 (GLP\_ESING)

奇异矩阵。

3 (GLP\_ECOND)

坏脾气的矩阵。

4 (GLP\_EBOUND)

无效的范围。

5 (GLP\_EFAIL)

解算器失败。

6 (GLP\_EOBJLL)

达到目标函数下限。

7 (GLP\_EOBJUL)

达到目标函数上限。

8 (GLP\_EITLIM)

迭代限制耗尽。

9 (GLP\_ETMLIM)

时间限制已用尽。

10 (GLP\_ENOPFS)

没有原始可行解。

11 (GLP\_ENODFS)

没有对偶可行解。

12 (GLP\_EROOT)

没有提供最优的根LP。

13 (GLP\_ESTOP)

按应用程序终止搜索。

14 (GLP\_EMIPGAP)

达到相对MIP间隙公差。

15 (GLP\_ENOFEAS)

没有原始/对偶可行解。

16 (GLP\_ENOCVG)

不收敛。

17 (GLP\_EINSTAB)

数值不稳定。

18 (GLP\_EDATA)

无效的数据。

19 (GLP\_ERANGE)

结果超出范围。

extra

包含以下字段的数据结构:

lambda

对偶变量

redcosts

缩减成本

time

用于解决LP/MIP问题的时间(秒)。

status

优化状态。

1 (GLP\_UNDEF)

解决方案状态未定义。

2 (GLP\_FEAS)

解决方案是可行的。

3 (GLP\_INFEAS)

解决方案是不可行的。

4 (GLP\_NOFEAS)

这个问题没有可行的解决办法。

5 (GLP\_OPT)

解是最优的。

6 (GLP\_UNBND)

问题没有无界解。

例子:

c = [10, 6, 4]';

A = [ 1, 1, 1;

10, 4, 5;

2, 2, 6];

b = [100, 600, 300]';

lb = [0, 0, 0]';

ub = [];

ctype = "UUU";

vartype = "CCC";

s = -1;

param.msglev = 1;

param.itlim = 100;

[xmin, fmin, status, extra] = ...

glpk (c, A, b, lb, ub, ctype, vartype, s, param);

**25.2二次规划**

八度也可以解决二次规划问题，这是

min 0.5 x'\*H\*x + x'\*q

受制于

A\*x = b

lb <= x <= ub

A\_lb <= A\_in\*x <= A\_ub

: [x, obj, info, lambda] = qp (x0, H)

: [x, obj, info, lambda] = qp (x0, H, q)

: [x, obj, info, lambda] = qp (x0, H, q, A, b)

: [x, obj, info, lambda] = qp (x0, H, q, A, b, lb, ub)

: [x, obj, info, lambda] = qp (x0, H, q, A, b, lb, ub, A\_lb, A\_in, A\_ub)

: [x, obj, info, lambda] = qp (…, options)

解一个二次规划(QP)。

求解由定义的二次规划

min 0.5 x'\*H\*x + x'\*q

x

受制于

A\*x = b

lb <= x <= ub

A\_lb <= A\_in\*x <= A\_ub

使用零空间活动集方法。

任何边界(A, b, lb, ub, A\_in, A\_lb, A\_ub)如果不存在，可以设置为空矩阵([])。约束A和A\_in是矩阵，每一行表示一个约束。其他边界是标量或向量取决于约束的数量。如果初始猜测是可行的，算法会更快。

Options是一个指定控制算法的附加参数的结构。目前，qp识别这些选项:“MaxIter”，“TolX”。

“MaxIter”禁止在优化停止之前算法迭代的最大次数。缺省值是200。取值必须为正整数。

“TolX”指定未知变量x的终止公差。默认值是sqrt (eps)或大约1e-8。

返回时，x是最小值的位置，fval包含目标函数在x处的值。

info

包含算法的运行时信息的结构。定义如下字段:

solveiter

找到解决方案所需的迭代次数。

info

整数，表示解决方案的状态。

0

这个问题是可行的和凸的。找到全球解决方案。

1

这个问题不是凸的。找到本地解决方案。

2

这个问题不是凸的和无界的。

3

达到的最大迭代数。

6

这个问题不可行。

参见:sqp。

: x = pqpnonneg (c, d)

: x = pqpnonneg (c, d, x0)

: x = pqpnonneg (c, d, x0, options)

: [x, minval] = pqpnonneg (…)

: [x, minval, exitflag] = pqpnonneg (…)

: [x, minval, exitflag, output] = pqpnonneg (…)

: [x, minval, exitflag, output, lambda] = pqpnonneg (…)

最小化(1/2 \* x' \* c \* x + d' \* x)，前提是x >= 0。

C和d必须是实矩阵，C必须是对称的正定矩阵。

X0是解x的可选初始猜测值。

Options是一个选项结构，用于改变算法的行为(参见optimset)。pqpnonneg识别一个选项:“MaxIter”。

输出:

x

解矩阵

minval

获得的最小模型值为1/2\*xmin'\*c\*xmin + d'\*xmin

exitflag

趋同的标志。0表示超过迭代次数，未达到收敛;>0表示算法收敛。(该算法是稳定的，并且在足够的迭代下会收敛。)

output

有两个字段的结构:

* "algorithm":使用的算法("nnls")
* “iterations”:迭代的次数。

lambda

拉格朗日乘数法。如果这些值不为零，对应的x值应该为零，表示解被压在一个坐标平面上。大小表示如果x >= 0约束在该方向上放松，残差将改善多少。

参见:lsqnonnegg, qp, optimset。

**25.3非线性规划**

使用连续二次规划求解器，Octave也可以执行一般的非线性最小化。

: [x, obj, info, iter, nf, lambda] = sqp (x0, phi)

: […] = sqp (x0, phi, g)

: […] = sqp (x0, phi, g, h)

: […] = sqp (x0, phi, g, h, lb, ub)

: […] = sqp (x0, phi, g, h, lb, ub, maxiter)

: […] = sqp (x0, phi, g, h, lb, ub, maxiter, tolerance)

最小化目标函数使用顺序二次规划(SQP)。

求解非线性程序

min phi (x)

x

受制于

g(x) = 0

h(x) >= 0

lb <= x <= ub

采用顺序二次规划方法。

第一个参数是对向量x0的初始猜测。

第二个参数是指向目标函数的函数句柄。目标函数必须接受一个矢量参数并返回一个标量。

第二个参数也可以是包含函数句柄的2或3个元素的单元数组。第一个元素应该指向目标函数，第二个元素应该指向计算目标函数梯度的函数，第三个元素应该指向计算目标函数的黑森系数的函数。如果没有提供梯度函数，则通过有限差分计算梯度。如果没有提供Hessian函数，则使用BFGS更新公式来近似Hessian。

当提供时，梯度函数phi{2}必须接受一个矢量参数并返回一个矢量。当提供时，Hessian函数phi{3}必须接受一个矢量参数并返回一个矩阵。

第三和第四个参数g和h是函数句柄，分别指向计算等式约束和不等式约束的函数。如果问题没有相等(或不等式)约束，则对g(或h)使用空矩阵([])。当提供时，这些相等和不等式约束函数必须接受一个矢量参数并返回一个矢量。

第三和第四个参数也可以是函数句柄的双元素单元数组。第一个元素应该指向约束函数，第二个元素应该指向计算约束函数梯度的函数:

[ d f(x) d f(x) d f(x) ]

transpose ( [ ------ ----- ... ------ ] )

[ dx\_1 dx\_2 dx\_N ]

第五个和第六个参数lb和ub包含x的下界和上界。它们必须与等式和不等式约束g和h一致。如果参数是向量，则x(i)受lb(i)和ub(i)的约束。边界也可以是一个标量，在这种情况下，x的所有元素将共享相同的边界。

第七个参数maxiter指定迭代的最大次数。缺省值是100。

第八个参数tolerance指定停止标准的容忍度。默认值为sqrt (eps)。

info返回的值可能是下列值之一:

101

算法正常终止。所有约束均满足指定公差。

102

日志含义更新BFGS失败。

103

日志含义达到最大迭代次数。

104

步长变得太小，即delta x小于tol \* norm (x)。

调用sqp的一个例子:

function r = g (x)

r = [ sumsq(x)-10;

x(2)\*x(3)-5\*x(4)\*x(5);

x(1)^3+x(2)^3+1 ];

endfunction

function obj = phi (x)

obj = exp (prod (x)) - 0.5\*(x(1)^3+x(2)^3+1)^2;

endfunction

x0 = [-1.8; 1.7; 1.9; -0.8; -0.8];

[x, obj, info, iter, nf, lambda] = sqp (x0, @phi, @g, [])

x =

-1.71714

1.59571

1.82725

-0.76364

-0.76364

obj = 0.053950

info = 101

iter = 8

nf = 10

lambda =

-0.0401627

0.0379578

-0.0052227

参见:qp。

**25.4线性最小二乘**

Octave还支持线性最小二乘最小化。也就是说，Octave可以找到参数b，使模型y = x\*b尽可能地拟合数据(x,y)，假设高斯噪声为零均值。如果假设噪声是各向同性的，则可以使用' \ '或' / '运算符或ols函数来解决问题。在假定噪声是各向异性的一般情况下，需要玻璃。

: [beta, sigma, r] = ols (y, x)

普通最小二乘估计。

OLS适用于多元模型y = x\*b + e其中y是一个t × p矩阵，x是一个t × k矩阵，b是一个k × p矩阵，e是一个t × p矩阵。

y的每一行是p变量观测值，其中每一列代表一个变量。同样，x的行表示k变量观察值或可能的设计值。此外，观测值x的集合必须有足够的秩k，否则b不能唯一估计。

假设观测误差e来源于一个隐含的p变量分布，其均值为零，p × p协方差矩阵S都是常数，条件都是x。此外，矩阵S相对于每个观测值是常数，使得均值(e) = 0, cov (vec (e)) = kron (S, I)。(对于不符合这一标准的情况，如自相关误差，请参见广义最小二乘，gls，以获得更有效的估计。)

贝塔、西格玛和r的返回值定义如下:

beta

如果矩阵x'\*x是满秩的，则直接通过inv (x'\*x) \*x '\* y计算矩阵b的OLS估计量。否则，β = pinv (x) \* y，其中pinv (x)表示x的伪逆。

sigma

矩阵s的OLS估计量，

sigma = (y-x\*beta)' \* (y-x\*beta) / (t-rank(x))

r

OLS残差矩阵r = y - x\*。

参见:gls, pinv。

: [beta, v, r] = gls (y, x, o)

广义最小二乘(GLS)模型。

对多元模型y = x\*B + E进行广义最小二乘估计，其中y是t × p矩阵，x是t × k矩阵，B是k × p矩阵，E是t × p矩阵。

y的每一行是p变量观测值，其中每一列代表一个变量。同样，x的行表示k变量观察值或可能的设计值。此外，观测值x的集合必须有足够的秩k，否则b不能唯一估计。

假设观测误差e来自一个基本的p变量分布，其平均值为零，但可能是异方差观测值。也就是说，一般来说，mean (e) = 0, cov (vec (e)) = (s²)\* 0，其中s是一个标量，0是一个t\*p × t\*p矩阵。

返回值beta, v和r定义如下。

Beta

矩阵b的GLS估计量。

v

标量s^2的GLS估计量。

r

GLS残差矩阵，r = y - x\*。

参见:ols。

: x = lsqnonneg (c, d)

: x = lsqnonneg (c, d, x0)

: x = lsqnonneg (c, d, x0, options)

: [x, resnorm] = lsqnonneg (…)

: [x, resnorm, residual] = lsqnonneg (…)

: [x, resnorm, residual, exitflag] = lsqnonneg (…)

: [x, resnorm, residual, exitflag, output] = lsqnonneg (…)

: [x, resnorm, residual, exitflag, output, lambda] = lsqnonneg (…)

使范数(c\*x - d)在x >= 0时最小。

C和d必须是实矩阵。

X0是解x的可选初始猜测值。

Options是一个选项结构，用于改变算法的行为(参见optimset)。lsqnonneg识别这些选项:"MaxIter"， "TolX"。

输出:

resnorm

残差的2-范数的平方，范数(c\*x-d)^2

Residual

残差:d-c\*x

exitflag

趋同的标志。0表示超过迭代次数，未达到收敛;>0表示算法收敛。(该算法是稳定的，并且在足够的迭代下会收敛。)

output

有两个字段的结构:

* "algorithm":使用的算法("nnls")
* “iterations”:迭代的次数。

lambda

拉格朗日乘数法。如果这些值不为零，对应的x值应该为零，表示解被压在一个坐标平面上。大小表示如果x >= 0约束在该方向上放松，残差将改善多少。

参见:pqpnonneg, lscov, optimset。

: x = lscov (A, b)

: x = lscov (A, b, V)

: x = lscov (A, b, V, alg)

: [x, stdx, mse, S] = lscov (…)

计算一个广义线性最小二乘拟合。

在模型b = Ax + w下估计x，其中假设噪声w服从协方差矩阵{\sigma^2} V的正态分布。

如果系数矩阵A的大小是n × p，则常数项b的向量/数组的大小必须是n × k。

可选的输入参数V可以是一个n元素的正权重向量(反方差)，或者是一个n × n对称的正半定矩阵，表示b的协方差。如果没有提供V，则返回普通最小二乘解。

alg输入参数是关于使用解决方法的指导，目前被忽略。

除了最小二乘估计矩阵x (p-by-k)之外，该函数还返回stdx (p-by-k)，即估计x的误差标准差;Mse (k-by-1)，估计数据误差协方差比例因子(\sigma^2);S (p × p，如果k > 1则为p × p × k)， x的误差协方差。

参考:Golub和Van Loan(1996)，矩阵计算(第三版)，约翰霍普金斯大学，第5.6.3节

参见:ols, gls, lsqnonnegg。

: optimset ()

: options = optimset ()

: options = optimset (par, val, …)

: options = optimset (old, par, val, …)

: options = optimset (old, new)

创建优化函数的选项结构。

在没有任何输入或输出参数的情况下调用optimset时，将打印所有有效优化参数的列表。

当只有一个输出而没有输入时，返回一个选项结构，其中所有有效的选项参数初始化为[]。

当使用参数/值对列表调用时，返回一个只初始化命名参数的选项结构。

如果第一个输入是现有的选项结构old，则从par/val列表或从选项结构new更新值。

如果par与标准参数的名称不完全匹配，则optimset将尝试将par与标准参数进行匹配，并在找到匹配时设置该参数的值。匹配不区分大小写，基于参数名称开头的字符匹配。如果Optimset发现多个不明确的匹配，则产生错误。如果没有找到匹配的标准参数，则发出警告并创建非标准参数。

有效参数的标准列表:

AutoScaling

ComplexEqn

Display

请求详细显示优化结果。值:

"off" [default]

没有显示。

"iter"

显示每次循环迭代的中间结果。

"final"

显示最后一次循环迭代的结果。

"notify"

如果函数未能收敛，则显示最后一次循环迭代的结果。

FinDiffType

FunValCheck

启用后，如果目标函数返回无效值(复数、NaN或Inf)，则显示错误。必须设置为“on”或“off”[default]。注意:函数fzero和fminbind正确处理Inf值，在这种情况下只有复数值或NaN会导致错误。

GradObj

当设置为“on”时，要最小化的函数必须返回第二个参数，该参数是函数在点x处的梯度或一阶导数。如果设置为“off”[默认值]，则通过有限差分计算梯度。

Jacobian

当设置为“on”时，要最小化的函数必须返回第二个参数，该参数是函数在x点的雅可比矩阵或一阶导数。如果设置为“off”[默认值]，则通过有限差分计算雅可比矩阵。

MaxFunEvals

在优化停止之前，函数计算的最大次数。必须是正整数。

MaxIter

停止优化前的最大算法迭代次数。必须是正整数。

OutputFcn

每次算法迭代执行一次的用户定义函数。

TolFun

函数输出的终止准则。如果一次算法迭代与下一次迭代之间计算的目标函数的差值小于TolFun，则优化停止。必须是一个正标量。

TolX

函数输入的终止准则。如果一次算法迭代与下一次迭代之间当前搜索点x的差值小于TolX，则停止优化。必须是一个正标量。

TypicalX

Updating

此列表可以由用户或其他加载的Octave包扩展。可以使用optimset的无参数形式查询更新后的有效参数列表。

注1:在匹配短参数名时，只考虑标准列表中的参数名，即使存在精确的非标准匹配，par也将始终扩展以匹配标准参数。与一个或多个标准参数有歧义的非标准参数的值不能通过optimset设置，只能使用结构体的setfield或点符号来设置。

注2:优化选项结构主要用于通过optimset和optimget操作已知参数。由于创建了非标准或不明确的参数，或者在创建优化选项结构后加载/卸载更改了已知参数列表的包，可能会导致对optimset或optimget的未来调用出现不可预测的行为。

参见:optimget。

: val = optimget (options, par)

: val = optimget (options, par, default)

从optimset创建的优化选项结构options中返回特定参数par的值。

如果par未定义，则返回提供的默认值，否则返回空矩阵。

如果par与标准参数的名称不完全匹配，则optimget将尝试将par与标准参数进行匹配，如果找到匹配，将返回该参数的值。匹配不区分大小写，基于参数名称开头的字符匹配。如果Optimget找到多个不明确的匹配，就会产生错误。如果没有找到匹配的标准参数，则发出警告。有关标准选项列表的信息，请参阅optimset。

注意:在匹配短参数名时，只考虑标准列表中的参数名，并且par将始终扩展以匹配标准参数，即使存在精确的非标准匹配。与一个或多个标准参数有歧义的非标准参数的值不能由optimget返回，只能使用结构体的getfield或点符号来访问。

参见:optimset。

**26统计**

Octave支持各种统计方法。重点是基本的描述性统计，但是Octave Forge统计包包括概率分布、统计测试、随机数生成等等。

分析数据的函数都假定多维数据排列在一个矩阵中，其中每一行是一个观测值，每一列是一个变量。因此，矩阵定义为

a = [ 0.9, 0.7;

0.1, 0.1;

0.5, 0.4 ];

包含来自二维分布的三个观测值。虽然这是默认的数据安排，但大多数函数支持不同的安排。

需要注意的是，统计函数不测试包含NaN、NA或Inf的数据。需要显式地检测和处理这些值。看isnan, isna, isinf, isfinite。

**26.1描述性统计**

描述统计的一个主要目标是简洁地表示大数据集的本质。Octave提供了均值、中位数和模式函数，它们都用一个与数据集中趋势相对应的数字来总结数据集。

: m = mean (x)

: m = mean (x, dim)

: m = mean (x, vecdim)

: m = mean (x, "all")

: m = mean (…, nanflag)

: m = mean (…, outtype)

计算x各元素的均值。

如果x是一个向量，则mean (x)返回x中定义为的元素的平均值

mean (x) = SUM\_i x(i) / N

其中N是x中元素的个数。

如果x是一个数组，那么mean(x)计算x的第一个非单维的平均值。

可选变量dim强制表示在指定的维度上操作，该维度必须是正整数值。在x中指定任何单个维度，包括任何超过ndims (x)的维度，将导致平均值等于x。

将维度指定为vecdim(一个不重复维度的向量)，将返回由vecdim定义的数组切片上的平均值。如果vecdim索引x的所有维度，那么它相当于选项"all"。vecdim中任何大于ndims (x)的维度都会被忽略。

将维度指定为"all"将强制mean对x的所有元素进行操作，并且相当于mean (x(:))。

可选的输入输出类型指定返回的数据类型。Outtype可以采用以下值:

'default' : 输出为double类型，除非输入为single，在这种情况下输出为single类型。

'double' : 输出类型为double。

'native' : 输出与(class (x))报告的输入类型相同，除非输入是逻辑的，在这种情况下输出是double类型。

可选变量nanflag指定使用任何先前指定的输入参数组合在计算中包括或排除NaN值。nanflag的默认值是" inclenan "，它在计算中保留NaN值。要排除NaN值，请将nanflag的值设置为“omitnan”。如果x由操作维度中的所有NaN值组成，则输出仍将包含NaN值。

参见:median, mode, movmean。

: m = median (x)

: m = median (x, dim)

: m = median (x, vecdim)

: m = median (x, "all")

: m = median (…, nanflag)

: m = median (…, outtype)

计算x中元素的中值。

当对x的元素进行排序时，例如s = sort (x)，中位数定义为

| s(ceil (N/2)) N odd

median (x) = |

| (s(N/2) + s(N/2+1))/2 N even

如果x是一个数组，则median (x)沿着x的第一个非单维进行操作。

可选变量dim强制median在指定的维度上操作，该维度必须是一个正整数。在x中指定任何单例维度，包括任何超过ndims (x)的维度，将导致中位数等于x。

将维度指定为vecdim(一个不重复维度的向量)，将返回vecdim定义的数组切片上的中位数。如果vecdim索引x的所有维度，那么它相当于选项"all"。vecdim中任何大于ndims (x)的维度都会被忽略。

将维度指定为“all”将强制median对x的所有元素进行操作，并且相当于median (x(:))。

Median(…，outtype)使用前面语法中的任何输入参数，返回指定数据类型的中位数。Outtype可以采用以下值:

"default"

输出为double类型，除非输入为single，在这种情况下输出为single类型。

"double"

输出类型为double。

"native".

输出与输入(class (x))具有相同的类型，除非输入是逻辑的，在这种情况下输出是double类型。

可选变量nanflag指定使用任何先前指定的输入参数组合在计算中包括或排除NaN值。nanflag的默认值是" inclenan "，它在计算中保留NaN值。要排除NaN值，请将nanflag的值设置为“omitnan”。如果x由操作维度中的所有NaN值组成，则输出仍将包含NaN值。

参见:mean, mode, movmedian。

: m = mode (x)

: m = mode (x, dim)

: [m, f, c] = mode (…)

计算数据集(模式)中出现频率最高的值。

Mode确定沿第一个非单维的值的频率，并返回频率最高的值。如果两个或多个值具有相同的频率模式，则返回最小的值。

如果给出了可选参数dim，则沿着这个维度操作。

返回变量f是该模式在数据集中出现的次数。

单元格数组c包含所有具有最大频率的元素。

参见:mean, median。

只使用一个数字，比如平均值，来表示整个数据集，可能无法给出数据的准确图像。描述拟合特性的一种方法是测量数据的离散度。八度程提供了几个测量色散的函数。

: [s, l] = bounds (x)

: [s, l] = bounds (x, dim)

: [s, l] = bounds (…, "nanflag")

返回输入数据x的最小值和最大值。

如果x是一个向量，则计算x元素的边界。如果x是一个矩阵，则计算每列的边界。对于多维数组，边界是在第一个非单维上计算的。

如果给出了可选参数dim，则沿着这个维度操作。

可选参数"nanflag"默认为"omitnan"，它不会在结果中包含NaN值。如果给出了参数“inclenan”，并且存在NaN，那么最小(s)和最大(l)元素的结果都是NaN。

边界是对数据集离散度的快速计算度量，但如果存在离群数据点，则不如iqr准确。

参见:range, iqr, mad, std。

: y = range (x)

: y = range (x, dim)

返回范围，即输入数据的最大值和最小值之间的差值。

如果x是一个向量，则在x的元素上计算值域。如果x是一个矩阵，则在x的每列上计算值域。

如果给出了可选参数dim，则沿着这个维度操作。

极差是对数据集离散度的快速计算度量，但如果存在离群数据点，则不如iqr准确。

参见:bounds, iqr, mad, std。

: Z = iqr (x)

: Z = iqr (x, dim)

: Z = iqr (x, "ALL")

返回x的四分位数范围，定义为x的第25和第75个百分位数之间的距离，使用:quantile (x， [0.25 0.75])

如果x是一个向量，iqr (x)将作用于x中的数据。

如果x是一个矩阵，iqr (x)将独立作用于x中的每一列，返回一个行向量Z。

如果x是一个n维数组，iqr (x)将在x的第一个非单维上独立操作，返回与x形状相同的数组Z，但非单维减少为1。

可选变量dim可用于强制iqr在指定的维度上操作。Dim可以是要操作的标量维，也可以是不重复维的向量。在任何一种情况下，dim都必须是正整数。vector dim通过iqr连接所有指定的维度以进行独立操作。

指定维度"ALL"将强制iqr对x的所有元素进行操作，相当于iqr (x(:))。类似地，指定一个包含x的所有非单元素维度的向量维度相当于iqr (x， " all ")。

如果x是标量，或者仅为dim指定了单例维度，则输出将为零(size (x))。

作为离散度的度量，四分位数间距受离群值的影响小于范围或标准值。

参见:bounds, mad, range, std, practice, quantile。

: m = mad (x)

: m = mad (x, opt)

: m = mad (x, opt, dim)

: m = mad (x, opt, vecdim)

: m = mad (x, opt, "all")

计算x元素的平均或中位数绝对偏差。

平均绝对偏差定义为

mad = mean (abs (x - mean (x)))

中位数绝对偏差定义为

mad = median (abs (x - median (x)))

如果x是向量，则对x中的每个元素进行计算。如果x是数组，则在第一个非单元素维上执行计算。

mad从计算中排除NaN值，类似于在var、mean和median中使用omitnan选项。

可选参数opt决定是否计算平均或中位数绝对偏差。默认值为0，对应于平均绝对偏差;值为1对应于绝对偏差中值。传递空输入[]默认表示绝对偏差(opt = 0)。

可选参数dim强制mad沿着指定的维度进行操作。在x中指定任何单例维度，包括任何超过ndims (x)的维度，将导致输出为0。

将维度指定为vecdim(一个不重复维度的向量)，将返回vecdim定义的数组切片的顶点。如果vecdim索引x的所有维度，那么它相当于选项"all"。vecdim中包含的任何大于ndims (x)的维度都将被忽略。

将维度指定为"all"将强制mad对x的所有元素进行操作，并且相当于mad (x(:))。

作为离散度的度量，mad受异常值的影响比std小。

参见:bounds, range, iqr, std, mean, median。

: y = meansq (x)

: y = meansq (x, dim)

计算向量x各元素的均方。

均方定义为

meansq (x) = 1/N SUM\_i x(i)^2

其中N是x向量的长度。

如果x是矩阵，则返回包含每列均方的行向量。

如果给出了可选参数dim，则沿着这个维度操作。

参见:var, std, moment。

: s = std (x)

: s = std (x, w)

: s = std (x, w, dim)

: s = std (x, w, vecdim)

: s = std (x, w, "ALL")

: s = std (…, nanflag)

: [s, m] = std (…)

计算向量x的元素的标准差。

标准偏差定义为

std (x) = sqrt ( 1/(N-1) SUM\_i (x(i) - mean(x))^2 )

其中N是x的元素个数。

如果x是一个数组，计算x的第一个非单维的标准差。

可选参数w决定要使用的权重方案。有效值为:

0(默认):

用N-1(总体标准差)归一化。这提供了标准差的最佳无偏估计量的平方根。

1：

用N(样本标准差)归一化。这提供了第二个矩在均值附近的平方根。

a vector:

计算非负权重的加权标准差。w的长度必须等于操作维中x的大小。在w中允许NaN值，将与x中的相关值相乘，并且可以通过nanflag选项排除NaN值。

an array:

类似于向量权重，但w必须与x的大小相同。如果操作维度以vecdim或"all"形式提供，并且w不是标量，则w必须是相同大小的数组。

注意:必须在指定以下任何维度选项之前指定w。要使用w的默认值，您可以传递一个空输入参数[]。

可选变量dim强制std在指定的维度上操作，该维度必须是一个正整数值。在x中指定任何单个维度，包括任何超过ndims (x)的维度，将导致标准偏差为0。

将维度指定为vecdim(一个不重复维度的向量)，将返回在vecdim定义的数组切片上计算的标准差。如果vecdim索引x的所有维度，那么它相当于选项"all"。vecdim中任何大于ndims (x)的维度都会被忽略。

将维度指定为"all"将强制std对x的所有元素进行操作，这相当于std (x(:))。

可选变量nanflag指定使用任何先前指定的输入参数组合在计算中包括或排除NaN值。nanflag的默认值是" inclenan "，它在计算中保留NaN值。要排除NaN值，请将nanflag的值设置为“omitnan”。如果x由操作维度中的所有NaN值组成，则输出仍将包含NaN值。

可选的第二个输出变量m包含用于计算标准差的x元素的平均值。如果v是加权标准差，那么m也是加权平均值。

参见:var, bounds, mad, range, iqr, mean, median。

除了知道离散度的大小之外，知道数据集的形状也是有用的。例如，数据点是集中在平均值的左边还是右边?Octave提供了几种常用的度量来描述数据集的形状。八度也可以计算矩允许任意形状的措施发展。

: v = var (x)

: v = var (x, w)

: v = var (x, w, dim)

: v = var (x, w, vecdim)

: v = var (x, w, "all")

: v = var (…, nanflag)

: [v, m] = var (…)

计算向量x各元素的方差。

方差定义为

var (x) = (1 / (N-1)) \* SUM\_i ((x(i) - mean(x))^2)

其中N是x的元素个数。

如果x是一个数组，计算x的第一个非单维的方差。

可选参数w决定要使用的权重方案。有效值为:

0(默认)：

用N-1(总体方差)归一化。这提供了方差的最佳无偏估计量的平方根。

1：

用N(样本方差)归一化。这提供了第二个矩在均值附近的平方根。

a vector:

计算非负权重的加权方差。w的长度必须等于操作维中x的大小。在w中允许NaN值，将与x中的相关值相乘，并且可以通过nanflag选项排除NaN值。

an array:

类似于向量权重，但w必须与x的大小相同。如果操作维度以vecdim或"all"形式提供，并且w不是标量，则w必须是相同大小的数组。

注意:必须在指定以下任何维度选项之前指定w。要使用w的默认值，您可以传递一个空输入参数[]。

可选变量dim强制var在指定的维度上操作，该维度必须是正整数值。在x中指定任何单例维度，包括任何超过ndims (x)的维度，将导致方差为0。

将维度指定为vecdim(一个不重复维度的向量)，将返回在vecdim定义的数组切片上计算的方差。如果vecdim索引x的所有维度，那么它相当于选项"all"。vecdim中任何大于ndims (x)的维度都会被忽略。

将维度指定为“all”将强制var对x的所有元素进行操作，并且相当于var (x(:))。

可选变量nanflag指定使用任何先前指定的输入参数组合在计算中包括或排除NaN值。nanflag的默认值是" inclenan "，它在计算中保留NaN值。要排除NaN值，请将nanflag的值设置为“omitnan”。如果x由操作维度中的所有NaN值组成，则输出仍将包含NaN值。

可选的第二个输出变量m包含用于计算方差的x元素的平均值。如果v是加权方差，那么m也是加权均值。

参见:std, mean, cov, skeness, kurtosis, moment。

: y = skewness (x)

: y = skewness (x, flag)

: y = skewness (x, flag, dim)

计算x元素的样本偏度。

样本偏度定义为

mean ((x - mean (x)).^3)

skewness (X) = ------------------------------------.

std (x).^3

可选参数flag控制使用哪种规范化。如果flag等于1(默认值，当flag省略或为空时使用)，则返回上面定义的样本偏度。如果flag等于0，则返回调整后的偏度系数:

sqrt (N\*(N-1)) mean ((x - mean (x)).^3)

skewness (X, 0) = ---------------------- \*--- ---------------------------------.

(N - 2) std (x).^3

其中N是x向量的长度。

调整后的偏度系数是通过将样本的第二和第三中心矩替换为它们的偏差校正版本来获得的。

如果x是一个矩阵，或者更一般地说是一个多维数组，则返回沿第一个非单维的偏度。如果给出了可选的dim参数，则沿着这个维度操作。

参见:var, kurtosis, moment.

: y = kurtosis (x)

: y = kurtosis (x, flag)

: y = kurtosis (x, flag, dim)

计算x元素的样本峰度。

样本峰度定义为

mean ((x - mean (x)).^4)

k1 = -------------------------------------

std (x).^4

可选参数flag控制使用哪种规范化。如果flag等于1(默认值，在省略flag或为空时使用)，则返回上面定义的样本峰度。如果flag等于0，则返回“偏差校正”峰度系数:

N - 1

k0 = 3 + ---------------------- \* ((N + 1) \* k1 - 3 \* (N - 1))

(N - 2)(N - 3)

其中N是x向量的长度。

偏差校正峰度系数是通过用样本的第二和第四中心矩替换它们的无偏版本得到的。它是对正常群体的总体峰度的无偏估计。

如果x是一个矩阵，或者更一般地说是一个多维数组，则返回沿第一个非单维的峰度。如果给出了可选的dim参数，则沿着这个维度操作。

参见:var, skewness, moment。

: m = moment (x, p)

: m = moment (x, p, type)

: m = moment (x, p, dim)

: m = moment (x, p, type, dim)

: m = moment (x, p, dim, type)

计算向量x的第p个中心矩。

x的第p个中心矩定义为:

1/N SUM\_i (x(i) - mean(x))^p

其中N是x向量的长度。

如果x是矩阵，则返回包含每列的第p个中心矩的行向量。

如果给出了可选参数dim，则沿着这个维度操作。

可选的字符串类型指定要计算的力矩的类型。有效的选项有:

"c"

中心矩(默认值)。

"a"

"ac"

绝对中心力矩。关于平均忽略符号的时刻定义为

1/N SUM\_i (abs (x(i) - mean(x)))^p

"r"

生的时刻。关于零的时刻定义为

moment (x) = 1/N SUM\_i x(i)^p

"ar"

绝对的原始时刻。关于零的时刻忽略符号定义为

1/N SUM\_i ( abs (x(i)) )^p

如果同时指定了type和dim，则它们可以以任意顺序出现。

参见: var, skewness, kurtosis。

: q = quantile (x)

: q = quantile (x, p)

: q = quantile (x, p, dim)

: q = quantile (x, p, dim, method)

对于样本x，计算p中累积概率值对应的分位数q。忽略x的所有非数值(nan)。

如果x是一个矩阵，计算每列的分位数，并将它们返回到一个矩阵中，使得q的第i行包含x的每列的p(i)个分位数。

如果p未指定，则返回[0.00 0.25 0.50 0.75 1.00]的分位数。可选参数dim确定计算分位数的维度。如果省略dim，则默认为第一个非单例维度。

可用于计算样本分位数的方法是R (https://www.r-project.org/)使用的九种方法。默认值为method = 5。

不连续样本分位数方法1、2和3

1. 方法1:经验分布函数逆。
2. 方法2:与方法1相似，但在不连续处取平均值。
3. 方法3:SAS定义:最近偶数阶统计量。

连续样本分位数方法4到9，其中p(k)是关于每种方法的代表性cdf的线性插值函数。

1. 方法4:p(k) = k / N，即对经验cdf进行线性插值，其中N为p的长度。
2. 方法5:p(k) = (k - 0.5) / n。也就是说，一个分段线性函数，其中结点是经验cdf步骤中间的值。
3. 方法6:p(k) = k / (N + 1)。
4. 方法7:p(k) = (k - 1) / (N - 1)。
5. 方法8:p(k) = (k - 1/3) / (N + 1/3)。无论x的分布如何，所得到的分位数估计值都近似为中位数无偏。
6. 方法9:p(k) = (k - 3/8) / (N + 1/4)。如果x是正态分布，则所得到的分位数估计对于期望阶统计量是近似无偏的。

Hyndman和Fan(1996)推荐方法8。Maxima、S和R(2.0.0之前的版本)使用7作为默认值。Minitab和SPSS的使用方法MATLAB使用方法5。

引用:

* 贝克尔，r.a.，钱伯斯，j.m.和威尔克斯，a.r.(1988)《新S语言》。Wadsworth & Brooks/Cole。
* Hyndman, R. J.和Fan, Y.(1996)统计软件包中的样本分位数，美国统计学家，50,361-365。
* R:一种统计计算语言与环境https://cran.r-project.org/doc/manuals/fullrefman.pdf。

示例：

x = randi (1000, [10, 1]); # Create empirical data in range 1-1000

q = quantile (x, [0, 1]); # Return minimum, maximum of distribution

q = quantile (x, [0.25 0.5 0.75]); # Return quartiles of distribution

参见: prctile。

: q = prctile (x)

: q = prctile (x, p)

: q = prctile (x, p, dim)

对于样本x，计算与累积概率值p对应的分位数q，以百分比表示。

如果x是一个矩阵，计算每列的百分位数，并将它们返回到一个矩阵中，使得q的第i行包含x的每列的p(i)个百分位数。

如果p未指定，则返回[0 25 50 75 100]的分位数。

可选参数dim确定计算百分位数的维度。如果省略dim，则默认为第一个非单例维度。

编程注意:x的所有非数值(nan)都会被忽略。

参见: quantile。

可以使用统计功能快速生成数据集的摘要视图。

: stats = statistics (x)

: stats = statistics (x, dim)

返回一个向量，包含向量x元素的最小值、第一四分位数、中位数、第三四分位数、最大值、平均值、标准差、偏度和峰度。

如果x是一个矩阵，计算第一个非单维的统计量。

如果给出了可选参数dim，则沿着这个维度操作。

参见: min, max, median, mean, std, skewness, kurtosis。

**26.2数据滑动窗口统计**

在完整数据集的一个分段(即窗口)上计算描述性统计通常是有用的。Octave提供了函数movfun，它将调用任意函数句柄和数据窗口并累积结果。已经提供了许多最常用的函数，例如数据窗口上的移动平均线(movmean)。

: y = movfun (fcn, x, wlen)

: y = movfun (fcn, x, [nb, na])

: y = movfun (…, "property", value)

将函数fcn应用于数据x上长度为wlen的移动窗口。

如果wlen是标量，则函数fcn应用于长度为wlen的移动窗口。当wlen是奇数时，窗口是对称的，并且在中心元素的两侧包括(wlen - 1) / 2个元素。例如，当计算索引5处窗口长度为3的输出时，movfun使用数据元素[4,5,6]。如果wlen是偶数，则窗口是不对称的，并且在中心元素的左边有wlen/2个元素，在中心元素的右边有wlen/2 - 1个元素。例如，当计算索引5处窗口长度为4的输出时，movfun使用数据元素[3,4,5,6]。

如果wlen是一个包含两个元素[nb, na]的数组，则该函数应用于移动窗口-nb:na。该窗口包含nb个当前元素之前的元素和na个当前元素之后的元素。总是包含当前元素。例如，给定wlen =[3,0]，则用于计算索引5的数据为[2,3,4,5]。

在计算过程中，数据输入x被重塑成一个二维wlen × n矩阵，fcn在这个新矩阵上被调用。因此，fcn必须接受数组输入参数，并沿着维度1应用计算，即沿着数组的列。

当应用于有n列的数组(可能是多维数组)时，fcn可能以两种格式返回结果:格式1)大小为1 × n × dim3 ×…× dimn的数组。这是Octave核心函数的典型输出格式。输入demo ("movfun"， 5)作为这个用例的一个例子。格式2)一个长度为n \* numel\_higher\_dims的行向量，其中numel\_higher\_dims是prod (size (x)(3:end))。第i个输入列的fcn输出必须在索引i:n:(n\*numel\_higher\_dims)处的输出中找到。这种格式在将函数连接到数组中或使用nthargout时非常有用。输入demo ("movfun"， 6)作为示例。

可以通过指定属性/值对来控制计算。有效的属性有

"dim"

沿着指定的维度操作，而不是第一个非单例维度的默认值。

"Endpoints"

此属性控制如何在窗口的边界(端点)计算结果。可能的值有:

"shrink" (default)

窗口在数组的开始和结束处被截断，以排除没有源数据的元素。例如，对于长度为3的窗口，y(1) = fcn (x(1:2))， y(end) = fcn (x(end-1:end))。

"discard"

使用超出原始数据数组的窗口的任何y值都将被删除。例如，对于一个10个元素的数据向量和一个长度为3的窗口，输出将只包含8个元素。第一个元素需要计算索引[0,1,2]上的函数，因此被丢弃。最后一个元素需要计算索引[9,10,11]上的函数，因此被丢弃。

"fill"

数据数组之外的任何窗口元素都被NaN替换。例如，对于长度为3的窗口，y(1) = fcn ([NaN, x(1:2)])， y(end) = fcn ([x(end-1:end)， NaN])。此选项通常会导致y在边界处具有NaN值，尽管它受到fcn处理NaN的方式以及属性"nancond"的影响。

user\_value

数据数组之外的任何窗口元素都被指定值user\_value替换，该值必须是一个数字标量。例如，对于长度为3的窗口，y(1) = fcn ([user\_value, x(1:2)])， y(end) = fcn ([x(end-1:end)， user\_value])。user\_value的一个常见选择是0。

"same"

数据数组之外的任何窗口元素都被边界处的x值替换。例如，对于长度为3的窗口，y(1) = fcn ([x(1)， x(1:2)])， y(end) = fcn ([x(end-1:end)， x(end)])。

"periodic"

对窗口进行包装，以便从数据的另一侧获取任何丢失的数据元素。例如，对于长度为3的窗口，y(1) = fcn ([x(end)， x(1:2)])， y(end) = fcn ([x(end-1:end)， x(1)])。

请注意，对于其中一些选项，边界处的窗口大小与中心部分的窗口大小不同，fcn必须在这些情况下工作。

"nancond"

控制从传递给fcn的数据中是否包含NaN和NA值(值:" inclenan ")或排除NaN值(值:"omitnan")。默认值为“include”。注意:“omitnan”选项尚未实现。

"outdim"

选择计算的哪个维度将出现在输出y中的行向量。这仅在fcn以格式1返回n维数组时有用。默认是返回所有输出维度。

编程注意:当fcn的输出有很多维度时，或者当选择所需输出的基本函数的包装器开销太大时，可以使用“outdim”属性来节省内存。当内存不是问题时，选择输出维度的最简单方法是首先使用movfun计算完整的结果，然后使用索引过滤该结果。如果代码复杂性不是问题，那么可以使用匿名函数创建包装器。例如，如果basefcn是一个返回k维行输出的函数，并且只需要D维，那么可以使用以下包装器。

fcn = @(x) basefcn (x)(:,columns(x) \* (D-1) + (1:columns(x)));

y = movfun (@fcn, …);

另见:移动切片，前置，后置，排列，重塑。

: slcidx = movslice (N, wlen)

: [slcidx, C, Cpre, Cpost, win] = movslice (…)

生成索引，将长度为N的向量切片为长度为wlen的窗口。

输入的N必须是正整数。

移动窗口长度输入wlen可以是一个不等于1的标量，也可以是一个包含2个元素的整数数组。对于标量值，如果为奇数，则窗口是对称的，并且在中心元素的两侧包括(wlen - 1) / 2个元素。如果wlen是偶数，则窗口是不对称的，并且在中心元素的左边有wlen/2个元素，在中心元素的右边有wlen/2 - 1个元素。当wlen是一个2元素数组[nb, na]时，窗口在当前元素的左边包含nb个元素，在当前元素的右边包含na个元素。

输出slcidx是一个完全适合向量的切片索引数组，其中每一列都是窗口从左向右移动时的一个单独的切片。对于标量wlen，切片具有wlen元素，对于数组wlen，切片具有nb + na + 1元素。

可选输出C是窗口中心位置的行向量，其中窗口完全保持在向量内。

可选输出Cpre和Cpost分别在vector的开头和结尾包含vector元素，导致窗口扩展到vector的末尾之外。

可选输出win是与slcidx具有相同行数的列向量，其中包含定义为中心相对位置模板的移动窗口。

参见:movfun。

: y = movmad (x, wlen)

: y = movmad (x, [nb, na])

: y = movmad (…, dim)

: y = movmad (…, "nancond")

: y = movmad (…, property, value)

计算在数据x上长度为len的滑动窗口上的移动平均绝对偏差。

如果wlen是标量，则将函数mad应用于长度为wlen的移动窗口。当wlen是奇数时，窗口是对称的，并且在中心元素的两侧包括(wlen - 1) / 2个元素。例如，当计算索引5处窗口长度为3的输出时，movmad使用数据元素[4,5,6]。如果wlen是偶数，则窗口是不对称的，并且在中心元素的左边有wlen/2个元素，在中心元素的右边有wlen/2 - 1个元素。例如，当计算索引5处窗口长度为4的输出时，movmad使用数据元素[3,4,5,6]。

如果wlen是一个包含两个元素[nb, na]的数组，则该函数应用于移动窗口-nb:na。该窗口包含nb个当前元素之前的元素和na个当前元素之后的元素。总是包含当前元素。例如，给定wlen =[3,0]，则用于计算索引5的数据为[2,3,4,5]。

如果给出了可选参数dim，则沿着这个维度操作。

可选的字符串参数"nancond"控制传递给mad的数据是应该包括NaN和NA值(" inclenan ")还是排除NaN和NA值("omitnan")。默认值为“include”。注意:“omitnan”选项尚未实现。

可以通过指定属性/值对来控制计算。有效的属性有

"Endpoints"

此属性控制如何在窗口的边界(端点)计算结果。可能的值有:

"shrink" (default)

窗口在数组的开始和结束处被截断，以排除没有源数据的元素。例如，对于长度为3的窗口，y(1) = mad (x(1:2))， y(end) = mad (x(end-1:end))。

"discard"

使用超出原始数据数组的窗口的任何y值都将被删除。例如，对于一个10个元素的数据向量和一个长度为3的窗口，输出将只包含8个元素。第一个元素需要计算索引[0,1,2]上的函数，因此被丢弃。最后一个元素需要计算索引[9,10,11]上的函数，因此被丢弃。

"fill"

数据数组之外的任何窗口元素都被NaN替换。例如，对于长度为3的窗口，y(1) = mad ([NaN, x(1:2)])， y(end) = mad ([x(end-1:end)， NaN])。此选项通常会导致y在边界处具有NaN值，尽管它受到mad处理NaN的方式以及属性“nancond”的影响。

user\_value

数据数组之外的任何窗口元素都被指定值user\_value替换，该值必须是一个数字标量。例如，对于长度为3的窗口，y(1) = mad ([user\_value, x(1:2)])， y(end) = mad ([x(end-1:end)， user\_value])。user\_value的一个常见选择是0。

"same"

数据数组之外的任何窗口元素都被边界处的x值替换。例如，对于长度为3的窗口，y(1) = mad ([x(1)， x(1:2)])， y(end) = mad ([x(end-1:end)， x(end)])。

"periodic"

对窗口进行包装，以便从数据的另一侧获取任何丢失的数据元素。例如，对于长度为3的窗口，y(1) = mad ([x(end)， x(1:2)])， y(end) = mad ([x(end-1:end)， x(1)])。

"SamplePoints"

注意:此选项尚未实现。

编程注意:这个函数是一个调用movfun的包装器。有关其他选项和文档，请参阅movfun。

参见:movfun, movslice, movmax, movmean, movmedian, movmin, movprod, movstd, movsum, movvar。

: y = movmax (x, wlen)

: y = movmax (x, [nb, na])

: y = movmax (…, dim)

: y = movmax (…, "nancond")

: y = movmax (…, property, value)

计算在数据x上长度为len的滑动窗口上的移动最大值。

如果wlen是标量，则函数max将应用于长度为wlen的移动窗口。当wlen是奇数时，窗口是对称的，并且在中心元素的两侧包括(wlen - 1) / 2个元素。例如，当计算索引5处窗口长度为3的输出时，movmax使用数据元素[4,5,6]。如果wlen是偶数，则窗口是不对称的，并且在中心元素的左边有wlen/2个元素，在中心元素的右边有wlen/2 - 1个元素。例如，当计算索引5处窗口长度为4的输出时，movmax使用数据元素[3,4,5,6]。

如果wlen是一个包含两个元素[nb, na]的数组，则该函数应用于移动窗口-nb:na。该窗口包含nb个当前元素之前的元素和na个当前元素之后的元素。总是包含当前元素。例如，给定wlen =[3,0]，则用于计算索引5的数据为[2,3,4,5]。

如果给出了可选参数dim，则沿着这个维度操作。

可选的字符串参数"nancond"控制NaN和NA值是否应该从传递给max的数据中包含(" inclenan ")或排除("omitnan")。默认值为“include”。注意:“omitnan”选项尚未实现。

可以通过指定属性/值对来控制计算。有效的属性有

"Endpoints"

此属性控制如何在窗口的边界(端点)计算结果。可能的值有:

"shrink" (default)

窗口在数组的开始和结束处被截断，以排除没有源数据的元素。例如，对于长度为3的窗口，y(1) = max (x(1:2))， y(end) = max (x(end-1:end))。

"discard"

使用超出原始数据数组的窗口的任何y值都将被删除。例如，对于一个10个元素的数据向量和一个长度为3的窗口，输出将只包含8个元素。第一个元素需要计算索引[0,1,2]上的函数，因此被丢弃。最后一个元素需要计算索引[9,10,11]上的函数，因此被丢弃。

"fill"

数据数组之外的任何窗口元素都被NaN替换。例如，对于长度为3的窗口，y(1) = max ([NaN, x(1:2)])， y(end) = max ([x(end-1:end)， NaN])。此选项通常会导致y在边界处具有NaN值，尽管它受到max处理NaN的方式以及属性“nancond”的影响。

user\_value

数据数组之外的任何窗口元素都被指定值user\_value替换，该值必须是一个数字标量。例如，对于长度为3的窗口，y(1) = max ([user\_value, x(1:2)])， y(end) = max ([x(end-1:end)， user\_value])。user\_value的一个常见选择是0。

"same"

数据数组之外的任何窗口元素都被边界处的x值替换。例如，对于长度为3的窗口，y(1) = max ([x(1)， x(1:2)])， y(end) = max ([x(end-1:end)， x(end)])。

"periodic"

对窗口进行包装，以便从数据的另一侧获取任何丢失的数据元素。例如，对于长度为3的窗口，y(1) = max ([x(end)， x(1:2)])， y(end) = max ([x(end-1:end)， x(1)])。

"SamplePoints"

注意:此选项尚未实现。

编程注意:这个函数是一个调用movfun的包装器。有关其他选项和文档，请参阅movfun。

参见:movfun, movslice, movmad, movmean, movmedian, movmin, movprod, movstd, movsum, movvar。

: y = movmean (x, wlen)

: y = movmean (x, [nb, na])

: y = movmean (…, dim)

: y = movmean (…, "nancond")

: y = movmean (…, property, value)

在数据x上计算长度为wlen的滑动窗口上的移动平均线。

如果wlen是标量，则将函数均值应用于长度为wlen的移动窗口。当wlen是奇数时，窗口是对称的，并且在中心元素的两侧包括(wlen - 1) / 2个元素。例如，当计算索引5处窗口长度为3的输出时，movmean使用数据元素[4,5,6]。如果wlen是偶数，则窗口是不对称的，并且在中心元素的左边有wlen/2个元素，在中心元素的右边有wlen/2 - 1个元素。例如，当计算索引5处窗口长度为4的输出时，movmean使用数据元素[3,4,5,6]。

如果wlen是一个包含两个元素[nb, na]的数组，则该函数应用于移动窗口-nb:na。该窗口包含nb个当前元素之前的元素和na个当前元素之后的元素。总是包含当前元素。例如，给定wlen =[3,0]，则用于计算索引5的数据为[2,3,4,5]。

如果给出了可选参数dim，则沿着这个维度操作。

可选的字符串参数"nancond"控制NaN和NA值是应该从传递给mean的数据中包含(" inclenan ")还是排除("omitnan")。默认值为“include”。注意:“omitnan”选项尚未实现。

可以通过指定属性/值对来控制计算。有效的属性有

"Endpoints"

此属性控制如何在窗口的边界(端点)计算结果。可能的值有:

"shrink" (default)

窗口在数组的开始和结束处被截断，以排除没有源数据的元素。例如，对于长度为3的窗口，y(1) = mean (x(1:2))， y(end) = mean (x(end-1:end))。

"discard"

使用超出原始数据数组的窗口的任何y值都将被删除。例如，对于一个10个元素的数据向量和一个长度为3的窗口，输出将只包含8个元素。第一个元素需要计算索引[0,1,2]上的函数，因此被丢弃。最后一个元素需要计算索引[9,10,11]上的函数，因此被丢弃。

"fill"

数据数组之外的任何窗口元素都被NaN替换。例如，对于长度为3的窗口，y(1) = mean ([NaN, x(1:2)])， y(end) = mean ([x(end-1:end)， NaN])。此选项通常会导致y在边界处具有NaN值，尽管它受到mean处理NaN的方式以及属性“nancond”的影响。

user\_value

数据数组之外的任何窗口元素都被指定值user\_value替换，该值必须是一个数字标量。例如，对于长度为3的窗口，y(1) = mean ([user\_value, x(1:2)])， y(end) = mean ([x(end-1:end)， user\_value])。user\_value的一个常见选择是0。

"same"

数据数组之外的任何窗口元素都被边界处的x值替换。例如，对于长度为3的窗口，y(1) = mean ([x(1)， x(1:2)])， y(end) = mean ([x(end-1:end)， x(end)])。

"periodic"

对窗口进行包装，以便从数据的另一侧获取任何丢失的数据元素。例如，对于长度为3的窗口，y(1) = mean ([x(end)， x(1:2)])， y(end) = mean ([x(end-1:end)， x(1)])。

"SamplePoints"

注意:此选项尚未实现。

编程注意:这个函数是一个调用movfun的包装器。有关其他选项和文档，请参阅movfun。

参见:movfun, movslice, movmad, movmax, movmedian, movmin, movprod, movstd, movsum, movvar。

: y = movmedian (x, wlen)

: y = movmedian (x, [nb, na])

: y = movmedian (…, dim)

: y = movmedian (…, "nancond")

: y = movmedian (…, property, value)

计算数据x上长度为wlen的滑动窗口上的移动中值。

如果wlen是标量，则函数movmedian应用于长度为wlen的移动窗口。当wlen是奇数时，窗口是对称的，并且在中心元素的两侧包括(wlen - 1) / 2个元素。例如，当计算索引5处窗口长度为3的输出时，movmedian使用数据元素[4,5,6]。如果wlen是偶数，则窗口是不对称的，并且在中心元素的左边有wlen/2个元素，在中心元素的右边有wlen/2 - 1个元素。例如，当计算索引5处窗口长度为4的输出时，movmedian使用数据元素[3,4,5,6]。

如果wlen是一个包含两个元素[nb, na]的数组，则该函数应用于移动窗口-nb:na。该窗口包含nb个当前元素之前的元素和na个当前元素之后的元素。总是包含当前元素。例如，给定wlen =[3,0]，则用于计算索引5的数据为[2,3,4,5]。

如果给出了可选参数dim，则沿着这个维度操作。

可选的字符串参数"nancond"控制传递给movmedian的数据中NaN和NA值是应该包括(" inclenan ")还是应该排除("omitnan")。默认值为“include”。注意:“omitnan”选项尚未实现。

可以通过指定属性/值对来控制计算。有效的属性有

"Endpoints"

此属性控制如何在窗口的边界(端点)计算结果。可能的值有:

"shrink" (default)

窗口在数组的开始和结束处被截断，以排除没有源数据的元素。例如，对于长度为3的窗口，y(1) = movmedian (x(1:2))， y(end) = movmedian (x(end-1:end))。

"discard"

使用超出原始数据数组的窗口的任何y值都将被删除。例如，对于一个10个元素的数据向量和一个长度为3的窗口，输出将只包含8个元素。第一个元素需要计算索引[0,1,2]上的函数，因此被丢弃。最后一个元素需要计算索引[9,10,11]上的函数，因此被丢弃。

"fill"

数据数组之外的任何窗口元素都被NaN替换。例如，对于长度为3的窗口，y(1) = movmedian ([NaN, x(1:2)])， y(end) = movmedian ([x(end-1:end)， NaN])。此选项通常会导致y在边界处具有NaN值，尽管它受到movmedian处理NaN的方式以及属性“nancond”的影响。

user\_value

数据数组之外的任何窗口元素都被指定值user\_value替换，该值必须是一个数字标量。例如，对于长度为3的窗口，y(1) = movmedian ([user\_value, x(1:2)])， y(end) = movmedian ([x(end-1:end)， user\_value])。user\_value的一个常见选择是0。

"same"

数据数组之外的任何窗口元素都被边界处的x值替换。例如，对于长度为3的窗口，y(1) = movmedian ([x(1)， x(1:2)])， y(end) = movmedian ([x(end-1:end)， x(end)])。

"periodic"

对窗口进行包装，以便从数据的另一侧获取任何丢失的数据元素。例如，对于长度为3的窗口，y(1) = movmedian ([x(end)， x(1:2)])， y(end) = movmedian ([x(end-1:end)， x(1)])。

"SamplePoints"

注意:此选项尚未实现。

编程注意:这个函数是一个调用movfun的包装器。有关其他选项和文档，请参阅movfun。

参见:movfun, movslice, movmad, movmax, movmean, movmin, movprod, movstd, movsum, movvar。

: y = movmin (x, wlen)

: y = movmin (x, [nb, na])

: y = movmin (…, dim)

: y = movmin (…, "nancond")

: y = movmin (…, property, value)

计算在数据x上长度为len的滑动窗口上的移动最小值。

如果wlen是标量，则函数min应用于长度为wlen的移动窗口。当wlen是奇数时，窗口是对称的，并且在中心元素的两侧包括(wlen - 1) / 2个元素。例如，当计算索引5处窗口长度为3的输出时，movmin使用数据元素[4,5,6]。如果wlen是偶数，则窗口是不对称的，并且在中心元素的左边有wlen/2个元素，在中心元素的右边有wlen/2 - 1个元素。例如，当计算索引5处窗口长度为4的输出时，movmin使用数据元素[3,4,5,6]。

如果wlen是一个包含两个元素[nb, na]的数组，则该函数应用于移动窗口-nb:na。该窗口包含nb个当前元素之前的元素和na个当前元素之后的元素。总是包含当前元素。例如，给定wlen =[3,0]，则用于计算索引5的数据为[2,3,4,5]。

如果给出了可选参数dim，则沿着这个维度操作。

可选的字符串参数"nancond"控制NaN和NA值是否应该从传递给min的数据中包含(" inclenan ")或排除("omitnan")。默认值是" inclenan "。注意:“omitnan”选项尚未实现。

可以通过指定属性/值对来控制计算。有效的属性有

"Endpoints"

此属性控制如何在窗口的边界(端点)计算结果。可能的值有:

"shrink" (default)

窗口在数组的开始和结束处被截断，以排除没有源数据的元素。例如，对于长度为3的窗口，y(1) = min (x(1:2))， y(end) = min (x(end-1:end))。

"discard"

使用超出原始数据数组的窗口的任何y值都将被删除。例如，对于一个10个元素的数据向量和一个长度为3的窗口，输出将只包含8个元素。第一个元素需要计算索引[0,1,2]上的函数，因此被丢弃。最后一个元素需要计算索引[9,10,11]上的函数，因此被丢弃。

"fill"

数据数组之外的任何窗口元素都被NaN替换。例如，对于长度为3的窗口，y(1) = min ([NaN, x(1:2)])， y(end) = min ([x(end-1:end)， NaN])。此选项通常会导致y在边界处具有NaN值，尽管它受到min处理NaN的方式以及属性“nancond”的影响。

user\_value

数据数组之外的任何窗口元素都被指定值user\_value替换，该值必须是一个数字标量。例如，对于长度为3的窗口，y(1) = min ([user\_value, x(1:2)])， y(end) = min ([x(end-1:end)， user\_value])。user\_value的一个常见选择是0。

"same"

数据数组之外的任何窗口元素都被边界处的x值替换。例如，对于长度为3的窗口，y(1) = min ([x(1)， x(1:2)])， y(end) = min ([x(end-1:end)， x(end)])。

"periodic"

对窗口进行包装，以便从数据的另一侧获取任何丢失的数据元素。例如，对于长度为3的窗口，y(1) = min ([x(end)， x(1:2)])， y(end) = min ([x(end-1:end)， x(1)])。

"SamplePoints"

注意:此选项尚未实现。

编程注意:这个函数是一个调用movfun的包装器。有关其他选项和文档，请参阅movfun。

参见:movfun, movslice, movmad, movmax, movmean, movmedian, movprod, movstd, movsum, movvar。

: y = movprod (x, wlen)

: y = movprod (x, [nb, na])

: y = movprod (…, dim)

: y = movprod (…, "nancond")

: y = movprod (…, property, value)

计算在数据x上长度为wlen的滑动窗口上的移动积。

如果wlen是一个标量，则函数movprod将应用于长度为wlen的移动窗口。当wlen是奇数时，窗口是对称的，并且在中心元素的两侧包括(wlen - 1) / 2个元素。例如，当计算索引5处窗口长度为3的输出时，movprod使用数据元素[4,5,6]。如果wlen是偶数，则窗口是不对称的，并且在中心元素的左边有wlen/2个元素，在中心元素的右边有wlen/2 - 1个元素。例如，当计算索引5处窗口长度为4的输出时，movprod使用数据元素[3,4,5,6]。

如果wlen是一个包含两个元素[nb, na]的数组，则该函数应用于移动窗口-nb:na。该窗口包含nb个当前元素之前的元素和na个当前元素之后的元素。总是包含当前元素。例如，给定wlen =[3,0]，则用于计算索引5的数据为[2,3,4,5]。

如果给出了可选参数dim，则沿着这个维度操作。

可选的字符串参数"nancond"控制传递给movprod的数据中NaN和NA值是应该包括(" inclenan ")还是排除("omitnan")。默认值为“include”。注意:“omitnan”选项尚未实现。

可以通过指定属性/值对来控制计算。有效的属性有

"Endpoints"

此属性控制如何在窗口的边界(端点)计算结果。可能的值有:

"shrink" (default)

窗口在数组的开始和结束处被截断，以排除没有源数据的元素。例如，对于长度为3的窗口，y(1) = movprod (x(1:2))， y(end) = movprod (x(end-1:end))。

"discard"

使用超出原始数据数组的窗口的任何y值都将被删除。例如，对于一个10个元素的数据向量和一个长度为3的窗口，输出将只包含8个元素。第一个元素需要计算索引[0,1,2]上的函数，因此被丢弃。最后一个元素需要计算索引[9,10,11]上的函数，因此被丢弃。

"fill"

数据数组之外的任何窗口元素都被NaN替换。例如，对于长度为3的窗口，y(1) = movprod ([NaN, x(1:2)])， y(end) = movprod ([x(end-1:end)， NaN])。此选项通常会导致y在边界处具有NaN值，尽管它受到movprod处理NaN的方式以及属性“nancond”的影响。

user\_value

数据数组之外的任何窗口元素都被指定值user\_value替换，该值必须是一个数字标量。例如，对于长度为3的窗口，y(1) = movprod ([user\_value, x(1:2)])， y(end) = movprod ([x(end-1:end)， user\_value])。user\_value的一个常见选择是0。

"same"

数据数组之外的任何窗口元素都被边界处的x值替换。例如，对于长度为3的窗口，y(1) = movprod ([x(1)， x(1:2)])， y(end) = movprod ([x(end-1:end)， x(end)])。

"periodic"

对窗口进行包装，以便从数据的另一侧获取任何丢失的数据元素。例如，对于长度为3的窗口，y(1) = movprod ([x(end)， x(1:2)])， y(end) = movprod ([x(end-1:end)， x(1)])。

"SamplePoints"

注意:此选项尚未实现。

编程注意:这个函数是一个调用movfun的包装器。有关其他选项和文档，请参阅movfun。

参见:movfun, movslice, movmad, movmax, movmean, movmedian, movmin, movstd, movsum, movvar。

: y = movstd (x, wlen)

: y = movstd (x, [nb, na])

: y = movstd (…, opt)

: y = movstd (…, opt, dim)

: y = movstd (…, "nancond")

: y = movstd (…, property, value)

在数据x上计算长度为wlen的滑动窗口上的移动标准差。

如果wlen是标量，则函数movstd将应用于长度为wlen的移动窗口。当wlen是奇数时，窗口是对称的，并且在中心元素的两侧包括(wlen - 1) / 2个元素。例如，当计算索引5处窗口长度为3的输出时，movstd使用数据元素[4,5,6]。如果wlen是偶数，则窗口是不对称的，并且在中心元素的左边有wlen/2个元素，在中心元素的右边有wlen/2 - 1个元素。例如，当计算索引5处窗口长度为4的输出时，movstd使用数据元素[3,4,5,6]。

如果wlen是一个包含两个元素[nb, na]的数组，则该函数应用于移动窗口-nb:na。该窗口包含nb个当前元素之前的元素和na个当前元素之后的元素。总是包含当前元素。例如，给定wlen =[3,0]，则用于计算索引5的数据为[2,3,4,5]。

可选参数opt决定要使用的规范化类型。有效值是

0:

用N-1归一化，提供方差的最佳无偏估计量的平方根[默认值]

1:

用N归一化，这提供了在平均值周围的第二个矩的平方根

如果给出了可选参数dim，则沿着这个维度操作。规范化参数opt必须在维度之前给出。

可选的字符串参数"nancond"控制传递给movstd的数据是应该包括NaN和NA值(" inclenan ")还是排除NaN和NA值("omitnan")。默认值为“include”。注意:“omitnan”选项尚未实现。

可以通过指定属性/值对来控制计算。有效的属性有

"Endpoints"

此属性控制如何在窗口的边界(端点)计算结果。可能的值有:

"shrink" (default)

窗口在数组的开始和结束处被截断，以排除没有源数据的元素。例如，对于长度为3的窗口，y(1) = movstd (x(1:2))， y(end) = movstd (x(end-1:end))。

"discard"

使用超出原始数据数组的窗口的任何y值都将被删除。例如，对于一个10个元素的数据向量和一个长度为3的窗口，输出将只包含8个元素。第一个元素需要计算索引[0,1,2]上的函数，因此被丢弃。最后一个元素需要计算索引[9,10,11]上的函数，因此被丢弃。

"fill"

数据数组之外的任何窗口元素都被NaN替换。例如，对于长度为3的窗口，y(1) = movstd ([NaN, x(1:2)])， y(end) = movstd ([x(end-1:end)， NaN])。此选项通常会导致y在边界处具有NaN值，尽管它受到movstd处理NaN的方式以及属性“nancond”的影响。

user\_value

数据数组之外的任何窗口元素都被指定值user\_value替换，该值必须是一个数字标量。例如，对于长度为3的窗口，y(1) = movstd ([user\_value, x(1:2)])， y(end) = movstd ([x(end-1:end)， user\_value])。user\_value的一个常见选择是0。

"same"

数据数组之外的任何窗口元素都被边界处的x值替换。例如，对于长度为3的窗口，y(1) = movstd ([x(1)， x(1:2)])， y(end) = movstd ([x(end-1:end)， x(end)])。

"periodic"

对窗口进行包装，以便从数据的另一侧获取任何丢失的数据元素。例如，对于长度为3的窗口，y(1) = movstd ([x(end)， x(1:2)])， y(end) = movstd ([x(end-1:end)， x(1)])。

"SamplePoints"

注意:此选项尚未实现。

编程注意:这个函数是一个调用movfun的包装器。有关其他选项和文档，请参阅movfun。

参见:movfun, movslice, movmad, movmax, movmean, movmedian, movmin, movprod, movsum, movvar。

: y = movsum (x, wlen)

: y = movsum (x, [nb, na])

: y = movsum (…, dim)

: y = movsum (…, "nancond")

: y = movsum (…, property, value) ¶

在数据x上计算长度为wlen的滑动窗口上的移动和。

如果wlen是标量，则将函数movsum应用于长度为wlen的移动窗口。当wlen是奇数时，窗口是对称的，并且在中心元素的两侧包括(wlen - 1) / 2个元素。例如，当计算索引5处窗口长度为3的输出时，movsum使用数据元素[4,5,6]。如果wlen是偶数，则窗口是不对称的，并且在中心元素的左边有wlen/2个元素，在中心元素的右边有wlen/2 - 1个元素。例如，当计算索引5处窗口长度为4的输出时，movsum使用数据元素[3,4,5,6]。

如果wlen是一个包含两个元素[nb, na]的数组，则该函数应用于移动窗口-nb:na。该窗口包含nb个当前元素之前的元素和na个当前元素之后的元素。总是包含当前元素。例如，给定wlen =[3,0]，则用于计算索引5的数据为[2,3,4,5]。

如果给出了可选参数dim，则沿着这个维度操作。

可选的字符串参数"nancond"控制传递给movsum的数据中NaN和NA值是应该包括(" inclenan ")还是排除("omitnan")。默认值为“include”。注意:“omitnan”选项尚未实现。

可以通过指定属性/值对来控制计算。有效的属性有

"Endpoints"

此属性控制如何在窗口的边界(端点)计算结果。可能的值有:

"shrink" (default)

窗口在数组的开始和结束处被截断，以排除没有源数据的元素。例如，对于长度为3的窗口，y(1) = movsum (x(1:2))， y(end) = movsum (x(end-1:end))。

"discard"

使用超出原始数据数组的窗口的任何y值都将被删除。例如，对于一个10个元素的数据向量和一个长度为3的窗口，输出将只包含8个元素。第一个元素需要计算索引[0,1,2]上的函数，因此被丢弃。最后一个元素需要计算索引[9,10,11]上的函数，因此被丢弃。

"fill"

数据数组之外的任何窗口元素都被NaN替换。例如，对于长度为3的窗口，y(1) = movsum ([NaN, x(1:2)])， y(end) = movsum ([x(end-1:end)， NaN])。此选项通常会导致y在边界处具有NaN值，尽管它受到movsum处理NaN的方式以及属性“nancond”的影响。

user\_value

数据数组之外的任何窗口元素都被指定值user\_value替换，该值必须是一个数字标量。例如，对于长度为3的窗口，y(1) = movsum ([user\_value, x(1:2)])， y(end) = movsum ([x(end-1:end)， user\_value])。user\_value的一个常见选择是0。

"same"

数据数组之外的任何窗口元素都被边界处的x值替换。例如，对于长度为3的窗口，y(1) = movsum ([x(1)， x(1:2)])， y(end) = movsum ([x(end-1:end)， x(end)])。

"periodic"

对窗口进行包装，以便从数据的另一侧获取任何丢失的数据元素。例如，对于长度为3的窗口，y(1) = movsum ([x(end)， x(1:2)])， y(end) = movsum ([x(end-1:end)， x(1)])。

"SamplePoints"

注意:此选项尚未实现。

编程注意:这个函数是一个调用movfun的包装器。有关其他选项和文档，请参阅movfun。

参见:movfun, movslice, movmad, movmax, movmean, movmedian, movmin, movprod, movstd, movvar。

: y = movvar (x, wlen)

: y = movvar (x, [nb, na])

: y = movvar (…, opt)

: y = movvar (…, opt, dim)

: y = movvar (…, "nancond")

: y = movvar (…, property, value)

计算数据x上长度为wlen的滑动窗口上的移动方差。

如果wlen是标量，则函数var将应用于长度为wlen的移动窗口。当wlen是奇数时，窗口是对称的，并且在中心元素的两侧包括(wlen - 1) / 2个元素。例如，当计算索引5处窗口长度为3的输出时，movvar使用数据元素[4,5,6]。如果wlen是偶数，则窗口是不对称的，并且在中心元素的左边有wlen/2个元素，在中心元素的右边有wlen/2 - 1个元素。例如，当计算索引5处窗口长度为4的输出时，movvar使用数据元素[3,4,5,6]。

如果wlen是一个包含两个元素[nb, na]的数组，则该函数应用于移动窗口-nb:na。该窗口包含nb个当前元素之前的元素和na个当前元素之后的元素。总是包含当前元素。例如，给定wlen =[3,0]，则用于计算索引5的数据为[2,3,4,5]。

可选参数opt决定要使用的规范化类型。有效值是

0:

用N-1进行归一化，提供方差的最佳无偏估计[默认值]

1:

用N归一化，这提供了平均值附近的第二个矩

如果给出了可选参数dim，则沿着这个维度操作。规范化参数opt必须在维度之前给出。

可选的字符串参数"nancond"控制NaN和NA值是否应该从传递给var的数据中包含(" inclenan ")或排除("omitnan")。默认值是" inclenan "。注意:“omitnan”选项尚未实现。

可以通过指定属性/值对来控制计算。有效的属性有

"Endpoints"

此属性控制如何在窗口的边界(端点)计算结果。可能的值有:

"shrink" (default)

窗口在数组的开始和结束处被截断，以排除没有源数据的元素。例如，对于长度为3的窗口，y(1) = var (x(1:2))， y(end) = var (x(end-1:end))。

"discard"

使用超出原始数据数组的窗口的任何y值都将被删除。例如，对于一个10个元素的数据向量和一个长度为3的窗口，输出将只包含8个元素。第一个元素需要计算索引[0,1,2]上的函数，因此被丢弃。最后一个元素需要计算索引[9,10,11]上的函数，因此被丢弃。

"fill"

数据数组之外的任何窗口元素都被NaN替换。例如，对于长度为3的窗口，y(1) = var ([NaN, x(1:2)])， y(end) = var ([x(end-1:end)， NaN])。此选项通常会导致y在边界处具有NaN值，尽管它受到var处理NaN的方式以及属性“nancond”的影响。

user\_value

数据数组之外的任何窗口元素都被指定值user\_value替换，该值必须是一个数字标量。例如，对于长度为3的窗口，y(1) = var ([user\_value, x(1:2)])， y(end) = var ([x(end-1:end)， user\_value])。user\_value的一个常见选择是0。

"same"

数据数组之外的任何窗口元素都被边界处的x值替换。例如，对于长度为3的窗口，y(1) = var ([x(1)， x(1:2)])， y(end) = var ([x(end-1:end)， x(end)])。

"periodic"

对窗口进行包装，以便从数据的另一侧获取任何丢失的数据元素。例如，对于长度为3的窗口，y(1) = var ([x(end)， x(1:2)])， y(end) = var ([x(end-1:end)， x(1)])。

"SamplePoints"

注意:此选项尚未实现。

编程注意:这个函数是一个调用movfun的包装器。有关其他选项和文档，请参阅movfun。

参见:movfun, movslice, movmad, movmax, movmean, movmedian, movmin, movprod, movstd, movsum。

**26.3基本统计功能**

Octave支持各种有用的统计功能。许多方法都是为进一步分析准备数据集的初始步骤。其他方法则提供了不同于基本描述性统计的测量方法。

: y = center (x)

: y = center (x, dim)

通过减去其平均值来居中数据。

如果x是向量，减去它的均值。

如果x是一个矩阵，对每一列都做上述操作。

如果给出了可选参数dim，则沿着这个维度操作。

编程说明:center在统计数据规范化方面有明显的应用。它对提高一般数值计算的精度也很有用。当一批数据中有一个较大的共同值时，可以减去平均值，进行计算，然后将平均值加回去得到最终答案。

参见:zscore。

: z = zscore (x)

: z = zscore (x, opt)

: z = zscore (x, opt, dim)

: [z, mu, sigma] = zscore (…)

计算x的Z值。

如果x是一个向量，减去它的均值，再除以它的标准差。如果标准差为0，则除以1。

可选参数opt决定了在计算标准偏差时使用的归一化，并且与std的相应参数具有相同的定义。

如果x是一个矩阵，沿着第一个非单维计算。如果给出了第三个可选参数dim，则沿着该维度进行操作。

可选输出mu和sigma包含均值和标准差。

参见:mean, std, center。

: z = normalize (x)

: z = normalize (x, dim)

: z = normalize (…, method)

: z = normalize (…, method, option)

: z = normalize (…, scale, scaleoption, center, centeroption)

: [z, c, s] = normalize (…)

使用几种可用的缩放和定心方法之一返回x中数据的规范化。

normalize默认情况下将返回x的zscore，定义为每个元素与x的平均值的标准差数。这相当于以数据的平均值为中心并按标准差缩放。

返回值z将具有与x相同的大小。可选的返回变量c和s是归一化中使用的定心和缩放因子，以便:

z = (x - c) ./ s

如果x是一个向量，则normalize将对x中的数据进行操作。

如果x是一个矩阵，则normalize将独立地作用于x中的每一列。

如果x是一个n维数组，normalize将在x的第一个非单维上独立操作。

如果给出了第二个可选参数dim，则沿着这个维度进行操作。

normalize忽略NaN值是x，类似于std、mean和median中的omitnan选项的行为。

可选输入方法和选项可用于指定对x执行的规范化类型。请注意，只有缩放和居中选项可以使用下面定义的任何方法一起指定。有效的归一化方法有:

zscore

(默认)将x中的元素标准化为与中心值的缩放距离。有效的选项:

std

(默认)数据以平均值(x)为中心，并按标准偏差缩放。

robust

数据以中位数(x)为中心，并按中位数绝对偏差进行缩放。

norm

Z为x的一般向量范数，option为确定向量范数类型的归一化因子p，根据:

z = [sum (abs (x) .^ p)] ^ (1/p)

P可以是任意正标量，具体值为:

p = 1

X通过sum (abs (X))归一化。

p = 2

(默认)x由元素的欧几里得范数或矢量大小归一化。

P = Inf

X被Max (abs (X))规范化。

scale

X由选项确定的因子进行缩放，该因子可以是数字标量或以下之一:

std

(默认)x按其标准差缩放。

mad

X按绝对偏差中值缩放。

first

X乘以它的第一个元素。

iqr

X按其四分位数范围进行缩放。

range

X被缩放以适合由选项指定的两个元素标量行向量的范围。默认取值范围为[0,1]。

center

X的偏移量由选项决定，可以是数字标量或以下选项之一:

mean

(默认)x被mean (x)平移。

median

x is shifted by median (x).

medianiqr

X按中位数(X)移位，并按四分位数范围缩放。

已知的MATLAB不兼容性:

1. 对于表类x的输入，还没有实现datvariables选项。

参见:zscore, iqr, norm, rescale, std, median, mean, mad。

: n = histc (x, edges)

: n = histc (x, edges, dim)

: [n, idx] = histc (…)

计算直方图计数。

当x是一个向量时，该函数计算x中落在由边定义的直方图箱中的元素的数量。这必须是一个单调递增值的向量，它定义了直方图箱的边缘。N (k)包含x中边(k) <= x <边(k+1)的元素个数。n的最后一个元素包含的x的元素数正好等于边的最后一个元素。

当x是n维数组时，计算沿着维度dim进行。如果未指定dim，则默认为第一个非单维。

当请求第二个输出参数时，还会返回索引矩阵。idx矩阵的大小与x相同，idx的每个元素都包含x对应元素所在的直方图bin的索引。

参见:历史。

Unique记录的惟一函数通常对统计有用。

: c = nchoosek (n, k)

: c = nchoosek (set, k)

计算n的二项式系数或列出一组项目的所有可能组合。

如果n是标量，则计算n和k的二项式系数，定义为

/ \

| n | n (n-1) (n-2) … (n-k+1) n!

| | = ------------------------- = ---------

| k | k! k! (n-k)!

\ /

这是大小为k的组中n个项目的组合数。

如果第一个参数是向量set，则生成set中元素的所有组合，每次取k个，每个组合有一行。结果c有k列和nchoosek (length (set)， k)行。

例如:

三个项目有多少种成对的组合方式?

nchoosek (3, 2)

⇒ 3

可能的配对是什么?

nchoosek (1:3, 2)

⇒ 1 2

1 3

2 3

编程注意:在计算二项式系数时，nchoosek只适用于非负的整数参数。将bincoeff用于非整数和负标量参数，或者使用n或k的矢量输入同时计算多个二项式系数。

参见:bincoeff, perms。

: P = perms (v)

: P = perms (v, "unique")

生成向量v的所有排列，每个排列有一行。

如果v是升序的，则以反字典顺序返回结果。如果v在不同的排列中，那么结果也是这样排列的。因此，按降序输入将产生按正常字典顺序排列的结果。结果为size阶乘(n) \* n，其中n是v的长度。任何重复的元素都包含在输出中。

如果给出了可选参数"unique"，则只返回唯一的排列，使用更少的内存和花费比调用unique (perms (v)， "rows")更少的时间。

示例1

perms ([1, 2, 3])

⇒

3 2 1

3 1 2

2 3 1

2 1 3

1 3 2

1 2 3

例二

perms ([1, 1, 2, 2], "unique")

⇒

2 2 1 1

2 1 2 1

2 1 1 2

1 2 2 1

1 2 1 2

1 1 2 2

编程注意:如果不使用“unique”选项，则v的长度不应超过10-12，以限制内存消耗。即使使用“唯一”，v中也不应超过10-12个唯一元素。

参见:permute, randperm, nchoosek。

: y = ranks (x)

: y = ranks (x, dim)

: y = ranks (x, dim, rtype)

返回x在第一个非单维上的排名(在顺序统计的意义上)。

如果给出了可选的dim参数，则沿着这个维度操作。

可选参数rtype决定了如何处理关系。下面所有的例子都假设输入为[1,2,2,4]。

0或“分数”(默认)用于分数排名(1,2.5，2.5, 4);

1或“competition”表示比赛排名(1,2,2,4);

2或“修改”为修改后的比赛排名(1,3,3,4);

3或“序数”表示序数排序(1,2,3,4);

4或“dense”表示密集排序(1,2,2,3)。

参见:长矛兵，肯德尔。

: cnt = run\_count (x, n)

: cnt = run\_count (x, n, dim)

计算x的长度为1,2，…，n-1且大于或等于n的第一个非单维向上运行的次数。

如果给出了可选参数dim，则沿着该维度进行操作。

参见:运行长度。

: count = runlength (x)

: [count, value] = runlength (x)

求所有公值序列的长度。

Count是一个包含每个重复值长度的向量。

可选输出值包含序列中重复的值。

runlength ([2, 2, 0, 4, 4, 4, 0, 1, 1, 1, 1])

⇒ 2 1 3 1 4

参见:run\_count。

**26.4相关与回归分析**

: c = cov (x)

: c = cov (x, y)

: c = cov (…, opt)

: c = cov (…, nanflag)

计算协方差矩阵。

两个变量向量A和B之间的协方差计算为:

cov (a,b) = 1/(N-1) \* SUM\_i (a(i) - mean (a)) \* (b(i) - mean (b))

其中N是向量a和b的长度。

如果只带一个参数调用，则计算cov (x, x)。如果x是一个向量，则这是x的标量方差。如果x是一个矩阵，则将x的每一行视为一个观测值，每一列作为一个变量，cov (x)的第(i, j)项是x中第i列和第j列之间的协方差。如果x的维数为n x m，则输出c将是一个m x m方协方差矩阵。

如果用两个参数调用，compute cov (x, y)，两个随机变量x和y之间的协方差x和y必须具有相同数量的元素，并且将被视为计算为cov (x(:)， y(:)的协方差的向量。输出将是一个2 × 2的协方差矩阵。

可选参数opt决定要使用的规范化类型。有效值是

0(默认):

N-1归一化。这提供了协方差的最佳无偏估计。

1:

用n归一化，这提供了平均值附近的第二个矩。当N = 1时，opt设置为1。

可选参数nanflag必须出现在参数列表的最后，并控制cov如何处理NaN值。三个有效值是:

includenan(默认):

将NaN值保留在x和y中。输出将遵循在算术运算中处理NaN值的正常规则。

omitrows:

包含NaN值的行在计算协方差之前从x和y中删除。一个变量中的NaN将从x和y中删除该行。

partialrows:

对于每一次第i次和第j次协方差计算，包含NaN值的行在x和y中都被独立地忽略。这可能导致使用不同数量的观测值N来计算协方差矩阵的每个元素。

兼容性注意:以前版本的cov将行x和y视为多变量随机变量。该版本试图通过将x和y视为两个单变量分布而不考虑形状来保持与MATLAB的完全兼容性，从而产生2x2输出矩阵。运行新版本的cov时，需要修改依赖Octave先前定义的代码。前面的行为可以通过使用NaN包的covm函数covm (x, y，“D”)来获得。

参见:corr。

: r = corr (x)

: r = corr (x, y)

计算相关系数矩阵。

如果x和y的每一行都是一个观测值，每一列都是一个变量，那么corr (x, y)的第(i, j)项是x中第i个变量与y中第j个变量之间的相关性。x和y必须具有相同的行数(观测值)。

corr (x,y) = cov (x,y) / (std (x) \* std (y))

如果只带一个参数调用，则计算corr (x, x)，即x列之间的相关性。

参见:cov。

: r = corrcoef (x)

: r = corrcoef (x, y)

: r = corrcoef (…, param, value, …)

: [r, p] = corrcoef (…)

: [r, p, lci, hci] = corrcoef (…)

计算相关系数矩阵。

X是一个数组，其中每一列包含一个变量，每一行是一个观测值。

如果给定第二个输入y(与x大小相同)，则计算x和y之间的相关系数。

参数、值是可选的参数和值对，它们可以修改计算结果。有效的选项有:

"alpha"

用于置信区间lci和hci边界的置信水平。默认值为0.05，即95%置信区间。

"rows"

确定NaN值的处理。可接受的值是“all”、“complete”和“pairwise”。默认为“all”。使用“complete”时，只考虑没有NaN值的行。使用“成对”，为每对变量选择无nan行。

输出r是每一对变量的Pearson积矩相关系数的矩阵。

输出p是对相关系数为零的零假设进行成对p值检验的矩阵。

输出lci和hci是矩阵，分别包含每个相关系数的95%置信区间的下界和上界。

参见: corr, cov, std。

: rho = spearman (x)

: rho = spearman (x, y)

计算斯皮尔曼等级相关系数。

对于两个数据向量x和y, Spearman 's rho是x和y的秩的相关系数。

如果x和y是从独立分布中得出的，则平均值为零，方差为1 / (N - 1)，其中N是x和y向量的长度，并且是渐近正态分布。

Spearman (x)等价于Spearman (x, x)。

参见: ranks, kendall。

: tau = kendall (x)

: tau = kendall (x, y)

计算Kendall的tau。

对于公共长度为N的两个数据向量x, y, Kendall 's tau是x和y的所有秩差符号的相关性;也就是说，如果x和y都有不同的元素，那么

1

tau = --------------- SUM sign (q(i) - q(j)) \* sign (r(i) - r(j))

N (N-1) i,j

其中q(i)和r(i)分别是x和y的秩。

如果x和y是从独立分布中提取的，Kendall的tau是渐近正态，均值为0，方差为(2 \* (2N+5)) / (9 \* N \* (N-1))。

Kendall (x)等价于Kendall (x, x)。

参见: ranks, spearman。

**26.5发行版**

Octave具有计算任意用户定义分布(离散)和实验数据(经验)的概率密度函数(PDF)、累积分布函数(CDF)和分位数(CDF的倒数)的函数。

下表总结了支持的发行版(按字母顺序排列)。

| **Distribution** | **PDF** | **CDF** | **Quantile** |
| --- | --- | --- | --- |
| Univariate Discrete Distribution | *discrete\_pdf* | *discrete\_cdf* | *discrete\_inv* |
| Empirical Distribution | *empirical\_pdf* | *empirical\_cdf* | *empirical\_inv* |

: pdf = discrete\_pdf (x, v, p)

对于x的每个元素，计算单变量离散分布在x处的概率密度函数(PDF)，该分布假设v中的值具有概率p。

: cdf = discrete\_cdf (x, v, p)

对于x的每个元素，计算单变量离散分布在x处的累积分布函数(CDF)，该分布假设v中的值具有概率p。

: q = discrete\_inv (x, v, p)

对于x的每个元素，计算单变量分布在x处的分位数(CDF的倒数)，该分布假设v中的值具有概率p。

: pdf = empirical\_pdf (x, data)

对于x的每个元素，计算从单变量样本数据获得的经验分布在x处的概率密度函数(PDF)。

: cdf = empirical\_cdf (x, data)

对于x的每个元素，计算从单变量样本数据获得的经验分布在x处的累积分布函数(CDF)。

: q = empirical\_inv (x, data)

对于x的每个元素，计算从单变量样本数据获得的经验分布的x处的分位数(CDF的倒数)。

**26.6随机数生成**

Octave可以从大量分布中生成随机数。随机数生成器基于特殊效用矩阵中描述的随机数生成器。

下表总结了可用的随机数生成器(按字母顺序排列)。

| **Distribution** | **Function** |
| --- | --- |
| Univariate Discrete Distribution | *discrete\_rnd* |
| Empirical Distribution | *empirical\_rnd* |
| Exponential Distribution | *rande* |
| Gamma Distribution | *randg* |
| Poisson Distribution | *randp* |
| Standard Normal Distribution | *randn* |
| Uniform Distribution | *rand* |
| Uniform Distribution (integers) | *randi* |

: rnd = discrete\_rnd (v, p)

: rnd = discrete\_rnd (v, p, r)

: rnd = discrete\_rnd (v, p, r, c, …)

: rnd = discrete\_rnd (v, p, [sz])

返回来自单变量分布的随机样本矩阵，该矩阵假设v中的值概率为p。

当使用单个size参数调用时，返回具有指定尺寸的方阵。当使用多个标量参数调用时，前两个参数作为行数和列数，任何其他参数指定额外的矩阵维度。尺寸也可以用维度为sz的向量指定。

如果没有给出大小参数，则结果矩阵是v和p的公共大小。

: rnd = empirical\_rnd (data)

: rnd = empirical\_rnd (data, r)

: rnd = empirical\_rnd (data, r, c, …)

: rnd = empirical\_rnd (data, [sz])

从单变量样本数据获得的经验分布中返回随机样本矩阵。

当使用单个size参数调用时，返回具有指定尺寸的方阵。当使用多个标量参数调用时，前两个参数作为行数和列数，任何其他参数指定额外的矩阵维度。尺寸也可以用维度为sz的向量指定。

如果没有给出大小参数，则结果矩阵是样本数据的随机排序。

**27集**

Octave有许多用于管理数据集的函数。集合被定义为唯一元素的集合，通常由按升序排序的数字向量表示。任何向量或矩阵都可以通过使用惟一函数除去重复项而转换为集合。然而，没有必要显式地创建set，因为所有操作set的函数在继续之前都会将它们的输入转换为set。

: y = unique (x)

: y = unique (x, "rows")

: y = unique (…, "sorted")

: y = unique (…, "stable")

: [y, i, j] = unique (…)

: [y, i, j] = unique (…, "first")

: [y, i, j] = unique (…, "last")

: [y, i, j] = unique (…, "legacy")

返回x的唯一元素。

如果输入x是列向量，则返回一个列向量;否则，返回一个行向量。X也可以是字符串的单元格数组。

如果给出了可选参数"rows"，则返回x的唯一行。要使用此选项，输入必须是二维数字矩阵。

可选参数“sorted”/“stable”控制唯一值在输出中出现的顺序。默认值为“sorted”，输出中的值按升序排列。另一种“stable”保留了输入x中的顺序。

如果请求，返回列索引向量i和j，使得y = x(i)和x = y(j)。

此外，如果i是请求的输出，则可以给出标志“first”或“last”之一。如果指定了"last"，则返回i中可能的最高索引，否则，如果指定了"first"，则返回最低索引。默认值是“first”。

例1:排序顺序

unique ([3, 1, 1, 2])

⇒ [1, 2, 3]

unique ([3, 1, 1, 2], "stable")

⇒ [3, 1, 2]

例2:索引选择

[~, i] = unique ([3, 1, 1, 2], "first")

⇒ i = [2; 4; 1]

[~, i] = unique ([3, 1, 1, 2], "last")

⇒ i = [3; 4; 1]

编程注意事项:输入标志“legacy”将算法更改为与R2012b之前的MATLAB版本兼容。具体来说，将索引排序标志更改为“last”，输出i, j的形状将遵循输入x的形状，而不是始终是列向量。

当排序顺序“稳定”时，第三个输出j尚未实现。

参见: uniquetol,union, intersect, setdiff, setxor, ismember。

: c = uniquetol (A)

: c = uniquetol (A, tol)

: c = uniquetol (…, property, value)

: [c, ia, ic] = uniquetol (…)

返回公差工具中A的唯一元素。

如果abs (x - y) <= tol \* max (abs (A(:))))，则两个值x和y在相对公差范围内。

输入A必须是实数(非复数)浮点类型(双精度或单精度)。

如果未指定tol，则双精度输入的默认公差为1e-12，单精度输入的默认公差为1e-6。

该函数也可以使用以下可选的属性/值对调用。属性/值对必须在其他输入参数之后传递:

"ByRows" (default: false)

当为true时，返回A的唯一行。A必须是二维数组才能使用此选项。对于行，唯一性标准更改为all (abs (x - y) <= tol\*max (abs (A)，[]，1))，它将行中的每个列组件与列特定的公差进行比较。

"DataScale"

公差测试更改为abs (x - y) <= tol\*DS，其中DS是一个标量，除非属性“ByRows”为真。在这种情况下，DS可以是一个标量，也可以是一个长度等于a中的列数的向量。使用DS的值1.0将把相对容差更改为绝对容差。使用Inf值将禁用测试。

"OutputAllIndices" (default: false)

当为true时，ia是一个单元格数组(而不是一个向量)，它包含a中所有元素的索引，这些元素的索引在C的容差范围内。也就是说，ia中的每个单元格对应于C中的一个唯一值，每个单元格中的值对应于a中的位置。

如果输入a是行向量，输出c就是行向量。对于所有其他情况，返回一个列向量。

可选输出ia是一个列索引向量，使得c = a (ia)。如果"ByRows"属性为真，则条件为c = A(ia，:)。如果“OutputAllIndices”属性为真，则值A(ia{i})都在唯一值c(i)的容差范围内。

可选输出ic是一个列索引向量，当a是一个向量时，a = c(ic)。当A是矩阵时，A(:) = c(ic)。如果"ByRows"属性为真，则A = c(ic，:)。

例如:小的舍入错误需要uniquetol，而不是unique

x = [1:5];

## Inverse\_Function (Function (x)) should return exactly x

y = exp (log (x));

D = unique ([x, y])

⇒ [1 2 3 3 4 5 5]

C = uniquetol ([x, y])

⇒ [1 2 3 4 5]

参见:unique, union, intersect, setdiff, setxor, ismember。

**27.1设置操作**

Octave支持几种基本的集合操作。Octave可以计算两个集合的并集、交集和差集。Octave还支持Exclusive Or设置操作。

集合操作的函数都以同样的方式工作，接受两个输入集合并返回第三个输入集合。例如，假设a和b包含两个集合，则

union (a, b)

计算两个集合的并集。

最后，可以使用ismember函数确定元素是否属于集合。因为集合是有序的，所以这个操作非常有效并且是O(log2(n))阶的，这比find函数的O(n)阶更可取。

: c = intersect (a, b)

: c = intersect (a, b, "rows")

: c = intersect (…, "sorted")

: c = intersect (…, "stable")

: c = intersect (…, "legacy")

: [c, ia, ib] = intersect (…)

返回a和b所共有的唯一元素。

如果a和b都是行向量，则返回一个行向量;否则，返回列向量。输入也可以是字符串的单元格数组。

如果给出了可选的输入“rows”，则返回a和b的公共行。输入必须是二维数字矩阵才能使用此选项。

可选参数“sorted”/“stable”控制唯一值在输出中出现的顺序。默认值为“sorted”，输出中的值按升序排列。另一种“稳定”保留输入中的顺序。

如果请求，返回列索引向量ia和ib，使得c = a(ia)和c = b(ib)。

编程注意:输入标志“legacy”将算法更改为与R2012b之前的MATLAB版本兼容。

参见:unique, union, setdiff, setxor, ismember。

: c = union (a, b)

: c = union (a, b, "rows")

: c = union (…, "sorted")

: c = union (…, "stable")

: c = union (…, "legacy")

: [c, ia, ib] = union (…)

返回在a或b中的唯一元素。

如果a和b都是行向量，则返回一个行向量;否则，返回列向量。输入也可以是字符串的单元格数组。

如果给出了可选的输入“rows”，则返回a或b中的行。输入必须是二维数字矩阵才能使用此选项。

可选参数“sorted”/“stable”控制唯一值在输出中出现的顺序。默认值为“sorted”，输出中的值按升序排列。另一种“稳定”保留输入中的顺序。

可选输出ia和ib是列索引向量，使得a(ia)和b(ib)是不相交的集合，其并集为c。

编程注意:输入标志“legacy”将算法更改为与R2012b之前的MATLAB版本兼容。

参见:unique, intersect, setdiff, setxor, ismember。

: c = setdiff (a, b)

: c = setdiff (a, b, "rows")

: c = setdiff (…, "sorted")

: c = setdiff (…, "stable")

: c = setdiff (…, "legacy")

: [c, ia] = setdiff (…)

返回a中不属于b的唯一元素。

如果a是一个行向量，返回一个行向量;否则，返回列向量。输入也可以是字符串的单元格数组。

如果给出了可选的输入“rows”，则返回不在b中的a中的行。输入必须是二维数字矩阵才能使用此选项。

可选参数“sorted”/“stable”控制唯一值在输出中出现的顺序。默认值为“sorted”，输出中的值按升序排列。另一种“稳定”保留输入中的顺序。

如果请求，返回索引向量ia，使得c = a(ia)。

编程注意:输入标志“legacy”将算法更改为与R2012b之前的MATLAB版本兼容。

参见:unique, union, intersect, setxor, ismember。

: c = setxor (a, b)

: c = setxor (a, b, "rows")

: c = setxor (…, "sorted")

: c = setxor (…, "stable")

: c = setxor (…, "legacy")

: [c, ia, ib] = setxor (…)

返回集合a或集合b独有的唯一元素。

如果a和b都是行向量，则返回一个行向量;否则，返回列向量。输入也可以是字符串的单元格数组。

如果给出了可选的输入"rows"，则返回集a和集b独占的行。输入必须是二维数字矩阵才能使用此选项。

可选参数“sorted”/“stable”控制唯一值在输出中出现的顺序。默认值为“sorted”，输出中的值按升序排列。另一种“稳定”保留输入中的顺序。

可选输出ia和ib是列索引向量，使得a(ia)和b(ib)是不相交的集合，其并集为c。

编程注意:输入标志“legacy”将算法更改为与R2012b之前的MATLAB版本兼容。

参见:unique, union, intersect, setdiff, ismember。

: tf = ismember (a, s)

: tf = ismember (a, s, "rows")

: [tf, s\_idx] = ismember (…)

返回一个与a形状相同的逻辑矩阵tf，如果a中的元素在s中找到，则为真(1)，如果不在s中找到，则为假(0)。

如果请求第二个输出参数，则还将返回每个匹配元素的s索引。

a = [3, 10, 1];

s = [0:9];

[tf, s\_idx] = ismember (a, s)

⇒ tf = [1, 0, 1]

⇒ s\_idx = [4, 0, 2]

输入a和s也可以是单元格数组。

a = {"abc"};

s = {"abc", "def"};

[tf, s\_idx] = ismember (a, s)

⇒ tf = 1

⇒ s\_idx = 1

如果给出了可选的第三个参数“rows”，则将a中的行与s中的行进行比较。输入必须是具有相同列数的二维矩阵才能使用此选项。

a = [1:3; 5:7; 4:6];

s = [0:2; 1:3; 2:4; 3:5; 4:6];

[tf, s\_idx] = ismember (a, s, "rows")

⇒ tf = logical ([1; 0; 1])

⇒ s\_idx = [2; 0; 5];

参见: lookup, unique, union, intersect, setdiff, setxor, ismembertol。

: tf = ismembertol (a, s)

: tf = ismembertol (a, s, tol)

: tf = ismembertol (a, s, name, value)

: [tf, s\_idx] = ismembertol (…)

检查值是否为在容差范围内的集合的成员。

此函数返回与a形状相同的逻辑矩阵tf，如果a中的元素在公差范围内接近s，则为真(1)，如果不是，则为假(0)。如果未指定tol，则使用默认公差1e-6。

如果请求第二个输出参数，则还将返回每个匹配元素的s索引。

输入a和s必须是数值。

a = [3, 10, 1];

s = [0:9];

[tf, s\_idx] = ismembertol (a, s)

⇒ tf = [1, 0, 1]

⇒ s\_idx = [4, 0, 2]

可选的属性/值对可以用来改变函数的行为。属性可以是以下字符串之一:

"ByRows"

如果设置为false(默认)，则分别处理a和s中的所有元素。如果设置为true，则对于在给定容差范围内匹配s中的一行的a中的每一行，tf将为true。如果满足条件all (abs (u-v) <= tol\*max (abs ([a;s])))，则u和v在容差范围内。

"OutputAllIndices"

如果设置为false(默认)，s\_idx包含其中一个匹配项的索引。如果设置为true, s\_idx是一个单元格数组，包含s中所有元素的索引，这些元素的索引在a中对应值的容差范围内。

"DataScale"

提供的值DS用于将公差试验中的比例因子更改为abs (u-v) <= tol\*DS。默认情况下，使用a和s中的最大绝对值作为比例因子。

例子:

s = [1:6].' \* pi;

a = 10.^log10 (x);

[tf, s\_idx] = ismembertol (a, s);

参见:ismember, lookup, unique, union, intersect, setdiff, setxor。

: p = powerset (a)

: p = powerset (a, "rows")

计算集合a的幂集(所有子集)。

集合a必须是数值矩阵或字符串的单元格数组。输出将始终是向量或字符串的单元格数组。

使用可选参数“rows”，集合a的每一行都被认为是集合的一个元素。要使用此参数，输入必须是二维数值矩阵。

参见:unique, union, intersect, setdiff, setxor, ismember。

**28多项式操作**

在Octave中，多项式由其系数表示(按降序排列)。例如，长度为N+1的向量c对应于下面的N阶多项式

p(x) = c(1) x^N + … + c(N) x + c(N+1).

**28.1求多项式**

由向量c表示的多项式的值可以很容易地在点x处求值，如下面的例子所示:

N = length (c) - 1;

val = dot (x.^(N:-1:0), c);

虽然上面的例子显示了计算多项式的值是多么容易，但它并不是最稳定的算法。对于较大的多项式，您应该使用更优雅的算法，例如霍纳方法，这正是Octave函数polyval所做的。

在x是方阵的情况下，c给出的多项式仍然是定义良好的。当x是一个标量时，明显的实现很容易用Octave表示，但在这种情况下，更优雅的算法也会表现得更好。polyvalm函数提供了这样一个算法。

: y = polyval (p, x)

: y = polyval (p, x, [], mu)

: [y, dy] = polyval (p, x, s)

: [y, dy] = polyval (p, x, s, mu)

求多项式p在x的指定值处的值。

如果x是一个向量或矩阵，则对x的每个元素求多项式的值。

当mu存在时，求(x - mu(1)) / mu(2)的多项式。

除了评估多项式之外，第二个输出表示预测区间y +/- dy，其中包含至少50%的未来预测。为了计算预测区间，必须提供源自polyfit的结构化变量s。

参见:polyvalm, polyaffine, polyfit, roots, poly。

: y = polyvalm (c, x)

在矩阵意义上计算一个多项式。

Polyvalm (c, x)将在矩阵意义上计算多项式，即使用矩阵乘法而不是像polyval中使用的逐元素乘法。

参数x必须是一个方阵。

参见:polyval, roots, poly。

**28.2查找根**

八度曲可以找到一个给定多项式的根。这是通过计算多项式的伴矩阵(见compan函数的定义)，然后找到它的特征值来完成的。

: r = roots (c)

计算多项式c的根。

对于有N个分量的向量c，返回多项式的根

c(1) \* x^(N-1) + … + c(N-1) \* x + c(N)

作为示例，下面的代码查找二次多项式的根

p(x) = x^2 - 5.

c = [1, 0, -5];

roots (c)

⇒ 2.2361

⇒ -2.2361

注意，真正的结果是+/- sqrt(5)，大约是+/- 2.2361。

参见:poly, company, fzero。

: z = polyeig (C0, C1, …, Cl)

: [v, z] = polyeig (C0, C1, …, Cl)

求解l次多项式特征值问题。

给定一个nxn矩阵多项式

C(s) = C0 + C1 s + … + Cl s^l

聚能解决特征值问题

(C0 + C1 z + … + Cl z^l) v = 0.

注意特征值z是矩阵多项式的零。Z是一个有n\*l个元素的行向量。V是一个矩阵(n x n\*l)它的列对应于特征向量。

参见:eig, eigs, company。

: A = compan (c)

计算多项式系数向量c对应的伴矩阵。

伴矩阵是

\_ \_

| -c(2)/c(1) -c(3)/c(1) … -c(N)/c(1) -c(N+1)/c(1) |

| 1 0 … 0 0 |

| 0 1 … 0 0 |

A = | . . . . . |

| . . . . . |

| . . . . . |

|\_ 0 0 … 1 0 \_|

伴随矩阵的特征值等于多项式的根。

参见:roots, poly, eight。

: [multp, idxp] = mpoles (p)

: [multp, idxp] = mpoles (p, tol)

: [multp, idxp] = mpoles (p, tol, reorder)

确定p中的唯一极点及其相关的多重性。

默认情况下，输出顺序从大小最大的极点到大小最小的极点。

如果两个极点之间的差小于相对公差，则认为它们是倍数。

abs (p1 - p0) / abs (p0) < tol

如果极点为0，则不进行缩放，并且tol被解释为绝对公差。tol的默认值是0.001。

如果可选参数reorder为false/ 0，则不对极点排序。

输出倍数是一个指定极点多重性的向量。multp(n)是指n极p(idxp(n))的多重性。

例如:

p = [2 3 1 1 2];

[m, n] = mpoles (p)

⇒ m = [1; 1; 2; 1; 2]

⇒ n = [2; 5; 1; 4; 3]

⇒ p(n) = [3, 2, 2, 1, 1]

参见:残，聚，根，conv, deconv。

**28.3多项式的乘积**

: y = conv (a, b)

: y = conv (a, b, shape)

卷积两个向量a和b。

当a和b是两个多项式的系数向量时，卷积表示乘积多项式的系数向量。

结果的大小由可选的shape参数决定，该参数接受以下值

shape = "full"

返回完整的卷积。(默认)结果是一个长度等于length (a) + length (b) - 1的向量。

shape = "same"

返回卷积的中心部分，其大小与a相同。

shape = "valid"

只返回不包含零填充边的部分。结果的大小为max (size (a) - size (b) + 1,0)。

参见:deconv, conv2, convn, fftconv。

: C = convn (A, B)

: C = convn (A, B, shape)

返回A和B的n-D卷积。

结果的大小由可选的shape参数决定，该参数接受以下值

shape = "full"

返回完整的卷积。(默认)

shape = "same"

返回卷积的中心部分，其大小与a相同。卷积的中心部分从索引层开始([size(B)/2] + 1)。

shape = "valid"

只返回不包含零填充边的部分。结果的大小为max (size (A) - size (B) + 1,0)。

参见:conv2, conv。

: b = deconv (y, a)

: [b, r] = deconv (y, a)

反卷积两个向量(多项式除法)。

[b, r] = deconv (y, a)解出b和r使得y = conv (a, b) + r。

如果y和a是多项式系数向量，b将包含多项式商的系数，r将是最低阶的剩余多项式。

参见:conv，渣。

: C = conv2 (A, B)

: C = conv2 (v1, v2, m)

: C = conv2 (…, shape)

返回A和B的二维卷积。

结果的大小由可选的shape参数决定，该参数接受以下值

shape = "full"

返回完整的卷积。(默认)

shape = "same"

返回卷积的中心部分，其大小与a相同。卷积的中心部分从索引层开始([size(B)/2] + 1)。

shape = "valid"

只返回不包含零填充边的部分。结果的大小为max (size (A) - size (B) + 1,0)。

当第三个参数是一个矩阵时，返回矩阵m在列方向上与向量v1的卷积以及在行方向上与向量v2的卷积。

参见:conv, convn。

: q = polygcd (b, a)

: q = polygcd (b, a, tol)

求两个多项式的最大公约数。

它等价于所有公根相乘得到的多项式。与反卷积一起，你可以减少两个多项式的比率。

公差工具默认为sqrt (eps)。

注意:这是一个数值不稳定的算法，不应该用于大型多项式。

示例代码:

polygcd (poly (1:8), poly (3:12)) - poly (3:8)

⇒ [ 0, 0, 0, 0, 0, 0, 0 ]

deconv (poly (1:8), polygcd (poly (1:8), poly (3:12))) - poly (1:2)

⇒ [ 0, 0, 0 ]

参见:poly, roots, conv, deconv, residue。

: [r, p, k, e] = residue (b, a)

: [b, a] = residue (r, p, k)

: [b, a] = residue (r, p, k, e)

第一个调用形式计算多项式b和a的商的部分分式展开。

商定义为

B(s) M r(m) N

---- = SUM ------------- + SUM k(i)\*s^(N-i)

A(s) m=1 (s-p(m))^e(m) i=1

其中M是极点的个数(r、p和e的长度)，k向量是N-1阶的多项式，表示直接贡献，e向量指定第M个残数极点的多重性。

例如,

b = [1, 1, 1];

a = [1, -5, 8, -4];

[r, p, k, e] = residue (b, a)

⇒ r = [-2; 7; 3]

⇒ p = [2; 2; 1]

⇒ k = [](0x0)

⇒ e = [1; 2; 1]

哪个表示下面的部分分式展开

s^2 + s + 1 -2 7 3

------------------- = ----- + ------- + -----

s^3 - 5s^2 + 8s - 4 (s-2) (s-2)^2 (s-1)

第二个调用形式执行逆运算，并从部分分式展开计算多项式的重构商b(s)/a(s);由残数、极点和一个由r、p和k指定的直接多项式表示，以及极点的多重性e。

如果没有明确指定多重性e，则多重性由函数极点决定。

例如:

r = [-2; 7; 3];

p = [2; 2; 1];

k = [1, 0];

[b, a] = residue (r, p, k)

⇒ b = [1, -5, 9, -3, 1]

⇒ a = [1, -5, 8, -4]

where mpoles is used to determine e = [1; 2; 1]

或者，多重性可以明确定义，例如，

r = [7; 3; -2];

p = [2; 1; 2];

k = [1, 0];

e = [2; 1; 1];

[b, a] = residue (r, p, k, e)

⇒ b = [1, -5, 9, -3, 1]

⇒ a = [1, -5, 8, -4]

哪个表示下面的部分分式展开

-2 7 3 s^4 - 5s^3 + 9s^2 - 3s + 1

----- + ------- + ----- + s = --------------------------

(s-2) (s-2)^2 (s-1) s^3 - 5s^2 + 8s - 4

参见:mpoles, poly, roots, conv, deconv。

**28.4导数/积分/变换**

Octave附带了计算多项式导数和积分的函数。函数polyder和polyint都返回描述结果的新多项式。作为一个例子，我们计算p(x) = x^2 + 1从0到3的定积分。

c = [1, 0, 1];

integral = polyint (c);

area = polyval (integral, 3) - polyval (integral, 0)

⇒ 12

: k = polyder (p)

: k = polyder (a, b)

: [q, d] = polyder (b, a)

返回多项式导数的系数，其系数由向量p给出。

如果给定一对多项式，则返回乘积a\*b的导数。

如果给定两个输入和两个输出，则返回多项式商b/a的导数。商分子是q，分母是d。

参见:polyint, polyval, polyreduce。

: q = polyint (p)

: q = polyint (p, k)

返回多项式的积分系数，其系数由向量p表示。

变量k是积分常数，默认值是0。

参见:polyder, polyval。

: g = polyaffine (f, mu)

返回仿射变换后多项式向量f的系数。

如果f是表示多项式f(x)的向量，则g = polyaffine (f, mu)是表示:

g(x) = f( (x - mu(1)) / mu(2) )

参见:polyval, polyfit。

**28.5多项式插值**

Octave具有对各种插值的良好支持，其中大多数在插值中有描述。前面章节中描述的函数的一个简单的替代方法是将单个多项式或分段多项式(样条)拟合到某些给定的数据点上。为了避免高度波动的多项式，人们通常希望对数据拟合低阶多项式。这通常意味着需要在最小二乘意义上拟合多项式，这正是polyfit函数所做的。

: p = polyfit (x, y, n)

: [p, s] = polyfit (x, y, n)

: [p, s, mu] = polyfit (x, y, n)

返回次数为n的多项式p(x)的系数，该系数使拟合到点[x(:)， y(:)]的最小二乘误差最小化。

N通常为≥0的整数，表示近似多项式的程度。如果n是逻辑向量，则将其用作掩码，以选择性地强制使用或忽略相应的多项式系数。

在行向量p中返回多项式系数。输出p可以直接与polyval一起使用，使用拟合的多项式来估计值。

可选输出s是一个包含以下字段的结构:

‘yf’

每个x的多项式的值。

‘X’

Vandermonde矩阵用于计算多项式系数。

‘R’

从QR分解得到三角形因子R。

‘C’

未缩放的协方差矩阵，形式上等于x ' \*x的倒数，但以最小化舍入误差传播的方式计算。

‘df’

自由度。

‘normr’

残差的范数。

第二个输出可由polyval用来计算预测值的统计误差限。特别地，p个系数的标准差由

sqrt (diag (s.C)/s.df) \* s.normr.

当第三个输出mu存在时，原始数据被居中并缩放，这可以提高拟合的数值稳定性。系数p与一个多项式有关

xhat = (x - mu(1)) / mu(2)

其中mu(1) = mean (x) mu(2) = STD (x)

例1:逻辑n和整数n

f = @(x) x.^2 + 5; # data-generating function

x = 0:5;

y = f (x);

## Fit data to polynomial A\*x^3 + B\*x^1

p = polyfit (x, y, logical ([1, 0, 1, 0]))

⇒ p = [ 0.0680, 0, 4.2444, 0 ]

## Fit data to polynomial using all terms up to x^3

p = polyfit (x, y, 3)

⇒ p = [ -4.9608e-17, 1.0000e+00, -3.3813e-15, 5.0000e+00 ]

参见:polyval, polyaffine, roots, vander, zscore。

在单个多项式不够好的情况下，解决方案是使用几个多项式拼凑在一起。函数样条将分段多项式(样条)拟合到一组数据上。

: pp = splinefit (x, y, breaks)

: pp = splinefit (x, y, p)

: pp = splinefit (…, "periodic", periodic)

: pp = splinefit (…, "robust", robust)

: pp = splinefit (…, "beta", beta)

: pp = splinefit (…, "order", order)

: pp = splinefit (…, "constraints", constraints)

对噪声数据x和y拟合一个分段三次样条曲线。

x是一个向量，y是一个向量或N-D数组。如果y是N-D数组，则x(j)与y(:，…，:，j)匹配。

P是一个正整数，它定义了沿x的间隔的数量，P +1是断点的数量。每个区间的点数相差不超过1。

可选属性periodic是一个逻辑值，它指定是否将周期边界条件应用于样条。周期的长度为max(断点)- min(断点)。默认值为false。

可选属性robust是一个逻辑值，它指定是否应用鲁棒拟合来减少离群数据点的影响。进行了三次加权最小二乘迭代。权重由之前的残差计算。异常值识别的灵敏度由属性β控制。beta的值被限制在0 < beta < 1的范围内。默认值是beta = 1/2。接近0的值给予所有数据相等的权重。增大的贝塔值减少了离群数据的影响。接近统一的价值观可能导致不稳定或等级不足。

拟合的样条返回为分段多项式pp，并可使用ppval求值。

样条由次阶多项式构成。默认是立方，order=3。P条样条有P+个阶自由度。在周期边界条件下，自由度降为P。

可选属性constraints是一个结构，指定对拟合的线性约束。该结构有三个字段:“xc”、“yc”和“cc”。

"xc"

约束的x坐标向量。

“yc”

位置xc的约束值。默认值是一个零数组。

“cc”

系数(矩阵)。默认值是1的数组。行数受限于分段多项式的阶数。

约束是0阶到1阶导数的线性组合

cc(1,j) \* y(xc(j)) + cc(2,j) \* y'(xc(j)) + ... = yc(:,...,:,j).

参见:interp1, unmkpp, ppval, spline, pchip, ppder, ppint, pp跳转。

用于构造分段多项式的断裂(或结)的数量是抑制输入数据x和y中存在的噪声的重要因素。下面的示例演示了这一点。

x = 2 \* pi \* rand (1, 200);

y = sin (x) + sin (2 \* x) + 0.2 \* randn (size (x));

## Uniform breaks

breaks = linspace (0, 2 \* pi, 41); % 41 breaks, 40 pieces

pp1 = splinefit (x, y, breaks);

## Breaks interpolated from data

pp2 = splinefit (x, y, 10); % 11 breaks, 10 pieces

## Plot

xx = linspace (0, 2 \* pi, 400);

y1 = ppval (pp1, xx);

y2 = ppval (pp2, xx);

plot (x, y, ".", xx, [y1; y2])

axis tight

ylim auto

legend ({"data", "41 breaks, 40 pieces", "11 breaks, 10 pieces"})

其结果如图28.1所示。

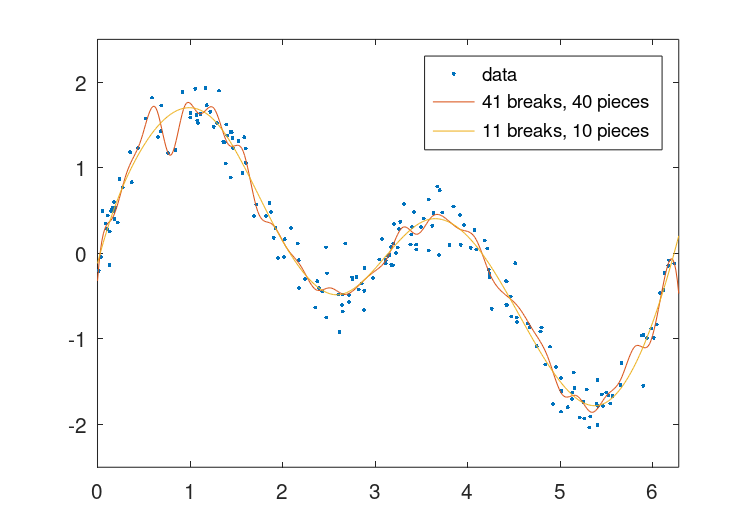


图28.1:41次断裂的分段多项式与11次断裂的分段多项式的拟合比较。与大量断裂的拟合表现出在基础函数中不存在的快速波纹。

分段多项式拟合，由样条提供，有连续的导数到1阶。例如，三次拟合具有连续的一阶和二阶导数。下面的代码演示了这一点

## Data (200 points)

x = 2 \* pi \* rand (1, 200);

y = sin (x) + sin (2 \* x) + 0.1 \* randn (size (x));

## Piecewise constant

pp1 = splinefit (x, y, 8, "order", 0);

## Piecewise linear

pp2 = splinefit (x, y, 8, "order", 1);

## Piecewise quadratic

pp3 = splinefit (x, y, 8, "order", 2);

## Piecewise cubic

pp4 = splinefit (x, y, 8, "order", 3);

## Piecewise quartic

pp5 = splinefit (x, y, 8, "order", 4);

## Plot

xx = linspace (0, 2 \* pi, 400);

y1 = ppval (pp1, xx);

y2 = ppval (pp2, xx);

y3 = ppval (pp3, xx);

y4 = ppval (pp4, xx);

y5 = ppval (pp5, xx);

plot (x, y, ".", xx, [y1; y2; y3; y4; y5])

axis tight

ylim auto

legend ({"data", "order 0", "order 1", "order 2", "order 3", "order 4"})

其结果如图28.2所示。

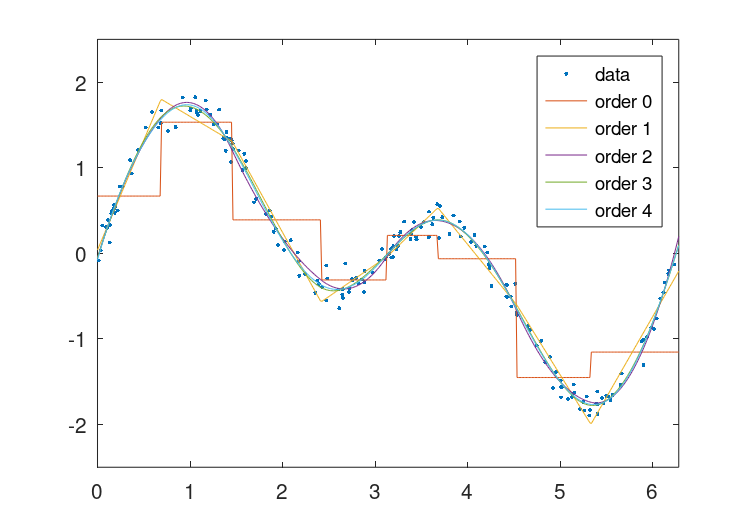


图28.2:分段常数多项式、线性多项式、二次多项式、三次多项式和四次多项式与噪声数据的比较高阶解更准确地表示底层函数，但代价是计算复杂性。

当提供拟合的底层函数是周期性的时，样条分析法能够应用所需的边界条件来显示周期性拟合。下面的代码演示了这一点。

## Data (100 points)

x = 2 \* pi \* [0, (rand (1, 98)), 1];

y = sin (x) - cos (2 \* x) + 0.2 \* randn (size (x));

## No constraints

pp1 = splinefit (x, y, 10, "order", 5);

## Periodic boundaries

pp2 = splinefit (x, y, 10, "order", 5, "periodic", true);

## Plot

xx = linspace (0, 2 \* pi, 400);

y1 = ppval (pp1, xx);

y2 = ppval (pp2, xx);

plot (x, y, ".", xx, [y1; y2])

axis tight

ylim auto

legend ({"data", "no constraints", "periodic"})

其结果如图28.3所示。

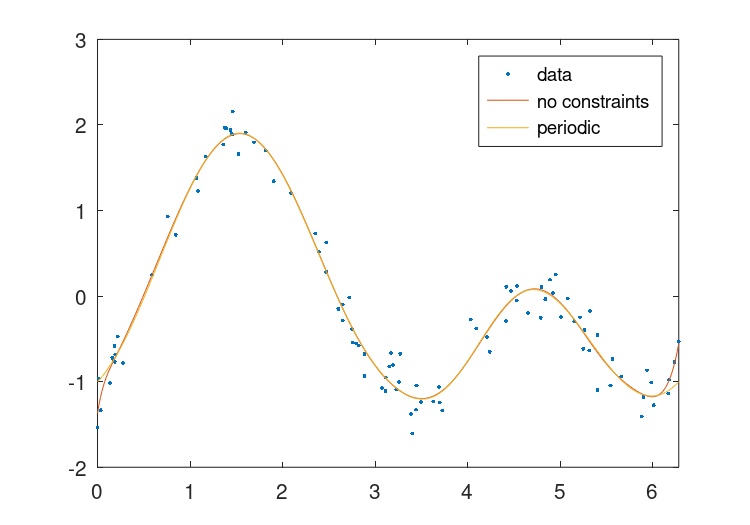


图28.3:有周期边界条件和没有周期边界条件的有噪声周期函数的分段多项式拟合比较。

还可以添加更复杂的约束。例如，下面的代码演示了在端点处夹紧值的周期性拟合，以及在端点处铰接的第二个周期性拟合。

## Data (200 points)

x = 2 \* pi \* rand (1, 200);

y = sin (2 \* x) + 0.1 \* randn (size (x));

## Breaks

breaks = linspace (0, 2 \* pi, 10);

## Clamped endpoints, y = y' = 0

xc = [0, 0, 2\*pi, 2\*pi];

cc = [(eye (2)), (eye (2))];

con = struct ("xc", xc, "cc", cc);

pp1 = splinefit (x, y, breaks, "constraints", con);

## Hinged periodic endpoints, y = 0

con = struct ("xc", 0);

pp2 = splinefit (x, y, breaks, "constraints", con, "periodic", true);

## Plot

xx = linspace (0, 2 \* pi, 400);

y1 = ppval (pp1, xx);

y2 = ppval (pp2, xx);

plot (x, y, ".", xx, [y1; y2])

axis tight

ylim auto

legend ({"data", "clamped", "hinged periodic"})

结果如图28.4所示。

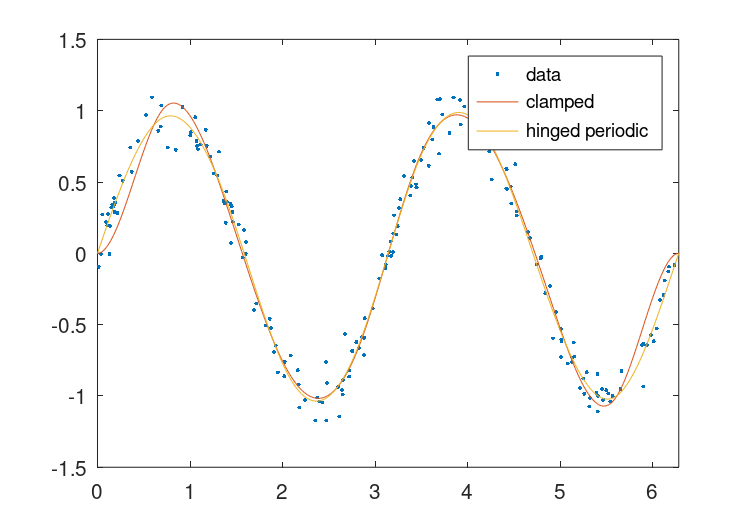


图28.4:两个周期分段三次拟合对噪声周期信号的比较。一种配合其端点夹紧，另一种配合其端点铰接。

样条函数还提供了鲁棒拟合的便利，其中减少了离群数据的影响。在下面的示例中，提供了三种不同的匹配。两个具有不同水平的异常值抑制，第三个说明非鲁棒解决方案。

## Data

x = linspace (0, 2\*pi, 200);

y = sin (x) + sin (2 \* x) + 0.05 \* randn (size (x));

## Add outliers

x = [x, linspace(0,2\*pi,60)];

y = [y, -ones(1,60)];

## Fit splines with hinged conditions

con = struct ("xc", [0, 2\*pi]);

## Robust fitting, beta = 0.25

pp1 = splinefit (x, y, 8, "constraints", con, "beta", 0.25);

## Robust fitting, beta = 0.75

pp2 = splinefit (x, y, 8, "constraints", con, "beta", 0.75);

## No robust fitting

pp3 = splinefit (x, y, 8, "constraints", con);

## Plot

xx = linspace (0, 2\*pi, 400);

y1 = ppval (pp1, xx);

y2 = ppval (pp2, xx);

y3 = ppval (pp3, xx);

plot (x, y, ".", xx, [y1; y2; y3])

legend ({"data with outliers","robust, beta = 0.25", ...

"robust, beta = 0.75", "no robust fitting"})

axis tight

ylim auto

结果如图28.5所示。

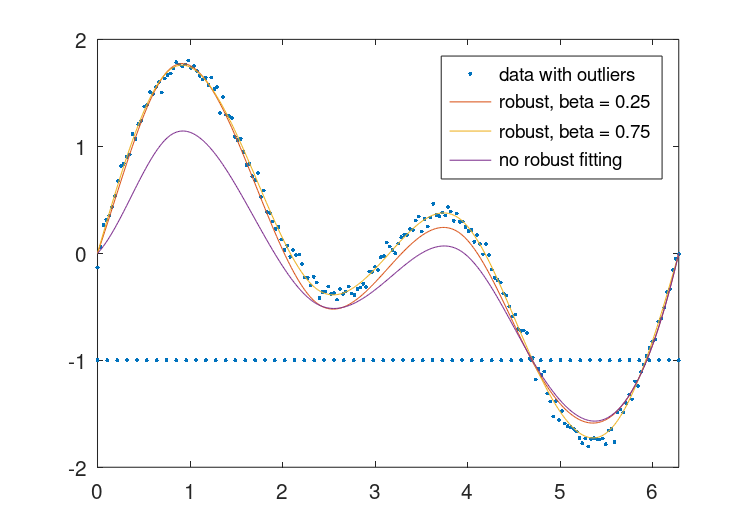


图28.5:两种不同水平的稳健拟合(beta = 0.25和0.75)对噪声数据和离群数据的比较。还包括没有稳健拟合(beta = 0)的传统拟合。

多项式解释的一种非常特殊的形式是帕岱尔近似。对于控制系统，连续时滞可以非常简单地用近似来建模。

: [num, den] = padecoef (T)

: [num, den] = padecoef (T, N)

以传递函数形式计算连续时滞T的n阶逼近。

exp (-sT)的帕德罗近似由下式定义

Pn(s)

exp (-sT) ~ -------

Qn(s)

在Pn(s)和Qn(s)中,都是由以下表达式定义的nof - order rational函数

N (2N - k)!N! k

Pn(s) = SUM --------------- (-sT)

k=0 (2N)!k!(N - k)!

Qn(s) = Pn(-s)

输入T和N必须是非负的数值标量。如果N未指定，则默认为1。

输出行向量num和den包含分子和分母系数以s的降序次方。它们都是n阶多项式。

例如:

t = 0.1;

n = 4;

[num, den] = padecoef (t, n)

⇒ num =

1.0000e-04 -2.0000e-02 1.8000e+00 -8.4000e+01 1.6800e+03

⇒ den =

1.0000e-04 2.0000e-02 1.8000e+00 8.4000e+01 1.6800e+03

函数ppval计算由mkpp或其他方法创建的分段多项式，而unmkpp返回关于分段多项式的详细信息。

下面的例子展示了如何将两个线性函数和一个二次函数组合成一个函数。这些函数中的每一个都在相邻区间上表示。

x = [-2, -1, 1, 2];

p = [ 0, 1, 0;

1, -2, 1;

0, -1, 1 ];

pp = mkpp (x, p);

xi = linspace (-2, 2, 50);

yi = ppval (pp, xi);

plot (xi, yi);

: pp = mkpp (breaks, coefs)

: pp = mkpp (breaks, coefs, d)

从样本点、断点和系数系数构造一个分段多项式结构。

break必须是一个严格递增值的向量。间隔的数量由ni = length (breaks) - 1给出。

当m是多项式时，顺序序列的大小必须为:ni-by-(m + 1)。

coefs的第i行coefs(i，:)包含了第i个区间内多项式的系数，从最高(m)到最低(0)度排序。

Coefs也可以是多维数组，指定一个向量值或数组值多项式。在这种情况下，多项式m阶是由最后一个维度的长度定义的。第一个维度(s)的大小由标量或向量d给出。如果d未给定，则将其设置为1。在这种情况下，p(r, i，:)包含在区间i上定义的第r个多项式的系数。在任何情况下，coefs都被重塑为大小为[ni\*prod(d) m]的二维矩阵。

编程注意:ppval在xi - break (i)处求多项式，即从xi中减去当前区间的下端点。在使用mkpp创建分段多项式对象时必须考虑到这一点。

参见:unmkpp, ppval, spline, pchip, ppder, ppint, pp跳转。

: [x, p, n, k, d] = unmkpp (pp)

提取分段多项式结构pp的分量。

这个函数是mkpp的逆函数:它提取了创建分段多项式结构pp所需的mkpp的输入。下面的代码明确了这个关系:

[breaks, coefs, numinter, order, dim] = unmkpp (pp);

pp2 = mkpp (breaks, coefs, dim);

通过这种方法得到的分段多项式结构pp2与原pp完全相同。通过直接访问结构pp的场也可以得到相同的结果。

组成部分是:

x

采样点或断点。

p

样本区间内点的多项式系数。P (i，:)包含区间i上多项式的系数，从最高次到最低次排序。如果d > 1，则p是一个大小为[n\*prod(d) m]的矩阵，其中i + (1:d)行是区间i内所有d个多项式的系数。

n

多项式片段或区间的个数，n = length (x) - 1。

k

多项式的阶数加1。

d

为每个区间定义的多项式的个数。

参见:mkpp, ppval, spline, pchip。

: yi = ppval (pp, xi)

求分段多项式结构pp在点xi处的值。

如果pp描述一个标量多项式函数，则结果是一个与xi形状相同的数组。否则，如果xi是向量，则结果的大小为[pp.dim, length(xi)]，如果是多维数组则为[pp.dim, size(xi)]。

参见:mkpp, unmkpp, spline, pchip。

: ppd = ppder (pp)

: ppd = ppder (pp, m)

计算分段多项式结构pp的分段m阶导数。

如果省略m，则计算一阶导数。

参见:mkpp, ppval, ppint。

: ppi = ppint (pp)

: ppi = ppint (pp, c)

计算分段多项式结构pp的积分。

C，如果给定，是积分常数。

参见:mkpp, ppval, ppder。

: jumps = ppjumps (pp)

求分段多项式的边界跳变。

如果有n个区间，pp的维数为d，则得到的数组维数为[d, n-1]。

参见:mkpp。

**28.6其他功能**

: y = poly (A)

: y = poly (x)

如果A是一个N × N的平方矩阵，poly (A)是det (z \* eye (N) - A)系数的行向量，det (z \* eye (N) - A)是A的特征多项式。

例如，下面的代码找到A的特征值，它们是poly (A)的根。

roots (poly (eye (3)))

⇒ 1.00001 + 0.00001i

1.00001 - 0.00001i

0.99999 + 0.00000i

事实上，所有三个特征值都是1，这强调了数值性能应该使用eig函数来计算特征值。

如果x是一个向量，poly (x)是多项式的系数的向量，其根是x的元素。也就是说，如果c是一个多项式，则d = roots (poly (c))的元素包含在c中。然而，由于排序和数值误差，向量c和d并不相同。

参见:根，8。

: polyout (c)

: polyout (c, x)

: str = polyout (…)

显示多项式c的格式化版本。

格式化的多项式

c(x) = c(1) \* x^n + … + c(n) x + c(n+1)

作为字符串返回，如果nargout为零，则写入屏幕。

第二个参数x指定每个术语使用的变量名，默认为字符串“s”。

参见:polyreduce。

: p = polyreduce (c)

通过去掉任何前导零，将多项式系数向量减少到最小项数。

参见:polyout。

**29 插值**

**29.1一维插值**

Octave支持几种一维插值方法，其中大部分将在本节中介绍。多项式插值和离散数据插值描述了其他方法。

: yi = interp1 (x, y, xi)

: yi = interp1 (y, xi)

: yi = interp1 (…, method)

: yi = interp1 (…, extrap)

: yi = interp1 (…, "left")

: yi = interp1 (…, "right")

: pp = interp1 (…, "pp")

一维插值。

插值输入数据以确定yi在点xi处的值。如果未指定，则取x为y的索引(1:length (y))。如果y是矩阵或n维数组，则对y的每一列执行插值。

插值方法有:

"nearest"

返回最近的邻居。

"previous"

返回前一个邻居。

"next"

返回下一个邻居。

"linear" (default)

从最近邻的线性插值。

"pchip"

分段三次Hermite插值-光滑一阶导数多项式保形插值。

"cubic"

三次插值(与“pchip”相同)。

"spline"

三次样条插值-平滑的一阶和二阶导数在整个曲线。

在上述任何方法的开头加上“\*”，强制interp1假设x是均匀间隔的，并且只引用x(1)和x(2)。这通常更快，而且不会更慢。默认方法是“linear”。

如果extrap是字符串“extrap”，则使用当前方法推断端点之外的值。如果extrap是一个数字，则用该数字替换端点以外的值。当未指定时，extra默认为NA。

如果指定了字符串参数“pp”，则不应提供xi，并且interp1返回一个分段多项式对象。这个对象稍后可以与ppval一起使用来计算插值。有一个等价，ppval (interp1 (x, y, method， "pp")， xi) == interp1 (x, y, xi, method， "extra ")。

x中的重复点指定一个不连续的插值。最多可以有2个具有相同值的连续点。如果x在增加，默认的不连续插值是右连续的。如果x在减小，默认的不连续插值是左连续的。可通过使用“左”或“右”选项分别选择左连续或右连续插补来指定插补的连续性条件。不连续插值只允许“最接近”和“线性”方法;在所有其他情况下，x值必须是唯一的。

使用interp1的一个例子是

xf = [0:0.05:10];

yf = sin (2\*pi\*xf/5);

xp = [0:10];

yp = sin (2\*pi\*xp/5);

lin = interp1 (xp, yp, xf);

near = interp1 (xp, yp, xf, "nearest");

pch = interp1 (xp, yp, xf, "pchip");

spl = interp1 (xp, yp, xf, "spline");

plot (xf,yf,"r", xf,near,"g", xf,lin,"b", xf,pch,"c", xf,spl,"m",

xp,yp,"r\*");

legend ("original", "nearest", "linear", "pchip", "spline");

参见:pchip, spline, interpinterp2, interp3, interpn。

各种插值方法之间有一些重要的区别。“样条”方法强制内插值的一阶和二阶导数都具有连续导数，而其他方法则没有。这意味着“样条”方法的结果通常更平滑。如果要插值的函数实际上是光滑的，那么“样条法”将给出很好的结果。然而，如果要计算的函数在某种程度上是不连续的，那么“pchip”插值可能会给出更好的结果。

这可以通过代码进行演示

t = -2:2;

dt = 1;

ti =-2:0.025:2;

dti = 0.025;

y = sign (t);

ys = interp1 (t,y,ti,"spline");

yp = interp1 (t,y,ti,"pchip");

ddys = diff (diff (ys)./dti) ./ dti;

ddyp = diff (diff (yp)./dti) ./ dti;

figure (1);

plot (ti,ys,"r-", ti,yp,"g-");

legend ("spline", "pchip", 4);

figure (2);

plot (ti,ddys,"r+", ti,ddyp,"g\*");

legend ("spline", "pchip");

结果如图29.1和图29.2所示。

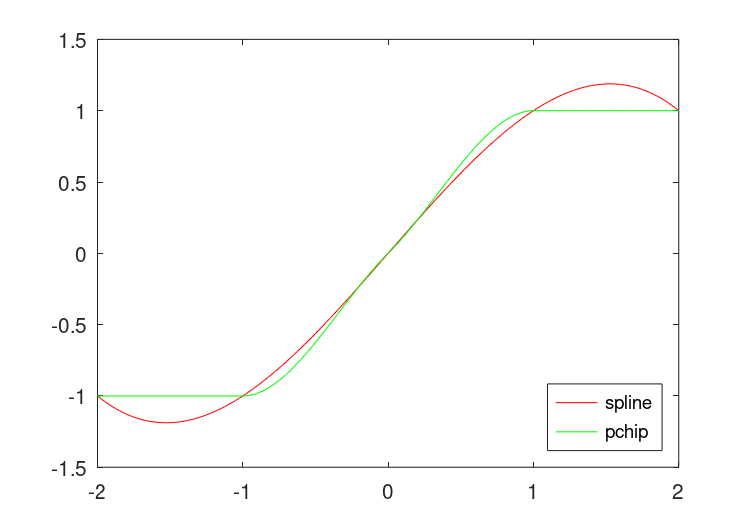


图29.1:“pchip”和“样条”插值方法对阶跃函数的比较

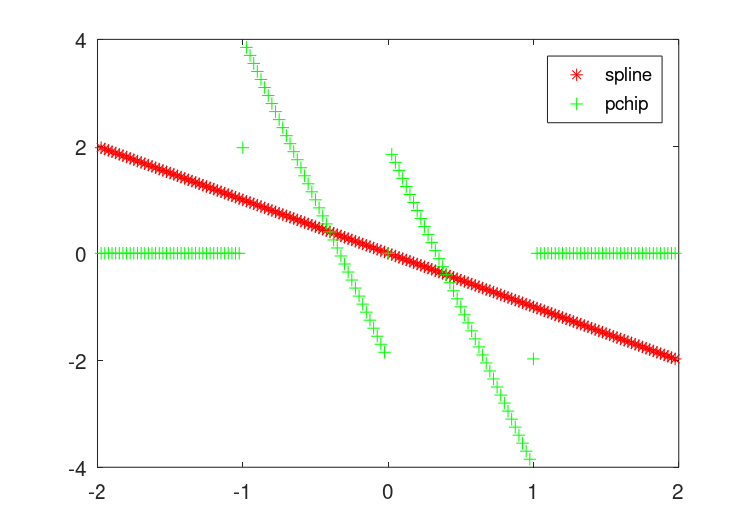


图29.2:一个阶跃函数的“pchip”插值方法和“样条”插值方法二阶导数的比较

傅里叶插值，是一种重采样技术将信号转换到频域，用零填充，然后重新转换到时域。

: y = interpft (x, n)

: y = interpft (x, n, dim)

傅里叶插值。

如果x是一个向量，那么用n个点对x进行重采样。假设x中的数据是相等的。如果x是矩阵或n维数组，则对x的每一列执行插值。

如果指定了dim，则沿着尺寸dim进行插值。

Interpft假设插值函数是周期性的，因此对插值的端点做了假设。

参见:interp1。

傅里叶插值有两个明显的局限性。首先，假设函数信号是周期性的，因此非周期性信号在边缘处表现得很差。其次，信号及其插值都需要在等距点进行采样。使用插入的一个例子是

t = 0 : 0.3 : pi; dt = t(2)-t(1);

n = length (t); k = 100;

ti = t(1) + [0 : k-1]\*dt\*n/k;

y = sin (4\*t + 0.3) .\* cos (3\*t - 0.1);

yp = sin (4\*ti + 0.3) .\* cos (3\*ti - 0.1);

plot (ti, yp, "g", ti, interp1 (t, y, ti, "spline"), "b", ...

ti, interpft (y, k), "c", t, y, "r+");

legend ("sin(4t+0.3)cos(3t-0.1)", "spline", "interpft", "data");

这证明了傅里叶插值对非周期函数的不良行为，如图29.3所示。

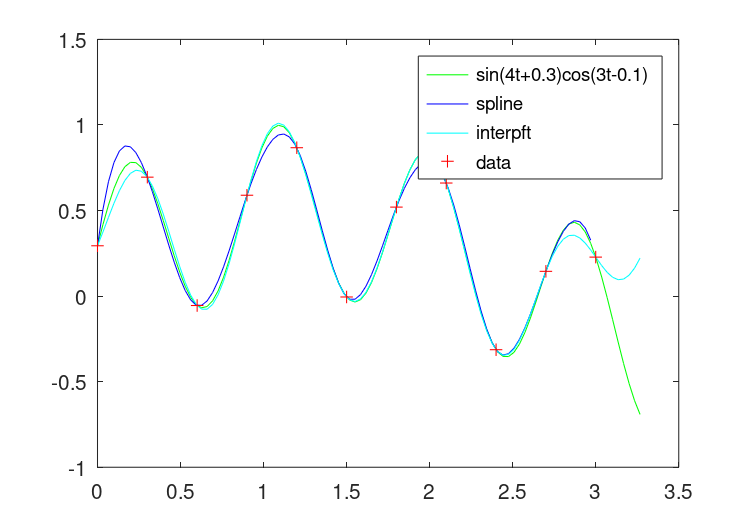


图29.3:非周期数据的interp1和interpft比较

此外，可以直接调用interp1函数下面的支持函数样条和查找。

: pp = spline (x, y)

: yi = spline (x, y, xi)

返回点x和y的三次样条插值。

当带两个参数调用时，返回分段多项式pp，它可以与ppval一起用于在特定点对多项式求值。

当使用第三个输入参数调用时，spline计算点xi处的样条曲线。第三种调用形式spline (x, y, xi)等价于ppval (spline (x, y)， xi)。

变量x必须是一个长度为n的向量。

Y可以是向量也可以是数组。如果y是一个向量，它的长度一定是n或n + 2。如果y的长度为n，则使用“非a结”结束条件。如果y的长度是n + 2，那么向量y的第一个和最后一个值就是三次样条在端点处的一阶导数的值。

如果y是一个数组，那么y的大小必须具有[s1, s2，…，sk, n]或[s1, s2，…，sk, n + 2]的形式。数组在内部被重塑为一个矩阵，其中前导维由s1 \* s2 \*…\* sk给出，然后该矩阵的每一行被单独处理。请注意，这与interp1完全相反，但是为了MATLAB兼容性而做的。

参见:pchip, ppval, mkpp, unmkpp。

**29.2多维插值**

在Octave中有三个多维插值函数，具有类似的功能。离散数据插值中描述了使用Delaunay镶嵌的方法。

: zi = interp2 (x, y, z, xi, yi)

: zi = interp2 (z, xi, yi)

: zi = interp2 (z, n)

: zi = interp2 (z)

: zi = interp2 (…, method)

: zi = interp2 (…, method, extrap)

二维插值。

插入参考数据x, y, z以确定坐标xi, yi处的zi。参考数据x, y可以是由meshgrid返回的矩阵，在这种情况下，x, y和z的大小必须相等。如果x, y是描述网格的向量，则长度(x) ==列(z)，长度(y) ==行(z)。在任何一种情况下，输入数据必须严格单调。

如果调用时没有x、y，只有一个参考数据矩阵z，则假定二维区域x = 1:列(z)， y = 1:行(z)。如果网格是规则的，并且点之间的距离不重要，则可以节省内存。

如果使用单个参考数据矩阵z和细化值n调用，则在每个原始间隔被递归细分n次的网格上执行插值。这导致在原始网格的每个间隔中有2^n-1个额外的点。如果省略n，则使用1的值。例如，当n==2时，区间[0,1]会得到一个点在[0,1 /4,1/2,3/ 4,1]处的精细区间。

插值方法有:

"nearest"

返回最近的邻居。

"linear" (default)

从最近邻的线性插值。

"pchip"

分段三次Hermite插值-光滑一阶导数多项式保形插值。

"cubic"

一阶导数光滑的卷积核函数三次插值方法。

"spline"

三次样条插值-平滑的一阶和二阶导数在整个曲线。

附加是一个标量。它将端点以外的值替换为extra。注意，如果使用extra，也必须指定method。如果省略了extrap并且方法是“样条”，则使用“样条”的外推值。否则，其他方法的默认附加值为“NA”。

参见:interp1, interp3, interpn, meshgrid。

: vi = interp3 (x, y, z, v, xi, yi, zi)

: vi = interp3 (v, xi, yi, zi)

: vi = interp3 (v, n)

: vi = interp3 (v)

: vi = interp3 (…, method)

: vi = interp3 (…, method, extrapval)

三维插值。

插入参考数据x, y, z, v，以确定坐标xi, yi, zi处的vi。参考数据x, y, z可以是矩阵，由meshgrid返回，在这种情况下，x, y, z和v的大小必须相等。如果x, y, z是描述立方体网格的向量，则长度(x) ==列(v)，长度(y) ==行(v)，长度(z) ==大小(v, 3)。在任何一种情况下，输入数据必须是严格单调的。

如果调用时没有x、y、z，只有一个参考数据矩阵v，则假定3- d区域x = 1:列(v)， y = 1:行(v)， z = 1:大小(v, 3)。如果网格是规则的，并且点之间的距离不重要，则可以节省内存。

如果使用单个参考数据矩阵v和细化值n调用，则在3-D网格上执行插值，其中每个原始间隔已递归细分n次。这导致在原始网格的每个间隔中有2^n-1个额外的点。如果省略n，则使用1的值。例如，当n==2时，区间[0,1]会得到一个点在[0,1 /4,1/2,3/ 4,1]处的精细区间。

插值方法有:

"nearest"

返回最近的邻居。

"linear" (default)

从最近邻的线性插值。

"cubic"

分段三次埃尔米特插值-光滑一阶导数多项式保形插值(尚未实现)。

"spline"

三次样条插值-平滑的一阶和二阶导数在整个曲线。

Extrapval是一个标量。它将端点以外的值替换为附加值。注意，如果使用extrapval，也必须指定method。如果省略了extrapval并且方法是“样条”，则使用“样条”的外推值。否则，其他方法的默认提取值为“NA”。

参见:interp1, interp2, interpn, meshgrid。

: vi = interpn (x1, x2, …, v, y1, y2, …)

: vi = interpn (v, y1, y2, …)

: vi = interpn (v, m)

: vi = interpn (v)

: vi = interpn (…, method)

: vi = interpn (…, method, extrapval)

执行n维插值，其中n至少为2。

n维数组v的每个元素表示由参数x1, x2，…，xn给出的位置上的一个值。参数x1, x2，…，xn要么是与“ndgrid”格式的数组v大小相同的n维数组，要么是向量。

参数y1, y2，…，yn表示数组vi被插值的点。它们可以是相同长度和方向的向量，在这种情况下，它们被解释为分散点的坐标。如果它们是不同方向或长度的向量，它们被用来形成“和网格”格式的网格。它们也可以是大小相等的n维数组。

如果省略x1，…，xn，则假定它们为x1 = 1: size (v, 1)，等等。如果指定了m，则插值在每个插值点之间的中间位置添加一个点。此过程执行m次。如果只指定了v，则假定m为1。

插值方法有:

"nearest"

返回最近的邻居。

"linear" (default)

从最近邻的线性插值。

"pchip"

分段三次埃尔米特插值-光滑一阶导数多项式保形插值(尚未实现)。

"cubic"

三次插值(与“pchip”相同[尚未实现])。

"spline"

三次样条插值-平滑的一阶和二阶导数在整个曲线。

默认方法是“linear”。

Extrapval是一个标量。它将端点以外的值替换为附加值。注意，如果使用extrapval，也必须指定method。如果省略了extrapval并且方法是“样条”，则使用“样条”的外推值。否则，任何其他方法的默认附加值都是NA。

参见:interp1, interp2, interp3, spline, and grid。

interpn和其他两个多维插值函数之间的一个显著区别是处理维度的方式。对于interp2和interp3, y轴被认为是矩阵的列，而x轴对应于数组的行。由于Octave按列主顺序对数组进行索引，因此任何数组的第一个维度都是列，因此interpn有效地反转了' x '和' y '维度。考虑一下这个例子，

x = y = z = -1:1;

f = @(x,y,z) x.^2 - y - z.^2;

[xx, yy, zz] = meshgrid (x, y, z);

v = f (xx,yy,zz);

xi = yi = zi = -1:0.1:1;

[xxi, yyi, zzi] = meshgrid (xi, yi, zi);

vi = interp3 (x, y, z, v, xxi, yyi, zzi, "spline");

[xxi, yyi, zzi] = ndgrid (xi, yi, zi);

vi2 = interpn (x, y, z, v, xxi, yyi, zzi, "spline");

mesh (zi, yi, squeeze (vi2(1,:,:)));

其中vi和vi2是相同的。分别在meshgrid和ndgrid函数中处理尺寸的反转。此代码的结果如图29.4所示。

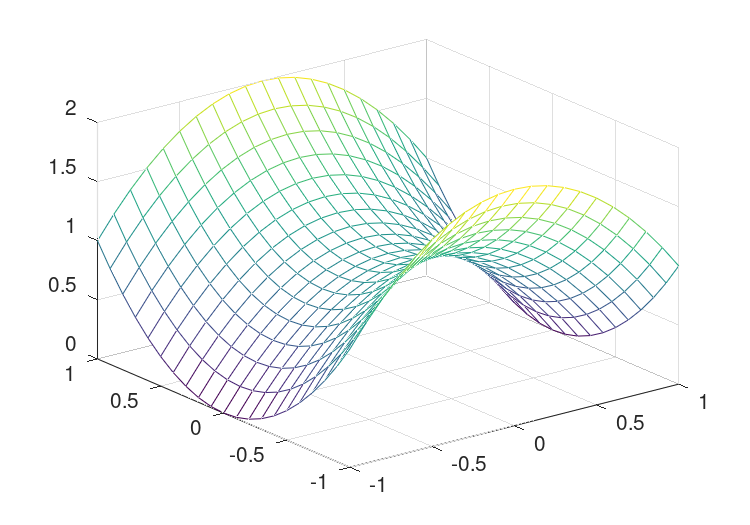


图29.4:使用interpn的演示

**30几何**

Octave中的大部分几何代码都是基于Qhull库的。Qhull的一些文档，特别是可以传递给delaunay, voronoi和convhull等的选项，与Octave用户相关。

**30.1 Delaunay三角测量**

德劳内三角剖分是由一组圆构成的。选择这些圆，使集合中至少有三个点在圆的圆周上进行三角剖分。点的集合中没有一个点落在任何一个圆内。

一般来说，任何圆周上都只有三个点。然而，在某些情况下，特别是在规则网格的情况下，一个圆周上可以有4个或更多的点。在这种情况下，德劳内三角剖分法并不是唯一的。

: tri = delaunay (x, y)

: tetr = delaunay (x, y, z)

: tri = delaunay (x)

: tri = delaunay (…, options)

计算二维或三维点集的Delaunay三角剖分。

对于二维集合，返回值tri是一个满足Delaunay圆周准则的三角形集合，即[x, y]没有数据点在定义三角形的圆周内。三角形集合tri是一个大小为[n, 3]的矩阵。每行定义一个三角形，三列是三角形的三个顶点。tri(i,j)的值是第i个三角形的第j个顶点的位置到x和y的索引。

对于三维集合，返回值tetr是满足Delaunay圆圆准则的四面体集合，即从[x, y, z]中没有数据点在定义的四面体的圆圆内。四面体集合是一个大小为[n, 4]的矩阵。每行定义一个四面体，四列是四面体的四个顶点。tetr(i,j)的值是指向x, y, z的索引，用于表示第i个四面体的第j个顶点的位置。

输入x也可以是一个有两列或三列的矩阵，其中第一列包含x数据，第二列包含y数据，可选的第三列包含z数据。

一个可选的最后参数必须是字符串或字符串的单元格数组，它包含传递给底层qhull命令的选项。有关详细信息，请参阅Qhull库的文档http://www.qhull.org/html/qh-quick.htm#options。默认选项为{"Qt"， "Qbb"， "Qc"}。如果Qhull在二维输入中失败，则使用{"Qt"， "Qbb"， "Qc"， "Qz"}选项再次尝试三角测量，这可能导致精度降低。

如果没有options或[]，则使用默认参数。否则，options将替换默认参数列表。要将用户选项附加到默认值，必须重复options中的默认参数。使用空字符串不传递任何参数。

x = rand (1, 10);

y = rand (1, 10);

tri = delaunay (x, y);

triplot (tri, x, y);

hold on;

plot (x, y, "r\*");

axis ([0,1,0,1]);

参见:delaunayn, convhull, voronoi, triplot, trimesh, tetramesh, trissurf。

对于3-D输入，delaunay返回一组满足delaunay圆周标准的四面体。类似地，delaunayn返回满足Delaunay圆周条件的n维单纯形。三角剖分的n维扩展称为镶嵌。

: T = delaunayn (pts)

: T = delaunayn (pts, options)

计算n维点集的Delaunay三角剖分。

Delaunay三角剖分是一组点的凸壳的镶嵌，使得n个三角形定义的n球不包含该集合中的任何其他点。

大小为[n, dim]的输入矩阵pts在维数为dim的空间中包含n个点。返回矩阵T的大小为[m, dim+1]。T的每一行包含一组返回到描述维度为dim的单纯形的原始点集的指标。例如，二维单纯形是一个三角形，三维单纯形是一个四面体。

第二个可选参数必须是字符串或由字符串组成的单元格数组，它包含传递给底层qhull命令的选项。有关详细信息，请参阅Qhull库的文档http://www.qhull.org/html/qh-quick.htm#options。默认选项取决于输入的尺寸:

* 2-D and 3-D: options = {"Qt", "Qbb", "Qc"}
* 4-D and higher: options = {"Qt", "Qbb", "Qc", "Qx"}

如果Qhull在二维输入中失败，则使用{"Qt"， "Qbb"， "Qc"， "Qz"}选项再次尝试三角测量，这可能导致精度降低。

如果没有options或[]，则使用默认参数。否则，options将替换默认参数列表。要将用户选项附加到默认值，必须重复options中的默认参数。使用空字符串不传递任何参数。

参见:delaunay, convhulln, voronoin, trimesh, tetramesh。

点集的德劳内三角剖分的一个例子是

rand ("state", 1);

x = rand (1, 10);

y = rand (1, 10);

T = delaunay (x, y);

X = [ x(T(:,1)); x(T(:,2)); x(T(:,3)); x(T(:,1)) ];

Y = [ y(T(:,1)); y(T(:,2)); y(T(:,3)); y(T(:,1)) ];

axis ([0, 1, 0, 1]);

plot (X, Y, "b", x, y, "r\*");

其结果如图30.1所示。

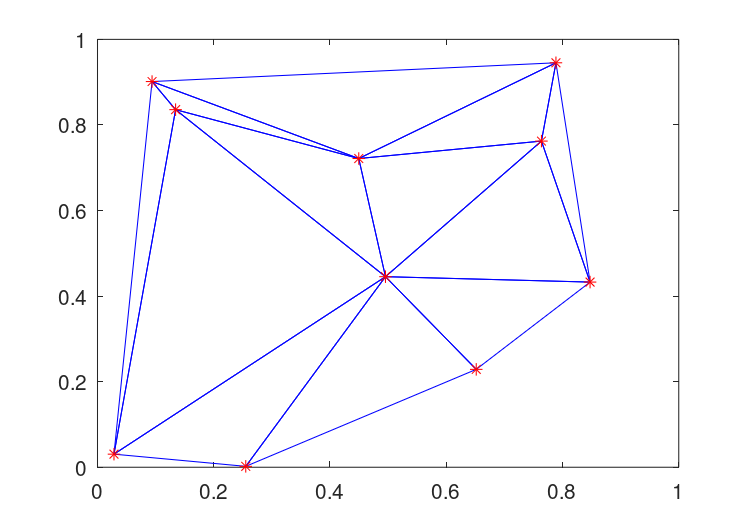


图30.1:随机点集的Delaunay三角剖分

**30.1.1绘制三角剖分图**

Octave具有triplot, trimesh和trissurf函数，用于绘制二维点集的Delaunay三角剖分。Tetramesh将绘制三维点集的三角剖分图。

: triplot (tri, x, y)

: triplot (tri, x, y, linespec)

: h = triplot (…)

绘制一个二维三角形网格。

tri通常是Delaunay三角剖分在x, y网格上的输出。tri的每一行代表一个三角形，并包含三个索引[x, y]，这些索引是x-y平面上三角形的顶点。

可以使用与plot命令格式相同的参数linespec来定义用于绘图的线样式。

可选的返回值h是创建的补丁对象的图形句柄。

参见:plot, trimesh, trissurf, delaunay。

: trimesh (tri, x, y, z, c)

: trimesh (tri, x, y, z)

: trimesh (tri, x, y)

: trimesh (…, prop, val, …)

: h = trimesh (…)

绘制一个三维三角形线框网格。

与使用矩形绘制网格的mesh不同，trimesh使用三角形绘制网格。

tri通常是Delaunay三角剖分在x, y网格上的输出。tri的每一行代表一个三角形，并包含三个索引[x, y]，这些索引是x-y平面上三角形的顶点。Z决定了每个顶点平面以上的高度。如果没有z输入，则三角形绘制为二维图形。

trimmesh的颜色是通过线性缩放z值来计算的，以适应当前颜色映射的范围。使用颜色和/或更改颜色映射来控制外观。

可选的是，网格的颜色可以通过提供c来独立于z来指定，c是颜色图数据的向量，或者是RGB数据的三列矩阵。c中指定的颜色数量必须等于z中顶点的数量或者等于tri中三角形的数量。

任何属性/值对都直接传递给底层补丁对象。完整的属性列表记录在补丁属性中。

可选的返回值h是创建的补丁对象的图形句柄。

参见:mesh, tetramesh, triplot, trissurf, delaunay, patch, hidden。

: trisurf (tri, x, y, z, c)

: trisurf (tri, x, y, z)

: trisurf (…, prop, val, …)

: h = trisurf (…)

绘制一个三维三角形曲面。

surf使用矩形绘制表面网格，而trissurf使用三角形绘制网格。

tri通常是Delaunay三角剖分在x, y网格上的输出。tri的每一行代表一个三角形，并包含三个索引[x, y]，这些索引是x-y平面上三角形的顶点。Z决定了每个顶点平面以上的高度。

trisurf的颜色是通过线性缩放z值来计算的，以适应当前颜色图的范围。使用颜色和/或更改颜色映射来控制外观。

可选的是，网格的颜色可以通过提供c来独立于z来指定，c是颜色图数据的向量，或者是RGB数据的三列矩阵。c中指定的颜色数量必须等于z中顶点的数量或者等于tri中三角形的数量。当指定每个顶点的颜色时，三角形将仅根据第一个顶点的颜色着色(参见补丁文档和设置为“flat”时的“FaceColor”属性)。

任何属性/值对都直接传递给底层补丁对象。完整的属性列表记录在补丁属性中。

可选的返回值h是创建的补丁对象的图形句柄。

参见:冲浪，三坐标，网格，delaunay, patch，阴影。

: tetramesh (T, X)

: tetramesh (T, X, C)

: tetramesh (…, property, val, …)

: h = tetramesh (…)

将m × 4矩阵T中定义的四面体显示为三维补丁。

T是典型的3-D点集合的Delaunay三角剖分的输出。T的每一行包含4个指标到四面体顶点的n × 3矩阵X。X中的每一行表示三维空间中的一个点。

向量C指定每个四面体的颜色作为当前颜色映射的索引。默认值是1:m，其中m是四面体的个数;索引被缩放以映射到颜色映射的全部范围。如果颜色图中的四面体多于颜色，则C中的值将循环重复。

调用tetramesh(…，"property"， "value"，…)将所有属性/值对作为附加参数直接传递给patch函数。完整的属性列表记录在补丁属性中。

可选返回值h是补丁句柄的向量，其中每个句柄按照t给出的顺序代表一个四面体。h的典型用例是将各自的补丁“可见”属性“打开”或“关闭”。

输入demo tetramesh查看使用tetramesh的示例。

参见:trimesh, delaunay, delaunayn, patch。

triplot与trimesh或trisurf之间的区别在于，前者仅绘制二维三角测量本身，而后者绘制函数f (x, y)的值。使用triplot函数的一个示例是

rand ("state", 2)

x = rand (20, 1);

y = rand (20, 1);

tri = delaunay (x, y);

triplot (tri, x, y);

它绘制了一组二维随机点的德劳内三角剖分图。上面的输出如图30.2所示。

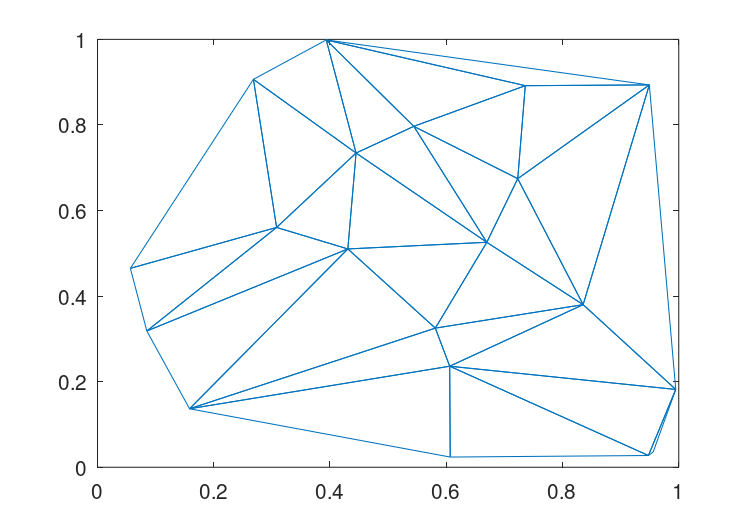


图30.2:随机点集的Delaunay三角剖分

**30.1.2三角剖分中的点识别**

通常需要确定n维空间中的特定点是否在该n维空间中点集的Delaunay镶嵌内，如果是，则哪个n -单纯形包含该点，以及镶嵌中哪个点最接近所需点。函数tsearch和dsearch在三角剖分中执行此功能，而在n维镶嵌中执行tsearch和dsearch。

为了确定由向量p表示的特定点是否属于n -单纯形的一个简单点，我们可以将该点的笛卡尔坐标写成关于n -单纯形的参数形式。这种参数形式称为该点的质心坐标。如果定义N-单纯形的点由N + 1个向量t(i，:)给出，那么定义点p的质心坐标由

p = beta \* t

其中包含N + 1个值，它们一起作为一个向量表示点p的质心坐标。为了确保值的唯一解，一个附加的标准是

sum (beta) == 1

，因此我们可以把上面的写成

p - t(end, :) = beta(1:end-1) \* (t(1:end-1, :)

- ones (N, 1) \* t(end, :)

我们可以解出

beta(1:end-1) = (p - t(end, :)) /

(t(1:end-1, :) - ones (N, 1) \* t(end, :))

beta(end) = sum (beta(1:end-1))

给出了点p的笛卡尔坐标到质心坐标的转换公式。重心坐标的一个重要性质是对于n -单纯形中的所有点

0 <= beta(i) <= 1

因此，在tsearch和tsearchn中的检验，本质上只需要用n -单纯形的每个单形的质心坐标来表示每个点，并检验beta的值。这正是在搜索中使用的实现。搜索是针对二维优化的，并且没有明确地形成重心坐标。

: idx = tsearch (x, y, t, xi, yi)

搜索封闭的德劳内凸包。

对于t = delaunay (x, y)，查找t中包含点(xi, yi)的索引。对于凸包外的点，idx为NaN。

参见:delaunay, delaunayn。

: idx = tsearchn (x, t, xi)

: [idx, p] = tsearchn (x, t, xi)

找出包含给定点的单纯形。

tsearchn通常与delaunayn一起使用:t = delaunayn (x)返回一组简单体t，然后Tsearchn返回包含xi的每个点的t的行索引。对于凸包外的点，idx为NaN。

如果请求，tsearchn还返回封闭的简单体的质心坐标p。

参见:delaunay, delaunayn, research。

一个使用搜索的例子可以从简单的三角测量中看到

x = [-1; -1; 1; 1];

y = [-1; 1; -1; 1];

tri = [1, 2, 3; 2, 3, 4];

由三角形定义的两个三角形组成的。然后我们就可以确定一个点落在哪个三角形里

tsearch (x, y, tri, -0.5, -0.5)

⇒ 1

tsearch (x, y, tri, 0.5, 0.5)

⇒ 2

我们可以确定一个点不在像这样的三角形内

tsearch (x, y, tri, 2, 2)

⇒ NaN

搜索和搜索在镶嵌图中找到离期望点最近的点。所需的点不一定要在镶嵌中，即使是这样，镶嵌的返回点也不一定是n -单纯形中找到所需点的顶点之一。

: idx = dsearch (x, y, tri, xi, yi)

: idx = dsearch (x, y, tri, xi, yi, s)

返回x, y中最近点的索引idx到元素[xi(:)， yi(:)]。

为了兼容性，接受变量s，但忽略它。

参见:搜索，搜索。

: idx = dsearchn (x, tri, xi)

: idx = dsearchn (x, tri, xi, outval)

: idx = dsearchn (x, xi)

: [idx, d] = dsearchn (…)

返回x中最近点到元素xi的索引idx。

如果提供了outval，则不包含在简单函数tri中的xi的值将被设置为outval。通常，tri由delaunayn (x)返回。

可选输出d包含查询点xi和最近的单纯形点x之间距离的列向量。

参见:研究，搜索。

使用研究的一个例子，使用上面的x, y和tri的值是

dsearch (x, y, tri, -2, -2)

⇒ 1

如果您希望标记镶嵌之外的点，那么搜索可以用作

dsearchn ([x, y], tri, [-2, -2], NaN)

⇒ NaN

dsearchn ([x, y], tri, [-0.5, -0.5], NaN)

⇒ 1

然后用NaN标记镶嵌外的点。

**30.2 Voronoi图**

泰森多边形法图或泰森多边形法镶嵌的一个点集s n维空间,是n维空间,这样所有的镶嵌在v (p),镶嵌的分区p是一个成员的年代,更接近比任何其他点p s。泰森多边形法图与德劳内点集的三角剖分,泰森多边形法,顶点的镶嵌的中心是circum-circles单形的德劳内镶嵌。

: voronoi (x, y)

: voronoi (x, y, options)

: voronoi (…, "linespec")

: voronoi (hax, …)

: h = voronoi (…)

: [vx, vy] = voronoi (…)

绘制点(x, y)的Voronoi图。

没有绘制点在无穷远处的Voronoi面。

options参数必须是字符串或由字符串组成的单元格数组，它包含传递给底层qhull命令的选项。有关详细信息，请参阅Qhull库的文档http://www.qhull.org/html/qh-quick.htm#options。

如果指定了“linespec”，则用于设置绘图的颜色和线条样式。

如果提供了轴图形句柄hax，则Voronoi图将在指定的轴上绘制，而不是在新图形中绘制。

如果请求单个输出参数，则将绘制Voronoi图，并返回该图的图形句柄h。

[vx, vy] = voronoi(…)返回voronoi顶点，而不是绘制图表。

x = rand (10, 1);

y = rand (size (x));

h = convhull (x, y);

[vx, vy] = voronoi (x, y);

plot (vx, vy, "-b", x, y, "o", x(h), y(h), "-g");

legend ("", "points", "hull");

参见:voronoin, delaunay, convhull。

: [C, F] = voronoin (pts)

: [C, F] = voronoin (pts, options)

计算n维Voronoi面。

大小为[n, dim]的输入矩阵pts在维数为dim的空间中包含n个点。

C包含Voronoi切面的点。对于每个面，列表F包含了Voronoi点的索引。

第二个可选参数必须是字符串或由字符串组成的单元格数组，它包含传递给底层qhull命令的选项。有关详细信息，请参阅Qhull库的文档http://www.qhull.org/html/qh-quick.htm#options。

默认选项取决于输入的尺寸:

2-D and 3-D: options = {"Qbb"}

4-D and higher: options = {"Qbb", "Qx"}

如果没有options或[]，则使用默认参数。否则，options将替换默认参数列表。要将用户选项附加到默认值，必须重复options中的默认参数。使用空字符串不传递任何参数。

参见:voronoi, convhulln, delaunayn。

使用voronoi的一个例子是

rand ("state",9);

x = rand (10,1);

y = rand (10,1);

tri = delaunay (x, y);

[vx, vy] = voronoi (x, y, tri);

triplot (tri, x, y, "b");

hold on;

plot (vx, vy, "r");

结果如图30.3所示。请注意，其中一个三角形的圆周已添加到此图中，以使Delaunay镶嵌和Voronoi图之间的关系更清晰。

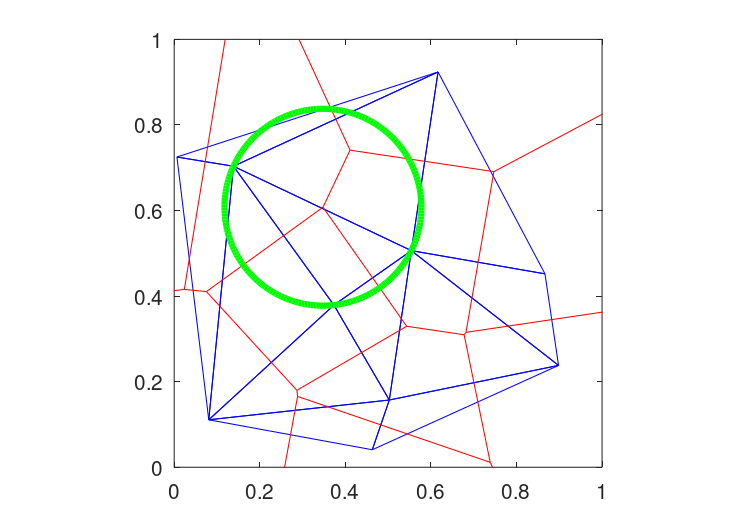


图30.3:随机点集的Delaunay三角剖分(蓝线)和Voronoi图(红线)

关于Voronoi图的面大小的附加信息，以及一组点中的哪些点在一个多边形中，可以分别使用polyarea和inpolygon函数来获得。

: a = polyarea (x, y)

: a = polyarea (x, y, dim)

用三角形法确定多边形的面积。

变量x和y定义了顶点对，因此必须具有相同的形状。它们可以是向量，也可以是数组。如果它们是数组，则分别处理x和y列，并为每个列返回一个区域。

如果给出了可选的dim参数，那么polyarea将沿着数组x和y的这个维度工作。

使用多区域的一个例子可能是

rand ("state", 2);

x = rand (10, 1);

y = rand (10, 1);

[c, f] = voronoin ([x, y]);

af = zeros (size (f));

for i = 1 : length (f)

af(i) = polyarea (c (f {i, :}, 1), c (f {i, :}, 2));

endfor

顶点在无穷远处的Voronoi图的面有无限的面积。矩形多边形的简化版本可以使用rectint

: area = rectint (a, b)

计算矩形或N-D框相交的面积或体积。

计算a中的矩形与b中的矩形的交点面积。支持n维盒，此时根据维数计算体积或超体积。

二维矩形定义为[xpos ypos width height]，其中xpos和ypos是左下角的位置。支持更高的维度，其中每个维度的最小值的坐标遵循该维度中的框的长度，例如，[xpos ypos zpos kpos…width height depth k\_length…]。

a和b的每一行都定义了一个矩形，如果两者都定义了多个矩形，那么输出面积就是一个矩阵，其中第i行对应于a的第i行，第j列对应于b的第j行。

参见polyarea。

: in = inpolygon (x, y, xv, yv)

: [in, on] = inpolygon (x, y, xv, yv)

对于由顶点点(xv, yv)定义的多边形，如果点(x, y)在多边形内部(或在边界上)则返回true;否则，返回false。

输入变量x和y，必须有相同的维数。

如果点恰好在多边形边缘上，则可选输出on返回true，否则返回false。

参见:delaunay。

使用多边形的一个例子可能是

randn ("state", 2);

x = randn (100, 1);

y = randn (100, 1);

vx = cos (pi \* [-1 : 0.1: 1]);

vy = sin (pi \* [-1 : 0.1 : 1]);

in = inpolygon (x, y, vx, vy);

plot (vx, vy, x(in), y(in), "r+", x(!in), y(!in), "bo");

axis ([-2, 2, -2, 2]);

结果如图30.4所示。

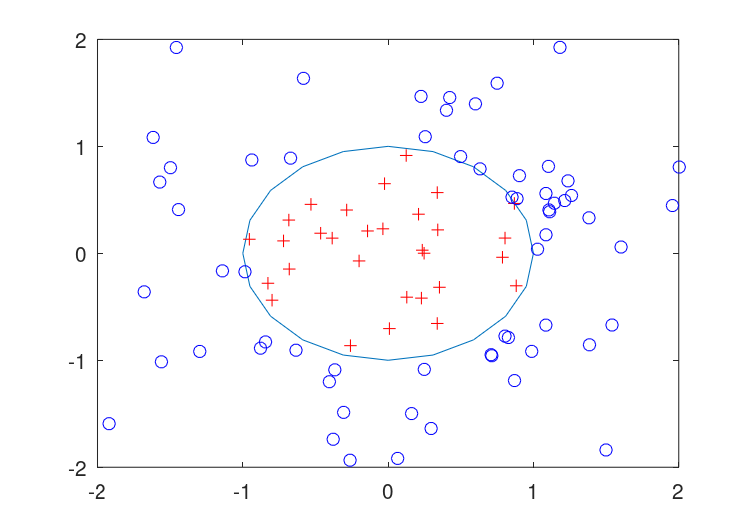


图30.4:inpolygon函数的演示，用于确定多边形内的点

**30.3凸壳**

一组点的凸包是包含所有点的最小凸包络。Octave具有convhull和convhulln函数，用于计算二维和n维点集的凸包。

: H = convhull (x, y)

: H = convhull (x, y, z)

: H = convhull (x)

: H = convhull (…, options)

: [H, V] = convhull (…)

计算一个2-D或3-D点集合的凸包。

船体H是指向原始点集合的线性索引向量，指定哪些点形成封闭船体。仅对于二维输入，输出在船体周围以逆时针的方式排序。

输入x也可以是一个有两列或三列的矩阵，其中第一列包含x数据，第二列包含y数据，可选的第三列包含z数据。

一个可选的最后参数必须是字符串或字符串的单元格数组，它包含传递给底层qhull命令的选项。有关详细信息，请参阅Qhull库的文档http://www.qhull.org/html/qh-quick.htm#options。默认选项为{"Qt"}。

如果没有options或[]，则使用默认参数。否则，options将替换默认参数列表。要将用户选项附加到默认值，必须重复options中的默认参数。使用空字符串不传递任何参数。

如果请求第二个输出V，则计算封闭凸包的体积。

参见:convhulln, delaunay, voronoi。

: h = convhulln (pts)

: h = convhulln (pts, options)

: [h, v] = convhulln (…)

计算点集合的凸包。

PTS是一个大小为[n, dim]的矩阵，在维数为dim的空间中包含n个点。

船体h是指向点集合的索引向量，并指定哪些点构成封闭船体。

第二个可选参数必须是字符串或由字符串组成的单元格数组，它包含传递给底层qhull命令的选项。有关详细信息，请参阅Qhull库的文档http://www.qhull.org/html/qh-quick.htm#options。默认选项取决于输入的尺寸:

2D, 3D, 4D: options = {"Qt"}

5D and higher: options = {"Qt", "Qx"}

如果没有options或[]，则使用默认参数。否则，options将替换默认参数列表。要将用户选项附加到默认值，必须重复options中的默认参数。使用空字符串不传递任何参数。

如果请求第二个输出v，则计算封闭凸包的体积。

参见:convhull, delaunayn, voronoin。

使用convhull的一个例子是

x = -3:0.05:3;

y = abs (sin (x));

k = convhull (x, y);

plot (x(k), y(k), "r-", x, y, "b+");

axis ([-3.05, 3.05, -0.05, 1.05]);

上面的输出如图30.5所示。

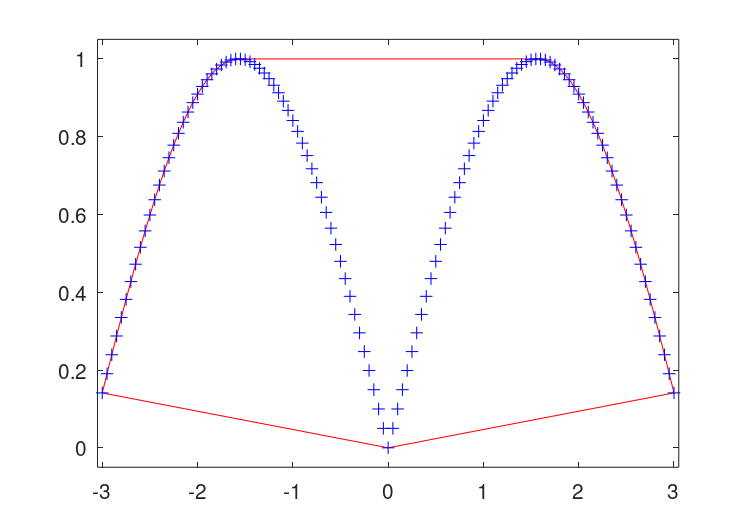


图30.5:一组简单点的凸包

**30.4离散数据插值**

Delaunay镶嵌的一个重要用途是，它可以用来从分散的数据插值到任意一组点。要做到这一点，用delaunay或delaunayn计算已知点集的n -单纯形。然后确定找到所需点的简单体。最后，使用简单点的顶点插值到所需的点。执行这种插值的函数是griddata, griddata3和griddatan。

: zi = griddata (x, y, z, xi, yi)

: zi = griddata (x, y, z, xi, yi, method)

: [xi, yi, zi] = griddata (…)

: vi = griddata (x, y, z, v, xi, yi, zi)

: vi = griddata (x, y, z, v, xi, yi, zi, method)

: vi = griddata (x, y, z, v, xi, yi, zi, method, options)

在指定点插值不规则的二维和三维源数据。

对于二维插值，输入x和y定义函数z = f (x, y)求值的点。输入x, y, z要么是相同长度的向量，要么是不相等的向量x, y被扩展到具有网格的二维网格，z是与x - y网格的结果大小匹配的二维矩阵。

插值点为(xi, yi)。当且仅当xi是行向量，yi是列向量时，mesh - grid将用于创建插值点的网格。

对于3-D插值，输入x, y和z定义函数v = f (x, y, z)求值的点。输入x, y, z是相同长度的向量，如果它们长度不等，则将它们扩展为具有网格的三维网格。输入v的大小必须匹配原始数据的大小，无论是作为一个向量还是一个矩阵。

可选的输入插值方法可以是“最接近”，“线性”或二维数据“v4”。当该方法“最接近”时，输出vi将是原始数据(x, y, z)中最接近查询点(xi, yi, zi)的点。当该方法为“线性”时，输出vi将是原始源数据在每个维度中最近的两个点之间的线性插值。仅在二维情况下，“v4”方法也可用，它实现了双调和样条插值。如果method省略或为空，则默认为“linear”。

对于3-D插值，可选参数options在计算用于插值的Delaunay三角剖分时直接传递给Qhull。有关默认值以及如何传递不同值的更多信息，请参阅delaunayn。

编程注意事项:如果输入是复数，则实部和虚部分别内插。插值通常基于Delaunay三角剖分。输入点凸包之外的任何查询值都将返回NaN。然而，“v4”方法不使用三角测量，并将返回原始数据之外的值(外推)。

参见:griddata3, griddatan, delaunay。

: vi = griddata3 (x, y, z, v, xi, yi, zi)

: vi = griddata3 (x, y, z, v, xi, yi, zi, method)

: vi = griddata3 (x, y, z, v, xi, yi, zi, method, options)

在指定点插值不规则的三维源数据。

输入x, y和z定义了函数v = f (x, y, z)求值的点。输入x, y, z是相同长度的向量，如果它们长度不等，则将它们扩展为具有网格的三维网格。输入v的大小必须匹配原始数据的大小，无论是作为一个向量还是一个矩阵。

插值点由xi, yi, zi指定。

可选的输入插值方法可以是“最接近”或“线性”。当该方法“最接近”时，输出vi将是原始数据(x, y, z)中最接近查询点(xi, yi, zi)的点。当该方法为“线性”时，输出vi将是原始源数据在每个维度中最近的两个点之间的线性插值。如果method省略或为空，则默认为“linear”。

可选参数options在计算用于插值的Delaunay三角剖分时直接传递给Qhull。有关默认值以及如何传递不同值的信息，请参阅delaunayn。

编程注意事项:如果输入是复数，则实部和虚部分别内插。插值基于Delaunay三角剖分，输入点的凸包之外的任何查询值都将返回NaN。

参见:griddata, griddatan, delaunayn。

: yi = griddatan (x, y, xi)

: yi = griddatan (x, y, xi, method)

: yi = griddatan (x, y, xi, method, options)

在xi指定的点插入不规则源数据x, y。

输入x是一个MxN矩阵，表示n维空间中的M个点。输入y是一个单值列向量(Mx1)，表示在点x处求值的函数，即y = fcn (x)。输入xi是一个点列表，函数输出yi应通过插值近似。xi必须具有与x相同的列数(N)，以便维度匹配。

可选的输入插值方法可以是“最接近”或“线性”。当该方法“最接近”时，输出yi将是原始数据x中最接近查询点xi的点。当该方法为“线性”时，输出yi将是原始源数据中最近的两个点之间的线性插值。如果method省略或为空，则默认为“linear”。

可选参数options在计算用于插值的Delaunay三角剖分时直接传递给Qhull。有关默认值以及如何传递不同值的信息，请参阅delaunayn。

例子

## Evaluate sombrero() function at irregular data points

x = 16\*gallery ("uniformdata", [200,1], 1) - 8;

y = 16\*gallery ("uniformdata", [200,1], 11) - 8;

z = sin (sqrt (x.^2 + y.^2)) ./ sqrt (x.^2 + y.^2);

## Create a regular grid and interpolate data

[xi, yi] = ndgrid (linspace (-8, 8, 50));

zi = griddatan ([x, y], z, [xi(:), yi(:)]);

zi = reshape (zi, size (xi));

## Plot results

clf ();

plot3 (x, y, z, "or");

hold on

surf (xi, yi, zi);

legend ("Original Data", "Interpolated Data");

编程注意事项:如果输入是复数，则实部和虚部分别内插。插值基于Delaunay三角剖分，输入点的凸包之外的任何查询值都将返回NaN。对于二维和三维数据，可以使用griddata函数进行插值。

参见:griddata, griddata3, delaunayn。

使用griddata函数的一个示例是

rand ("state", 1);

x = 2\*rand (1000,1) - 1;

y = 2\*rand (size (x)) - 1;

z = sin (2\*(x.^2+y.^2));

[xx,yy] = meshgrid (linspace (-1,1,32));

zz = griddata (x, y, z, xx, yy);

mesh (xx, yy, zz);

它从随机分散的点插入到一个均匀的网格。上面的输出如图30.6所示。

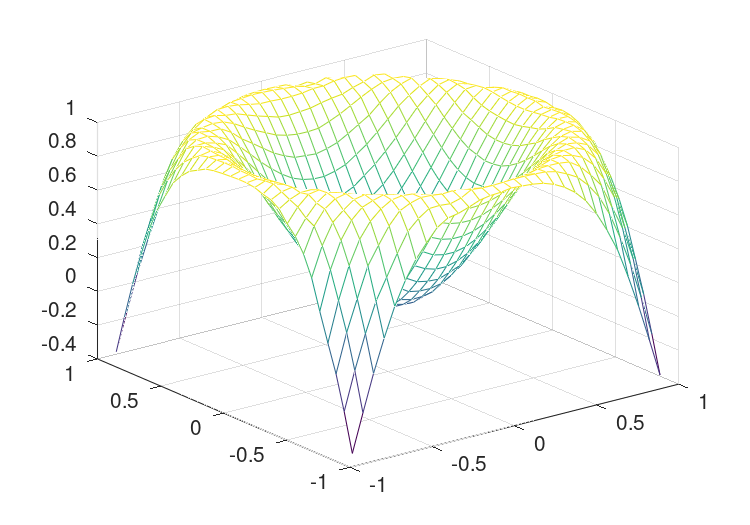


图30.6:从分散的数据到规则网格的插值

**30.5向量旋转矩阵**

Octave的几何函数中还包括在三维空间中支持矢量旋转的基本函数。每个主轴x、y、z的旋转都提供了单独的函数。根据欧拉旋转定理，任意向量p的任意旋转R可以表示为三个主要旋转的乘积:

p' = Rp = Rz\*Ry\*Rx\*p

: T = rotx (angle)

Rotx返回3x3变换矩阵，该变换矩阵对应于向量围绕x轴进行的指定角度(以度为单位)的有效旋转，其中一个正角度对应于从x正侧观察y-z平面时的逆时针旋转。

变换矩阵的形式为:

| 1 0 0 |

T = | 0 cos(angle) -sin(angle) |

| 0 sin(angle) cos(angle) |

当作用于列向量时，这个旋转矩阵打算用作左乘矩阵，使用符号v = T\*u。例如，一个矢量u，指向正y轴，绕x轴旋转90度，将得到一个指向正z轴的矢量:

>> u = [0 1 0]'

u =

0

1

0

>> T = rotx (90)

T =

1.00000 0.00000 0.00000

0.00000 0.00000 -1.00000

0.00000 1.00000 0.00000

>> v = T\*u

v =

0.00000

0.00000

1.00000

参见:roty, roz。

: T = roty (angle)

Roty返回3x3变换矩阵，该变换矩阵对应于向量围绕y轴进行的指定角度(以度为单位)的有效旋转，其中正角度对应于从y正侧观察z-x平面时的逆时针旋转。

变换矩阵的形式为:

| cos(angle) 0 sin(angle) |

T = | 0 1 0 |

| -sin(angle) 0 cos(angle) |

当作用于列向量时，这个旋转矩阵打算用作左乘矩阵，使用符号v = T\*u。例如，一个矢量u，指向正z轴，绕y轴旋转90度，将得到一个指向正x轴的矢量:

>> u = [0 0 1]'

u =

0

0

1

>> T = roty (90)

T =

0.00000 0.00000 1.00000

0.00000 1.00000 0.00000

-1.00000 0.00000 0.00000

>> v = T\*u

v =

1.00000

0.00000

0.00000

参见:rotx, rotz。

: T = rotz (angle)

Rotz返回3x3变换矩阵，该变换矩阵对应于一个矢量围绕z轴以指定角度(以度为单位)进行的有效旋转，其中一个正角度对应于从z正侧观察x-y平面时的逆时针旋转。

变换矩阵的形式为:

| cos(angle) -sin(angle) 0 |

T = | sin(angle) cos(angle) 0 |

| 0 0 1 |

当作用于列向量时，这个旋转矩阵打算用作左乘矩阵，使用符号v = T\*u。例如，一个矢量u，指向正x轴，绕z轴旋转90度，将得到一个指向正y轴的矢量:

>> u = [1 0 0]'

u =

1

0

0

>> T = rotz (90)

T =

0.00000 -1.00000 0.00000

1.00000 0.00000 0.00000

0.00000 0.00000 1.00000

>> v = T\*u

v =

0.00000

1.00000

0.00000

参见:rox, roty。

**31信号处理**

本章描述了Octave中可用的信号处理和快速傅立叶变换函数。快速傅里叶变换计算与FFTW或FFTPACK库取决于如何构建Octave。

: y = fft (x)

: y = fft (x, n)

: y = fft (x, n, dim)

使用快速傅里叶变换(FFT)算法计算x的离散傅里叶变换。

FFT沿着数组的第一个非单维计算。因此，如果x是一个矩阵，fft (x)计算x的每一列的fft。

如果带两个参数调用，则n应该是一个整数，指定要使用的x的元素数量，或者是一个空矩阵，指定应该忽略它的值。如果n大于计算FFT的维度，则调整x的大小并用零填充。否则，如果n小于计算FFT的维度，则截断x。

如果带三个参数调用dim，则dim是一个整数，指定执行FFT的矩阵的维数。

参见:ifft, fft2, fftn, fftw。

: x = ifft (y)

: x = ifft (y, n)

: x = ifft (y, n, dim)

使用快速傅里叶变换(FFT)算法计算y的离散傅里叶反变换。

逆FFT沿阵列的第一个非单维计算。因此，如果y是一个矩阵，ifft (y)计算y的每一列的逆FFT。

如果带两个参数调用，则n应该是一个整数，指定要使用y的元素数量，或者是一个空矩阵，指定应该忽略它的值。如果n大于计算逆FFT所沿的维度，则y将调整大小并填充零。否则，如果n小于计算逆FFT的维数，则y被截断。

如果带三个参数调用dim，则dim是一个整数，指定执行逆FFT的矩阵的维数。

参见:fft, ifft2, ifftn, fftw。

: B = fft2 (A)

: B = fft2 (A, m, n)

使用快速傅里叶变换(FFT)算法计算A的二维离散傅里叶变换。

可选参数m和n可以用来指定要使用的A的行数和列数。如果它们中的任何一个大于A的大小，则会调整A的大小并填充0。

如果A是一个多维矩阵，则分别处理A的每个二维子矩阵。

参见:ifft2, fft, fftn, fftw。

: A = ifft2 (B)

: A = ifft2 (B, m, n)

使用快速傅里叶变换(FFT)算法计算B的二维离散傅里叶逆变换。

可选参数m和n可以用来指定B要使用的行数和列数。如果它们中的任何一个大于B的大小，则会调整B的大小并填充0。

如果B是一个多维矩阵，则B的每个二维子矩阵分别处理。

参见:fft2, ifft, ifftn, fftw。

: B = fftn (A)

: B = fftn (A, size)

使用快速傅里叶变换(FFT)算法计算A的n维离散傅里叶变换。

可选的vector实参size可用于指定要使用的数组的维度。如果size的元素小于A的相应维度，则在执行FFT之前截断A的维度。否则，如果一个元素的size大于相应的维度，则调整A的大小并填充0。

参见:fftn, fft, fft2, fftw。

: A = ifftn (B)

: A = ifftn (B, size)

使用快速傅里叶变换(FFT)算法计算B的反n维离散傅里叶变换。

可选的vector实参size可用于指定要使用的数组的维度。如果size的元素小于B的相应维度，则在执行逆FFT之前截断B的维度。否则，如果一个元素的size大于相应的维度，则B被调整大小并填充0。

参见:fftn, ifft, ifft2, fftw。

Octave使用FFTW库来执行FFT计算。当Octave启动并初始化FFTW库时，它们会读取一个系统范围的文件(在Unix系统上，通常是/etc/fftw/wisdom)，其中包含有助于加快FFT计算的信息。这种信息被称为智慧。系统范围的文件允许智慧在使用FFTW库的所有应用程序之间共享。

使用fftw函数生成和保存智慧。使用与FFTW库一起提供的实用程序(在Unix系统上是FFTW -wisdom)，您甚至可以将Octave生成的智能添加到系统范围的智能文件中。

: method = fftw ("planner")

: fftw ("planner", method)

: wisdom = fftw ("dwisdom")

: fftw ("dwisdom", wisdom)

: nthreads = fftw ("threads")

: fftw ("threads", nthreads)

管理FFTW智慧数据。

智慧数据可用于显著加快fft的计算，但意味着其计算的初始成本。当FFTW库初始化时，它们读取一个系统范围的智能文件(通常在/etc/fftw/wisdom中)，允许智能在Octave以外的应用程序之间共享。或者，fftw函数可以用来导入智慧。例如,

wisdom = fftw ("dwisdom")

将现有的智慧保存到八度使用的字符串智慧。然后可以将该字符串保存到文件中，并分别使用save和load命令进行恢复。现有的智慧可以按照以下方式重新导入

fftw ("dwisdom", wisdom)

如果智慧是一个空字符串，那么所使用的智慧将被清除。

在计算傅里叶变换的过程中产生了更多的智慧。这种智慧产生的方式也由fftw函数控制。对待智慧有五种不同的方式:

"estimate"

指定不执行计算某个特定项的最佳方法的运行时度量，并使用简单的启发式方法来选择(可能是次优的)计划。这种方法的优点是在生成平面时很少或没有开销，这适用于只计算一次的傅里叶变换。

"measure"

在这种情况下，将考虑一系列执行转换的算法，并根据其执行时间选择最佳算法。

"patient"

与“measure”类似，但考虑了更广泛的算法。

"exhaustive"

类似于“度量”，但所有可能用于处理变换的算法都被考虑在内。

"hybrid"

由于算法的运行时度量可能非常昂贵，因此这是一种折衷方法，其中“度量”用于转换到8192的大小，而超过该大小则使用“估计”方法。

默认的方法是“estimate”。查询当前方法

method = fftw ("planner")

或者通过使用

fftw ("planner", method)

请注意，重新启动Octave时，计算智慧将丢失。但是，如果将智能数据保存到上述文件中，则可以重新加载它。保存的智慧文件不应该在不同的平台上使用，因为它们效率不高，并且失去了计算智慧的意义。

用于计算计划和执行转换的线程数量可以使用

fftw ("threads", NTHREADS)

注意，Octave必须使用多线程的FFTW支持来编译这个特性。默认情况下，使用当前进程可用的(逻辑)处理器数或3个(以较小者为准)。

参见:fft, fft, fft2, ifft2, fftn, ifftn。

: c = fftconv (x, y)

: c = fftconv (x, y, n)

使用FFT卷积两个向量进行计算。

C = fftconv (x, y)返回长度等于length (x) + length (y) - 1的向量。如果x和y是两个多项式的系数向量，则返回值是乘积多项式的系数向量。

计算通过调用函数fftfilt来使用FFT。如果指定了可选参数n，则使用n点FFT。

参见:deconv, conv, conv2。

: y = fftfilt (b, x)

: y = fftfilt (b, x, n)

使用FFT用FIR滤波器b滤波x。

如果x是一个矩阵，过滤矩阵的每一列。

给定可选的第三个参数n, fftfilt使用重叠添加方法使用n点FFT过滤x和b。FFT的大小必须是2的偶数次幂，并且必须大于或等于b的长度。如果指定的n不满足这些条件，则自动调整为最接近的值。

参见:filter, filter2。

: y = filter (b, a, x)

: [y, sf] = filter (b, a, x, si)

: [y, sf] = filter (b, a, x, [], dim)

: [y, sf] = filter (b, a, x, si, dim)

对数据x应用1-D数字滤波器。

Filter返回以下线性定常差分方程的解:

N M

SUM a(k+1) y(n-k) = SUM b(k+1) x(n-k) for 1<=n<=length(x)

k=0 k=0

其中N=length(a)-1, M=length(b)-1。结果在x的第一个非单元素维度上计算，如果提供，则在dim上计算。

方程的等价形式为:

N M

y(n) = - SUM c(k+1) y(n-k) + SUM d(k+1) x(n-k) for 1<=n<=length(x)

k=1 k=0

式中c = a/a(1)， d = b/a(1)。

如果提供了第四个参数si，则将其作为系统的初始状态，并将最终状态返回为sf。状态向量是一个列向量，它的长度等于最长系数向量的长度减去1。如果未提供si，则初始状态向量被设置为全零。

对于Z变换，y是将离散时间信号x传递给下述有理系统函数的结果:

M

SUM d(k+1) z^(-k)

k=0

H(z) = ---------------------

N

1 + SUM c(k+1) z^(-k)

k=1

参见:filter2, fftfilt, frequency。

: y = filter2 (b, x)

: y = filter2 (b, x, shape)

将2-D FIR滤波器b应用于x。

如果指定了参数shape，则返回所需形状的数组。可能的值有:

"full"

pad x，在过滤之前，四周都是零。

"same"

无填充x(默认值)

"valid"

过滤后修剪x，使边缘效果不包括在内。

注意，这只是卷积的一个变化，参数颠倒，b旋转180度。

参见:conv2。

: [h, w] = freqz (b, a, n, "whole")

: [h, w] = freqz (b)

: [h, w] = freqz (b, a)

: [h, w] = freqz (b, a, n)

: h = freqz (b, a, w)

: [h, w] = freqz (…, Fs)

: freqz (…)

返回分子和分母系数分别为b和a的有理IIR滤波器的复频率响应h。

响应在0到2\*pi之间的n个角频率上进行评估。

输出值w是频率的矢量。

如果省略a，则假定分母为1(这对应于一个简单的FIR滤波器)。

如果省略n，则假设值为512。为了最快的计算，n应该分解成少量的小质数。

如果省略第四个参数“whole”，则响应在0和pi之间的频率上进行评估。

freqz (b, a, w)

计算在矢量w中特定频率处的响应。w的值以弧度为单位测量。

[…] = freqz (…, Fs)

假设采样率为f，返回频率以Hz而不是弧度为单位。如果您正在评估特定频率w的响应，则应该以Hz而不是弧度为单位请求这些频率。

freqz (…)

绘制h的幅值和相位响应，而不是返回它们。

参见:freqz\_plot。

: freqz\_plot (w, h)

: freqz\_plot (w, h, freq\_norm)

绘制h的幅值和相位响应。

如果可选的freq\_norm参数为真，则频率向量w以标准化弧度为单位。如果freq\_norm为假，或者没有给出，则w以赫兹为单位测量。

参见:频率。

: y = sinc (x)

计算sinc函数。

返回sin (\*x) / (\*x)。

: b = unwrap (x)

: b = unwrap (x, tol)

: b = unwrap (x, tol, dim)

通过适当地添加或减去2\*pi的倍数来展开弧度相位，以消除大于tol的跳跃。

Tol默认为pi。

unwrap将沿着维度dim工作。如果未指定dim，则默认为第一个非单例维度。

unwrap忽略所有非有限输入值(Inf, NaN, NA)。

: [a, b] = arch\_fit (y, x, p, iter, gamma, a0, b0)

使用Engle原ARCH论文中的评分算法对时间序列y拟合ARCH回归模型。

模型是

y(t) = b(1) \* x(t,1) + … + b(k) \* x(t,k) + e(t),

h(t) = a(1) + a(2) \* e(t-1)^2 + … + a(p+1) \* e(t-p)^2

其中，e(t)为N(0, h(t))，给定一个时间序列向量y到时间t-1和一个(普通)回归量矩阵x到t。残差方差的回归顺序由p指定。

如果用正整数k调用arch\_fit (y, k, p)，则拟合一个ARCH(k, p)过程，即对x的第t行执行上述操作

[1, y(t-1), …, y(t-k)]

还可以指定迭代次数iter、更新因子gamma以及评分算法的初始值a0和b0。

: y = arch\_rnd (a, b, t)

模拟一个长度为t, AR系数为b, CH系数为a的ARCH序列。

结果y(t)符合模型

y(t) = b(1) + b(2) \* y(t-1) + … + b(lb) \* y(t-lb+1) + e(t),

e(t)在y到t-1的情况下，等于N(0, h(t)

h(t) = a(1) + a(2) \* e(t-1)^2 + … + a(la) \* e(t-la+1)^2

: [pval, lm] = arch\_test (y, x, p)

对于线性回归模型

y = x \* b + e

针对CH(p)的选择，对无条件异方差的零假设进行拉格朗日乘数(LM)检验。

即，模型为

y(t) = b(1) \* x(t,1) + … + b(k) \* x(t,k) + e(t),

已知y = t-1 x = t, e(t) = N(0, h(t)

h(t) = v + a(1) \* e(t-1)^2 + … + a(p) \* e(t-p)^2,

空值是a(1) ==…== a(p) == 0。

如果第二个参数是标量整数k，则在k阶的线性自回归模型中执行相同的检验，即与

[1, y(t-1), …, y(t-k)]

作为x的第t行。

在零点下，LM近似具有p个自由度的齐方分布，pval是检验的p值(1减去该分布在LM处的CDF)。

如果没有给出输出参数，则显示p值。

: x = arma\_rnd (a, b, v, t, n)

返回ARMA模型的模拟。

定义了ARMA模型

x(n) = a(1) \* x(n-1) + … + a(k) \* x(n-k)

+ e(n) + b(1) \* e(n-1) + … + b(l) \* e(n-l)

其中k是向量a的长度，l是向量b的长度，e是方差为v的高斯白噪声。函数返回一个长度为t的向量。

可选参数n给出用于初始化的虚拟x(i)的数量，即生成长度为t+n的序列并返回x(n+1:t+n)。如果省略n，则使用n = 100。

: x = autoreg\_matrix (y, k)

给定一个时间序列(向量)y，返回一个矩阵，第一列为1，其他列为y的前k个滞后值。

换句话说，对于t > k， [1, y(t-1)，…，y(t-k)]是结果的第t行。

得到的矩阵可以用作自回归中的回归矩阵。

: c = bartlett (m)

返回长度为m的Bartlett(三角形)窗口的过滤系数。

巴特利特窗口的定义见，例如，A.V.奥本海姆和R. W.谢弗，离散时间信号处理。

: c = blackman (m)

: c = blackman (m, "periodic")

: c = blackman (m, "symmetric")

返回长度为m的Blackman窗口的过滤系数。

如果给出了可选参数"periodic"，则返回窗口的周期形式。这相当于去掉最后一个系数后长度为m+1的窗口。可选参数"symmetric"等同于不指定第二个参数。

对于Blackman窗口的定义，请参见，例如，A.V. Oppenheim & r.w. Schafer，离散时间信号处理。

: y = detrend (x, p)

如果x是一个向量，则dettrend (x, p)从数据x中移除p阶多项式的最佳拟合。

如果x是一个矩阵，dettrend (x, p)对x中的每一列做同样的处理。

第二个参数p是可选的。如果未指定，则假定值为1。这相当于去除线性趋势。

多项式的阶也可以以字符串的形式给出，在这种情况下，p必须是“常数”(对应于p=0)或“线性”(对应于p=1)。

参见polyfit。

: [d, dd] = diffpara (x, a, b)

返回一个积分时间序列的差分参数的估计量d。

使用[2\*pi\*a/t, 2\*pi\*b/ t]中的频率进行估计。如果省略b，则使用区间[2\*pi/T, 2\*pi\*a/T]。如果省略b和a，则使用a = 0.5 \* sqrt (T)和b = 1.5 \* sqrt (T)，其中T为样本量。如果x是矩阵，则估计每列的差分参数。

上述区间内所有频率的估计量以dd的形式返回。

d的值就是dd的均值。

参考文献:P.J. Brockwell和R.A. Davis。时间序列:理论与方法。施普林格1987年。

: [newphi, newv] = durbinlevinson (c, oldphi, oldv)

执行Durbin-Levinson算法的一个步骤。

向量c表示从滞后0到t的自协方差[gamma\_0，…，gamma\_t]， oldphi表示基于c(t-1)的系数，oldv表示相应的误差。

如果省略oldphi和oldv，则执行算法从1到t的所有步骤。

: y = fftshift (x)

: y = fftshift (x, dim)

执行向量x的移位，用于fft和ifft函数，以便将频率0移动到向量或矩阵的中心。

如果x是N个元素的向量，对应以dt为间隔的N个时间样本，则fftshift (fft (x))对应频率

f = [ -(ceil((N-1)/2):-1:1), 0, (1:floor((N-1)/2)) ] \* df

其中df = 1 / (N \* dt)

如果x是一个矩阵，对于行和列也是一样。如果x是一个数组，那么每个维度都是相同的。

可选的dim参数可用于限制排列发生的维度。

参见:ifftshift。

: y = ifftshift (x)

: y = ifftshift (x, dim)

取消fftshift函数的动作。

对于偶数长度x, fftshift是它自己的逆，但奇数长度稍有不同。

参见:fftshift。

: fd = fractdiff (x, d)

计算分数差(1-L)^d x，其中L表示滞后算子，d大于-1。

: c = hamming (m)

: c = hamming (m, "periodic")

: c = hamming (m, "symmetric")

返回长度为m的汉明窗口的过滤系数。

如果给出了可选参数"periodic"，则返回窗口的周期形式。这相当于去掉最后一个系数后长度为m+1的窗口。可选参数"symmetric"等同于不指定第二个参数。

对于汉明窗口的定义，参见，例如，A.V. Oppenheim & r.w. Schafer，离散时间信号处理。

: c = hanning (m)

: c = hanning (m, "periodic")

: c = hanning (m, "symmetric")

返回长度为m的汉宁窗口的过滤系数。

如果给出了可选参数"periodic"，则返回窗口的周期形式。这相当于去掉最后一个系数后长度为m+1的窗口。可选参数"symmetric"等同于不指定第二个参数。

对于汉宁窗口的定义，参见，例如，A.V. Oppenheim & r.w. Schafer，离散时间信号处理。

: H = hurst (x)

通过重新缩放的范围统计量估计样本x的Hurst参数。

如果x是矩阵，则对每列的参数进行估计。

: pp = pchip (x, y)

: yi = pchip (x, y, xi)

返回点x和y的分段三次埃尔米特插值多项式(pchip)。

如果带两个参数调用，则返回分段多项式pp，它可以与ppval一起用于计算特定点上的多项式。

当使用第三个输入参数调用时，pchip在点xi处计算pchip多项式。第三种调用形式相当于ppval (pchip (x, y)， xi)。

变量x必须是长度为n的严格单调向量(递增或递减)。

Y可以是向量也可以是数组。如果y是一个向量，那么它的长度必须与x相同。如果y是一个数组，那么y的大小必须具有[s1, s2，…，sk, n]的形式。该数组在内部被重塑为一个矩阵，其中前导维由s1 \* s2 \*…\* sk给出，然后该矩阵的每一行被单独处理。请注意，这与interp1完全相反，但是为了MATLAB兼容性而做的。

参见:样条，ppval, mkpp, unmkpp。

: [Pxx, w] = periodogram (x)

: [Pxx, w] = periodogram (x, win)

: [Pxx, w] = periodogram (x, win, nfft)

: [Pxx, f] = periodogram (x, win, nfft, Fs)

: [Pxx, f] = periodogram (…, "range")

: periodogram (…)

返回x的周期图(功率谱密度)。

可能的输入有:

x

数据向量。如果x是实值，则估计单侧谱。如果x是复值的，或者“range”指定“two - sided”，则估计全谱。

win

窗口权重数据。如果窗口为空或未指定，则使用默认矩形窗口。否则，在计算周期图之前，将窗口应用于信号(x .\* win)。窗口数据必须是与x长度相同的向量。

nfft

频带数。默认值是256或比x (max(256,2)的长度大2的下一个幂次。^nextpow2(长度(x)))))。如果nfft大于输入的长度，那么x将被补零到nfft的长度。

Fs

采样率。默认为1。

range

光谱范围。“onessided”从[0:nfft/2+1]计算频谱。“双面”从[0:nfft-1]计算频谱。

可选的第二输出w是归一化的角频率。对于单侧计算，如果nfft是偶数，则w在[0,pi]范围内，如果nfft是奇数，则w在[0,pi)范围内。类似地，对于双边计算，w在[0,2 \*pi]或[0,2 \*pi]的范围内，取决于nfft。

如果指定采样频率Fs，则单侧计算的输出频率f将在[0,Fs/2]或[0,Fs/2]范围内。对于双面计算，范围将为[0,Fs]。

在没有输出的情况下调用时，将立即在当前图形窗口中绘制周期图。

参见:fft。

: y = sinetone (freq, rate, sec, ampl)

返回频率为frequency的正弦信号，在采样速率下，其长度为秒秒，幅度为ampl。

参数frequency和ampl可以是相同大小的向量。

默认值是rate = 8000, sec = 1, ampl = 64。

参见:正弦波。

: y = sinewave (m, n, d)

返回一个包含m元素的向量，其中第i个元素由sin (2 \* pi \* (i+d-1) / n)给出。

d的默认值为0,n的默认值为m。

参见:sinetone。

: sde = spectral\_adf (c)

: sde = spectral\_adf (c, win)

: sde = spectral\_adf (c, win, b)

返回给定自协方差向量c，窗口名称win和带宽b的谱密度估计器。

窗口名称，例如“triangle”或“rectangle”用于搜索一个名为win\_lw的函数。

如果省略win，则使用三角形窗口。

如果省略b，则使用1 / sqrt (length (x))。

参见:spectral\_xdf。

: sde = spectral\_xdf (x)

: sde = spectral\_xdf (x, win)

: sde = spectral\_xdf (x, win, b)

返回给定数据向量x、窗口名称win和带宽b的谱密度估计器。

窗口名称，例如“triangle”或“rectangle”用于搜索名为win\_sw的函数。

如果省略win，则使用三角形窗口。

如果省略b，则使用1 / sqrt (length (x))。

参见:spectral\_adf。

: savg = spencer (x)

返回Spencer对x每列的15点移动平均。

: y = stft (x)

: y = stft (x, win\_size)

: y = stft (x, win\_size, inc)

: y = stft (x, win\_size, inc, num\_coef)

: y = stft (x, win\_size, inc, num\_coef, win\_type)

: [y, c] = stft (…)

通过应用win\_size数据点的窗口和inc点的增量，计算具有num\_coef系数的向量x的短时傅里叶变换。

在计算傅里叶变换之前，应用以下窗口之一:

"hanning"

win\_type = 1

"hamming"

win\_type = 2

"rectangle"

win\_type = 3

窗口名可以作为字符串传递，也可以通过win\_type号传递。

以下默认值用于未指定的参数:win\_size = 80, inc = 24, num\_coef = 64，和win\_type = 1。

y = stft (x，…)根据num\_coef正频率返回傅里叶系数的绝对值。

[y, c] = stft (x，…)返回整个stft矩阵y和一个包含窗口大小、增量和窗口类型的3元素向量c，这是合成函数所需要的。

参见:合成。

: x = synthesis (y, c)

从其短时傅里叶变换y和指定窗口大小、增量和窗口类型的3元素向量c计算信号。

y和c的值可以由

[y, c] = STFT (x，…)

参见:stft。

: [a, v] = yulewalker (c)

给定自协方差向量c [gamma\_0，…，gamma\_p]，用Yule-Walker估计拟合AR (p)-模型。

返回AR系数a和白噪声方差v。

**32图像处理**

由于图像基本上是一个矩阵，Octave是一个非常强大的处理和分析图像的环境。为了说明在Octave中进行图像处理是多么容易，下面的示例将加载图像，通过5 × 5平均滤波器对其进行平滑，并计算平滑图像的梯度。

I = imread ("myimage.jpg");

S = conv2 (I, ones (5, 5) / 25, "same");

[Dx, Dy] = gradient (S);

在这个例子中，S包含平滑图像，Dx和Dy包含图像的偏空间导数。

**32.1加载和保存图像**

大多数图像处理任务的第一步是将图像加载到Octave中，这是通过imread函数完成的。imwrite函数是将镜像写入磁盘的对应函数。

总之，大多数图像处理代码将遵循此代码的结构

I = imread ("my\_input\_image.img");

J = process\_my\_image (I);

imwrite (J, "my\_output\_image.img");

: [img, map, alpha] = imread (filename)

: […] = imread (url)

: […] = imread (…, ext)

: […] = imread (…, idx)

: […] = imread (…, param1, value1, …)

从各种文件格式读取图像。

从文件文件名或在线资源url中以矩阵形式读取图像。如果两者都没有给出，但指定了ext，则查找扩展名为ext的文件。

输出的大小和类别取决于图像的格式。彩色图像作为MxNx3矩阵返回。灰度和黑白图像的大小为MxN。多页图像将有一个额外的第四个维度。

图像的位深度决定了输出的类别:“uint8”、“uint16”或“单一”用于灰度和彩色，“逻辑”用于黑白。请注意，索引的图像总是返回colormap的索引，而与map是否是请求的输出无关。要获取实际的RGB图像，请使用ind2rgb。当读取多个索引图像时，从第一个获取map。在某些罕见的情况下，这可能是不正确的，并且可以使用iminfo来获得每个图像的颜色图。

有关表示图像的更多信息，请参阅Octave手册。(参见表示图像)

一些文件格式，如TIFF和GIF，能够在单个文件中存储多个图像。Idx可以是指定要读取的图像索引的标量或矢量。默认情况下，Octave将只读取第一页。

根据文件格式，可以使用参数、值对配置图像读取。支持以下选项:

"Frames" or "Index"

这是指定idx的另一种方法。当以这种方式指定它时，它的值也可以是字符串"all"。

"Info"

这个选项是为了与MATLAB兼容而存在的，但是没有效果。为了在从单个文件读取多个图像时获得最佳性能，请使用“Index”选项。

"PixelRegion"

控制要读取的图像区域。该值必须是一个单元格数组，包含两个数组，每个数组包含3个元素{[rows]， [cols]}。数组中的元素是要读取的起始像素、增量像素和结束像素。如果省略增量值，则默认为1。例如，以下语句都是等价的:

imread (filename, "PixelRegion", {[200 600], [300 700]});

imread (filename, "PixelRegion", {[200 1 600], [300 1 700]});

imread (filename)(200:600, 300:700);

参见:imwrite, iminfo, imformat。

: imwrite (img, filename)

: imwrite (img, filename, ext)

: imwrite (img, map, filename)

: imwrite (…, param1, val1, …)

以各种文件格式写入图像。

图像可以是二值图像、灰度图像、RGB图像或多维图像。img的大小和类别应该与使用imread阅读时所期望的相同:分别为颜色空间和多个页面保留第三和第四个维度。如果它是一个索引图像，还必须指定colormap映射。

如果未提供ext，则使用filename的文件扩展名来确定格式。实际支持的格式取决于在构建Octave期间所做的选项。使用imformats检查对不同图像格式的支持。

根据文件格式的不同，可以使用param, val对配置图像的写入。支持以下选项:

‘Alpha’

Alpha(透明度)通道的图像。这必须是一个具有相同类的矩阵，并且有img的行数和列数。在多页图像的情况下，第四个维度的大小也必须匹配，第三个维度必须是单一的。默认情况下，图像将是完全不透明的。

‘Compression’

压缩使用一个图像。取值为none(默认值)、bzip、fax3、fax4、jpeg、lzw、rle或deflate。注意，并不是所有的压缩类型都适用于默认为Magick库的所有图像格式。

‘DelayTime’

对于接受动画的格式(如GIF)，控制一帧在移动到下一帧之前显示的时间。该值必须是标量(它将应用于img中的所有帧)，或者长度等于img中的帧数的向量。单位为秒，取值范围为0 ~ 65535，默认为0.5。

‘DisposalMethod’

对于接受动画的格式(如GIF)，控制在绘制下一帧之前对一帧发生的操作。它的值可以是以下字符串之一:"doNotSpecify"(默认);“leaveInPlace”;“restoreBG”;和“restorePrevious”，或长度等于img帧数的字符串的单元格数组。

‘LoopCount’

对于接受动画的格式(如GIF)，控制序列重复的次数。值Inf表示无限循环(默认)，值0或1表示序列只播放一次(循环0次)，值2或以上表示循环次数(循环两次意味着播放完整序列3次)。如果在写入文件结束时只有一个映像，则忽略此选项。

‘Quality’

设置压缩的质量。该值应为0到100之间的整数，值越大表示视觉质量越好，压缩率越低。默认为75。

‘WriteMode’

一些文件格式，如TIFF和GIF，能够在单个文件中存储多个图像。该选项指定是否应该将img添加到文件(如果它存在)，或者是否应该为它创建一个新文件(可能覆盖现有文件)。该值应该是字符串“Overwrite”(默认)或“Append”。

尽管有这个选项，但编写多页图像的最有效方法是传递一个4维img给imwrite，当使用imread并将“Index”选项设置为“all”时，可以期望使用相同的矩阵。

参见:imread, iminfo, imformats。

: val = IMAGE\_PATH ()

: old\_val = IMAGE\_PATH (new\_val)

: old\_val = IMAGE\_PATH (new\_val, "local")

查询或设置内部变量，该变量指定要在其中搜索映像文件的目录列表，列表以冒号分隔。

当从带有“local”选项的函数内部调用时，该变量将在函数及其调用的任何子例程的本地更改。退出函数时，恢复原来的变量值。

参见:EXEC\_PATH, OCTAVE\_HOME, OCTAVE\_EXEC\_HOME。

可以获取磁盘上映像文件的信息，而不必实际将其读入Octave。这是使用iminfo函数完成的，该函数提供了对存储在图像文件头中的许多参数的读访问。

: info = imfinfo (filename)

: info = imfinfo (url)

: info = imfinfo (…, ext)

从文件中读取图像信息。

iminfo返回一个包含存储在文件文件名中的图像信息的结构。如果没有文件文件名，并且指定了ext，则它将查找名为filename的文件和扩展名ext，即名为filename.ext的文件。

输出结构info包含以下字段:

‘Filename’

镜像文件的全称。

‘FileModDate’

最后一次修改文件的日期。

‘FileSize’

磁盘上映像的字节数

‘Format’

图像格式(如“jpeg”)。

‘Height’

以像素为单位的图像高度。

‘Width’

以像素为单位的图像宽度。

‘BitDepth’

每个通道每个像素的位数。

‘ColorType’

图像类型。取值为“grayscale”、“indexed”、“truecolor”、“CMYK”或“undefined”。

‘XResolution’

图像的X分辨率。

‘YResolution’

图像的Y分辨率。

‘ResolutionUnit’

图像分辨率单位。取值为“英寸”、“厘米”或“未定义”。

‘DelayTime’

以百分之一秒为单位的时间(0到65535)，在动画序列中显示下一个图像之前必须过期。

‘LoopCount’

动画循环的迭代次数。

‘ByteOrder’

支持它的格式的尾端选项。取值为“little-endian”、“big-endian”或“undefined”。

‘Gamma’

图像的伽马值。由于显示器的不同，在两个不同的工作站上显示的相同彩色图像可能看起来不同。

‘Quality’

JPEG/MIFF/PNG压缩级别。取值范围为[0 - 100]的整数。

‘DisposalMethod’

仅对GIF图像有效，在创建GIF动画时控制如何呈现连续帧(如何处理前一帧)。取值为“doNotSpecify”、“leaveInPlace”、“restoreBG”或“restorePrevious”。对于非gif文件，value是一个空字符串。

‘Chromaticities’

Value是一个1x8矩阵，其中白色、红色、绿色和蓝色点的x、y色度值按顺序排列。

‘Comment’

影像注释。

‘Compression’

压缩类型。取值为none、bzip、fax3、fax4、jpeg、lzw、rle、deflate、lzma、jpeg2000、jbig2、jbig2或undefined。

‘Colormap’

每个图像的颜色图。

‘Orientation’

图像相对于行和列的方向。值是TIFF 6规范中定义的1到8之间的整数，用于MATLAB兼容性。

‘Software’

用于生成图像的相机或图像输入设备的软件或固件的名称和版本。

‘Make’

录音设备的制造商。这是制造DSC、扫描仪、视频数字化仪或其他设备所产生的图像。

‘Model’

在“制造”栏中注明的录音设备的型号或型号。

‘DateTime’

Exif标准定义的映像创建的日期和时间，即文件被更改的日期和时间。

‘ImageDescription’

Exif标准定义的图像标题。

‘Artist’

相机所有者，摄影师或图像创建者的名称。

‘Copyright’

声称对该图像拥有权利的个人或组织的版权声明。

‘DigitalCamera’

一个结构体，包含从Exif标记检索到的信息。

‘GPSInfo’

具有从Exif标记检索到的地理标记信息的结构体。

参见:imread, imwrite, imshow, imformats。

默认情况下，Octave的图像IO函数(imread, imwrite和iminfo)使用GraphicsMagick库进行操作。这意味着支持大量的图像格式，但考虑到科学中大量的图像格式及其通常的封闭性质，不可能有一个能够读取所有图像格式的图书馆。因此，函数imformats保留了一个可配置的可用格式列表、它们的扩展以及映像IO函数应该使用的函数。这允许通过创建针对特定文件格式的函数来扩展Octave的图像IO功能。

虽然可以直接调用额外的函数，但正确配置Octave的imformat允许从文件格式中抽象出一致的代码。

重要的是要注意，文件格式实际上不是由其文件扩展名定义的，GraphicsMagick能够读取和写入比imformats列出的更多的文件格式。这意味着，即使扩展名不正确或缺少扩展名，图像仍然可以正确读取，即使未列出的格式也不一定不受支持。

: imformats ()

: formats = imformats (ext)

: formats = imformats (format)

: formats = imformats ("add", format)

: formats = imformats ("remove", ext)

: formats = imformats ("update", ext, format)

: formats = imformats ("factory")

管理支持的图像格式。

format是一个结构，包含了支持的每种文件格式的信息，或者来自特定格式ext，字段ext显示的值。它包含以下字段:

ext

文件格式的名称。这可能与文件扩展名匹配，但Octave会自动检测文件格式。

description

文件格式的长描述。

isa

一个函数句柄，用于确认文件是否为指定格式。

write

一个函数句柄，用于在文件具有指定格式时写入。

read

一个函数句柄，用于打开指定格式的文件。

info

一个函数句柄，用于获取指定格式的图像信息。

alpha

如果格式支持alpha通道(透明或哑光)，则为逻辑值。

multipage

如果格式支持多页(每个文件多个图像)，则为逻辑值。

可以通过选项“add”，“remove”和“update”来改变Octave管理文件格式的方式，并提供具有所需字段的结构格式。选项“factory”将配置重置为默认值。

Octave包可以通过使用PKG\_ADD和PKG\_DEL命令来扩展Octave的图像读取功能。

参见:iminfo, imread, imwrite。

**32.2显示图像**

图像处理的一个自然部分是图像的可视化。最基本的函数是imshow函数，它显示第一个输入参数中给出的图像。

: imshow (im)

: imshow (im, limits)

: imshow (im, map)

: imshow (rgb, …)

: imshow (filename)

: imshow (…, string\_param1, value1, …)

: h = imshow (…)

显示图像im，其中im可以是二维(灰度图像)或三维(RGB图像)矩阵。

如果limits是2元素向量[low, high]，则使用介于low和high之间的显示范围来显示图像。如果传递空矩阵作为限制，则显示范围计算为图像中最小值和最大值之间的范围。

如果map是有效的颜色映射，则图像将使用提供的颜色映射显示为索引图像。

如果给出的是文件名而不是图像，则将读取并显示该文件。

如果给定，参数string\_param1的值为value1。String\_param1可以是以下任意值:

"displayrange"

Value1是上面描述的显示范围。

"colormap"

Value1是显示索引图像时使用的颜色映射。

"xdata"

如果value1是一个2元素向量，它必须以[xfirst, xlast]的形式包含水平图像限制，其中xfirst和xlast是角像素中心的横坐标。否则value1必须是一个向量，并且只有第一个和最后一个元素将分别用于xfirst和xlast。

"ydata"

如果value1是一个2元素向量，它必须包含形式为[yfirst, ylast]的垂直图像限制，其中yfirst和ylast是角像素中心的纵坐标。否则value1必须是一个向量，并且yfirst和ylast分别只使用第一个和最后一个元素。

可选的返回值h是图像的图形句柄。

参见:image, imagesc, colormap, gray2ind, rgb2ind。

: image (img)

: image (x, y, img)

: image (…, "prop", val, …)

: image ("prop1", val1, …)

: h = image (…)

将矩阵显示为索引彩色图像。

img的元素是当前颜色图的索引。

X和y是可选的2元素向量，[min, max]，它们指定拐角像素中心的坐标。如果将范围指定为[max, min]，则图像将沿着该轴反转。为方便起见，可以将x和y指定为n元素向量，与img中数据的长度相匹配。但是，只有第一个和最后一个元素将用于确定轴的极限。

可以为图像对象指定多个属性/值对，但它们必须成对出现。

可选的返回值h是图像的图形句柄。

实现注意:图像的原点(0,0)位于左上角。对于普通地块，原点位于左下方。Octave通过正常绘制数据来处理这种反转，然后通过将ydir属性设置为“reverse”来反转y轴的方向。每当需要叠加图像和普通情节时，这就会产生影响。推荐的解决方案是先显示图像，然后使用例如flipud (ydata)绘制反转的ydata。

调用形式:图像函数可以以两种形式调用:高级和低级。当使用普通选项调用时，将使用高级表单，它首先调用newplot来准备图形图形和轴。当图像的唯一输入是属性/值对时，使用低级形式创建图像对象的新实例并将其插入当前轴中。

图形属性:在图像属性中记录了完整的属性列表。

参见:imshow, imagesc, colormap。

: imagesc (img)

: imagesc (x, y, img)

: imagesc (…, climits)

: imagesc (…, "prop", val, …)

: imagesc ("prop1", val1, …)

: imagesc (hax, …)

: h = imagesc (…)

将矩阵图像的缩放版本显示为彩色图像。

对颜色图进行缩放，使矩阵的条目占据整个颜色图。如果给出了climits = [lo, hi]，则该范围被设置为当前轴的“clim”。

X和y是可选的2元素向量，[min, max]，它们指定拐角像素中心的坐标。如果将范围指定为[max, min]，则图像将沿着该轴反转。为方便起见，可以将x和y指定为n元素向量，与img中数据的长度相匹配。然而，只有第一个和最后一个元素将被用来确定图像限制。

可选的返回值h是图像的图形句柄。

调用形式:可以以两种形式调用imagesc函数:高级和低级。当使用普通选项调用时，将使用高级表单，它首先调用newplot来准备图形图形和轴。当图像的唯一输入是属性/值对时，使用低级形式创建图像对象的新实例并将其插入当前轴中。完整的属性列表记录在图像属性。

参见:image, imshow，出租车。

**32.3表示图像**

通常Octave支持四种不同类型的图像，灰度图像，RGB图像，二进制图像和索引图像。灰度图像用m × n矩阵表示，其中每个元素对应于像素的强度。RGB图像用m × n × 3数组表示，其中每个3向量对应于每个像素的红、绿、蓝强度。

灰度或RGB图像中像素值的实际含义取决于矩阵的类别。如果矩阵是double类，像素强度在0到1之间，如果是uint8类，像素强度在0到255之间，如果是uint16类，像素强度在0到65535之间。

二值象是逻辑类的m × n矩阵。二值图像中的像素如果为假则为黑色，如果为真则为白色。

索引图像由一个m × n的整数矩阵和一个c × 3的彩色映射组成。每个整数对应于颜色映射中的一个索引，颜色映射中的每一行对应于RGB颜色。颜色映射必须是double类，值在0到1之间。

以下方便函数可用于图像格式之间的转换。

: dimg = im2double (img)

: dimg = im2double (img, "indexed")

将图像转换为双精度。

img到双精度的转换取决于输入图像的类型。支持以下输入类:

‘uint8, uint16, and int16’

从类的值的范围被缩放到区间[0 1]。

‘logical’

True和false值分别被赋值为1和0。

‘single’

值被强制转换为double类型。

‘double’

返回相同的图像。

如果img是一个索引图像，那么第二个参数应该是字符串"indexed"。如果是这样，那么img必须是浮点类，或者是无符号整数类，并且它将被简单地转换为double。如果它是一个整数类，则应用+1的偏移量。

参见:double。

: img = gray2ind (I)

: img = gray2ind (I, n)

: img = gray2ind (BW)

: img = gray2ind (BW, n)

: [img, map] = gray2ind (…)

将灰度或二值强度图像转换为索引图像。

索引图像将由n个不同的强度值组成。如果没有给出n默认为64灰度图像或2二进制黑白图像。

如果n小于等于256，则输出img为uint8类;否则返回类为uint16。

参见:ind2gray, rgb2ind。

: I = ind2gray (x, map)

将颜色索引图像转换为灰度强度图像。

图像x必须是一个索引图像，它将使用colormap映射进行转换。如果map没有包含足够的图像颜色，则在转换为灰度之前，将x中超出范围的像素映射到映射中的最后一个颜色。

输出I与输入x属于同一类，可以是uint8、uint16、single或double。

实现说明:有几种方法可以将颜色转换为灰度强度。该函数使用rgb2gray得到的亮度值I = 0.299\*R + 0.587\*G + 0.114\*B。其他可能性包括来自rgb2hsv的值组件或使用来自ind2rgb的单个颜色通道。

参见:gray2ind, ind2rgb。

: [x, map] = rgb2ind (rgb)

: [x, map] = rgb2ind (R, G, B)

将红绿蓝(RGB)色彩空间中的图像转换为索引图像。

输入图像rgb可以指定为大小为MxNx3的单个矩阵，也可以指定为三个独立的变量R、G和B，以及它的三个颜色通道红、绿和蓝。

它输出一个索引图像x和一个colormap映射，以解释与输入完全相同的图像。没有抖动或其他形式的颜色量化被执行。索引图像x的输出类可以是uint8, uint16或double，按顺序指定图像中唯一颜色的数量(这将等于map中的行数)。

还支持多维索引图像(大小为MxNx3xK)，既可以通过单个输入(rgb)，也可以通过三个颜色通道作为单独的变量。

参见:ind2rgb, rgb2hsv, rgb2gray。

: rgb = ind2rgb (x, map)

: [R, G, B] = ind2rgb (x, map)

将索引图像转换为红色、绿色和蓝色组件。

图像x必须是一个索引图像，它将使用colormap映射进行转换。如果map没有包含足够的图像颜色，则x中超出范围的像素将映射到映射中的最后一个颜色。

输出可能是单个RGB图像(MxNx3矩阵，其中M和N是原始图像x维度，每个红色，绿色和蓝色通道各一个)。或者，可以返回大小为MxN的单个红色、绿色和蓝色矩阵。

还支持多维索引图像(大小为MxNx1xK)。

参见:rgb2ind, ind2gray, hsv2rgb。

Octave还提供了制作和使用电影框架结构的工具。这些结构将图像数据(“cdata”字段)与相应的颜色映射(“colormap”字段)封装在一起。

: frame = getframe ()

: frame = getframe (hax)

: frame = getframe (hfig)

: frame = getframe (…, rect)

捕捉一个图形或轴作为一个电影帧结构。

不带参数，捕获当前轴，不包括ticklabels、title和x/y/zlabels。返回的结构帧有一个字段cdata，它包含NxMx3 (RGB) uint8矩阵形式的实际图像数据，以及为MATLAB兼容性提供的字段colormap，但始终为空。

如果第一个参数hax是一个轴句柄，那么捕获这个轴，而不是gca返回的当前轴。

如果第一个参数hfig是一个图形句柄，则捕获整个对应的图形画布。

最后，如果提供第二个参数rect，它必须是一个四元素向量([left bottom width height])，定义要捕获的图形内部区域。不管图形“units”属性如何，rect必须以像素为单位定义。

参见:im2frame, frame2im, movie。

: movie (mov)

: movie (mov, n)

: movie (mov, n, fps)

: movie (h, …)

播放由一组帧结构定义的电影。

movie mov必须是带有字段“cdata”和“colormap”的帧结构数组，由getframe函数返回。默认情况下，所有图像在当前轴上以12 fps的速度显示一次。

可选参数n是一个标量或整数向量，控制电影显示的次数和特定帧的显示:

第一个元素:

n(1) > 0

Play the movie n(1) times.

n(1) < 0

Play the movie abs (n(1) times alternatively in forward and backward order.

其他要素(如有):

将显示的mov中帧的索引。

如果第一个参数是图形或轴h的句柄，则影片将在该图形或轴中播放，而不是在当前轴中播放。

参见:getframe, im2frame, frame2im。

: [x, map] = frame2im (frame)

将电影帧转换为索引图像。

一个电影帧只是一个带有字段“cdata”和“colormap”的结构体。

当frame是struct数组时，支持n维图像或电影。在这种情况下，x将分别为索引电影和RGB电影的MxNx1xK或MxNx3xK，每个帧沿着第4维连接。

参见:im2frame, getframe。

: frame = im2frame (rgb)

: frame = im2frame (x, map)

将图像转换为电影帧。

一个电影帧只是一个带有字段“cdata”和“colormap”的结构体。

当每个图像投影，RGB图像的MxN和MxNx3的矩阵大小沿着第四维连接时，给出了对n维图像的支持。在这种情况下，返回值是一个结构数组。

参见:frame2im。

colormap函数用于更改当前轴或图形的颜色映射。

: cmap = colormap ()

: cmap = colormap (map)

: cmap = colormap ("default")

: cmap = colormap (map\_name)

: cmap = colormap (hax, …)

: colormap map\_name

查询或设置当前颜色映射。

在没有输入参数的情况下，colormap返回当前颜色映射。

Colormap (map)设置当前的颜色映射为map。颜色映射应该是一个n行× 3列的矩阵。这些列分别包含红色、绿色和蓝色的强度。所有条目必须在0到1之间。返回新的颜色映射。

Colormap(“default”)恢复默认的Colormap(包含64个条目的viridis map)。返回默认的颜色映射。

映射也可以由字符串map\_name指定，该字符串是返回颜色映射的函数的名称。

如果第一个参数hax是一个轴句柄，则查询或设置这些轴的颜色映射。

为方便起见，也可以将此函数与命令形式colormap map\_name一起使用。

内置颜色映射列表如下:

| **Map** | **Description** |
| --- | --- |
| viridis | default |
| turbo | colormap traversing blue, cyan, green, yellow, red; modern replacement for jet. |
| jet | colormap traversing blue, cyan, green, yellow, red. |
| cubehelix | colormap traversing black, blue, green, red, white with increasing intensity. |
| hsv | cyclic colormap traversing Hue, Saturation, Value space. |
| rainbow | colormap traversing red, yellow, blue, green, violet. |
| ————- | ——————————————————————————————— |
| hot | colormap traversing black, red, orange, yellow, white. |
| cool | colormap traversing cyan, purple, magenta. |
| spring | colormap traversing magenta to yellow. |
| summer | colormap traversing green to yellow. |
| autumn | colormap traversing red, orange, yellow. |
| winter | colormap traversing blue to green. |
| ————- | ——————————————————————————————— |
| gray | colormap traversing black to white in shades of gray. |
| bone | colormap traversing black, gray-blue, white. |
| copper | colormap traversing black to light copper. |
| pink | colormap traversing black, gray-pink, white. |
| ocean | colormap traversing black, dark-blue, white. |
| ————- | ——————————————————————————————— |
| colorcube | equally spaced colors in RGB color space. |
| flag | cyclic 4-color map of red, white, blue, black. |
| lines | cyclic colormap with colors from axes *"ColorOrder"* property. |
| prism | cyclic 6-color map of red, orange, yellow, green, blue, violet. |
| ————- | ——————————————————————————————— |
| white | all white colormap (no colors). |

参见:翠绿，涡轮，喷射，立方螺旋，hsv，彩虹，热，冷，春，夏，秋，冬，灰色，骨，铜，粉红，海洋，彩色立方体，旗帜，线条，棱镜，白色。

: tf = iscolormap (cmap)

如果cmap是一个colormap，则返回true。

色图是一个真正的矩阵，类单或双，有3列。每一行代表一种颜色。这三列分别包含红色、绿色和蓝色的强度。

颜色图中的所有值都应该在[0 - 1]范围内，但这不是强制的。每个函数必须决定如何处理超出此范围的值。

参见:colormap, rgbplot。

以下函数返回预定义的颜色映射，与使用colormap函数按名称请求的颜色映射相同。

: rgbplot (cmap)

: rgbplot (cmap, style)

: h = rgbplot (…)

绘制颜色图的组成部分。

显示cmap有两种不同的样式:

profile (default)

为每个通道(红、绿、蓝)绘制颜色图的RGB线轮廓，并适当地为其绘制颜色线。每条线表示颜色图中RGB分量的强度。

composite

在x轴上绘制颜色图，这样就可以看到实际的索引颜色，而不是单个颜色组件。

可选的返回值h是创建的绘图的图形句柄。

运行demo rgbplot查看rgbplot的示例和每个样式选项。

参见:colormap。

: map = autumn ()

: map = autumn (n)

创建颜色映射。这个色图的范围从红色到橙色再到黄色。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = bone ()

: map = bone (n)

创建颜色映射。这个色图从黑到白，有灰蓝色的阴影。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = colorcube ()

: map = colorcube (n)

创建颜色映射。该色图由RGB色彩空间中尽可能多的等间距颜色(不是灰色)组成。

如果有规则间隔的颜色没有达到完美数量n，那么颜色图中剩下的条目就是纯红色、绿色、蓝色和灰色的渐变。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = cool ()

: map = cool (n)

创建颜色映射。色图从青色到品红不等。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = copper ()

: map = copper (n)

创建颜色映射。这个色图从黑色到浅铜色调不等。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = cubehelix ()

: map = cubehelix (n)

: map = cubehelix (n, start, rots, hue, gamma)

创建立方体螺旋颜色图。

这个色图从黑色到白色，经过蓝色、绿色和红色色调，同时保持单调增加的强度感知。这是通过螺旋将颜色立方体从黑色穿越到白色来实现的，因此称为立方螺旋，同时考虑到根据1953年NTSC规范的每个通道的感知亮度。

rgbplot (cubehelix (256))

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参考文献:Green, d.a.， 2011，天文强度图像显示的配色方案，印度天文学会公报，39,289。

参见:colormap。

: map = flag ()

: map = flag (n)

创建颜色映射。随着每次索引的变化，这个颜色图会在红色、白色、蓝色和黑色之间循环。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = gray ()

: map = gray (n)

创建灰色色图。这个色图从黑到白，带有灰色阴影。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = hot ()

: map = hot (n)

创建颜色映射。这个颜色图的范围从黑色到暗红色、红色、橙色、黄色到白色。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = hsv ()

: map = hsv (n)

创建颜色映射。这个色图从红色开始，经过黄色、绿色、青色、蓝色和品红，最后回到红色。

它对于显示周期函数很有用。通过所有可能的值线性变化色相，同时保持恒定的最大饱和度和值，获得地图。等效代码为hsv2rgb ([(0:N-1)'/N, ones(N,2)])。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = jet ()

: map = jet (n)

创建颜色映射。这个色图的范围从深蓝色到蓝色、青色、绿色、黄色、红色到暗红色。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

编程注意:喷射色图在感知上不是统一的。如果这很重要的话，可以试试翠绿色图。对于具有更好感知特性的jet，可以尝试涡轮颜色图。

参见:colormap, turbo。

: map = lines ()

: map = lines (n)

创建颜色映射。这个颜色图由当前轴“ColorOrder”属性中的颜色列表组成。默认为蓝色、橙色、黄色、紫色、绿色、浅蓝色和暗红色。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = ocean ()

: map = ocean (n)

创建颜色映射。这张色图从黑到白，中间有深浅不一的蓝色。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = pink ()

: map = pink (n)

创建颜色映射。这个色图从黑色到白色，带有灰粉色的阴影。

当在灰度图像上使用时，此色图给出了棕褐色的色调。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = prism ()

: map = prism (n)

创建颜色映射。随着每个指数的变化，这个颜色图通过红色、橙色、黄色、绿色、蓝色和紫色循环。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = rainbow ()

: map = rainbow (n)

创建颜色映射。这个色图的范围从红色到橙色、黄色、绿色、蓝色到紫色。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = spring ()

: map = spring (n)

创建颜色映射。这个色图从品红到黄色不等。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = summer ()

: map = summer (n)

创建颜色映射。这个色图从绿色到黄色不等。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = turbo ()

: map = turbo (n)

创建颜色映射。这个色图的范围从深蓝色到绿色再到暗红色;类似于过时的喷气机颜色图，但在感知上是一致的。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = viridis ()

: map = viridis (n)

创建颜色映射。这个颜色图的范围从深紫蓝色到蓝色、绿色到黄色。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = white ()

: map = white (n)

创建颜色映射。这个色图完全是白色的。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: map = winter ()

: map = winter (n)

创建颜色映射。这个色图从蓝色到绿色不等。

参数n必须是标量。如果未指定，则使用当前颜色映射的长度，即64。

参见:colormap。

: cmap = contrast (x)

: cmap = contrast (x, n)

返回使图像对比度最大化的灰色色图。

返回的颜色映射将有n行。如果n未定义，则使用当前颜色图的大小。

参见:彩色地图，照亮。

以下三个函数修改而不是替换现有的颜色图。

: map\_out = brighten (beta)

: map\_out = brighten (map, beta)

: map\_out = brighten (h, beta)

: brighten (…)

使色图变亮或变暗。

参数beta必须是-1到1之间的标量，其中负值使颜色图变暗，正值使颜色图变亮。

如果省略map参数，则将该函数应用于当前颜色映射。

第一个参数也可以是一个有效的图形句柄h，在这种情况下，将照亮应用于与此句柄相关联的颜色图。

如果未指定输出，则将结果写入当前颜色映射。

参见:色图，对比。

: spinmap ()

: spinmap (t)

: spinmap (t, inc)

: spinmap ("inf")

将颜色映射循环t秒，颜色增量为inc。

这两个参数都是可选的。周期时间默认为5秒，增量默认为2。如果给出了“inf”选项，则连续循环，直到按下Control-C。

旋转时，原来的颜色1变成颜色2，颜色2变成颜色3，以此类推。允许正增量或负增量，较高的inc值将导致更快地通过颜色图循环。

参见:colormap。

: whitebg ()

: whitebg (color)

: whitebg ("none")

: whitebg (hfig)

: whitebg (hfig, color)

: whitebg (hfig, "none")

反转当前配色方案中的颜色。

根属性也被颠倒，这样所有后续的图都将使用新的配色方案。

如果存在可选参数color，则将背景颜色设置为color而不是反转。color可以是表示八种已知颜色之一的字符串，也可以是RGB三元组。特殊字符串参数"none"将绘图恢复为出厂默认颜色。

如果第一个参数hfig是图形句柄或图形句柄列表，则操作这些图形，而不是gcf返回的当前图形。除非0出现在数字列表中，否则不会更改根属性。

编程注意:whitebg通过改变指定图形的子元素的颜色属性来操作。只有具有单一颜色的对象会受到影响。例如，具有单个“FaceColor”的补丁将被更改，但具有阴影(“interp”)的补丁将不会被修改。对于反演，新的颜色就是简单的RGB空间的反演:cnew = [1- r1 - g1 - b]。指定颜色后，轴和图形被设置为新颜色，然后调整子对象的颜色，使其与新背景具有一定的对比度(可见性)。

参见:reset, get, set。

以下函数可用于操作颜色映射。

: [Y, newmap] = cmunique (X, map)

: [Y, newmap] = cmunique (RGB)

: [Y, newmap] = cmunique (I)

将输入图像X转换为使用最小颜色图newmap的输出索引图像Y。

当输入是一个索引图像(带有colormap map的X)时，输出是一个colormap newmap，其中任何重复的行都已被消除。输出图像Y是原始输入图像，其索引经过调整以匹配新的(可能更小的)颜色映射。

当输入是RGB图像(MxNx3数组)时，输出颜色映射将为原始图像中的每个唯一颜色包含一个条目。在最坏的情况下，新地图的行数可能与原始图像中的像素数一样多。

当输入是灰度图像I时，输出颜色图将包含原始图像中每个唯一强度值的一个条目。在最坏的情况下，新地图的行数可能与原始图像中的像素数一样多。

实现细节:

newmap总是一个Mx3矩阵，即使输入图像是灰度图像I(所有三个RGB平面被分配相同的值)。

如果新的颜色映射的大小小于或等于256，则输出图像的类型为uint8。否则，输出图像为double类。

参见:rgb2ind, gray2ind。

: [Y, newmap] = cmpermute (X, map)

: [Y, newmap] = cmpermute (X, map, index)

重新排序颜色映射中的颜色。

当只使用两个参数调用时，cmpermute会随机重新排列colormap映射并返回一个新的colormap newmap。它还返回索引的图像Y，当使用newmap显示时，它相当于原始输入图像X。

当使用可选的第三个参数调用时，新colormap中的颜色顺序由index定义。

注意:索引不应该有重复的元素，否则函数会失败。

**32.4在图像顶部绘图**

如果使用gnuplot来显示图像，则可以在图像上进行绘图。由于图像是一个矩阵，因此它是按行和列值索引的。然而，绘图系统是基于传统的(x, y)系统。为了最小化两个系统之间的差异，Octave将坐标系统的原点放置在(1,1)处像素对应的点上。因此，要在图像上绘制由行值和列值给出的点，只需调用plot，列值作为第一个参数，行值作为第二个参数。作为一个例子，下面的代码生成了一个随机强度在0到1之间的图像，并在强度高于0.99的像素上显示了红色圆圈。

I = rand (100, 100);

[row, col] = find (I > 0.99);

hold ("on");

imshow (I);

plot (col, row, "ro");

hold ("off");

**32.5色彩转换**

Octave支持从RGB颜色系统到HSV颜色系统的转换，反之亦然。还可以将彩色RGB图像转换为灰度图像。

: hsv\_map = rgb2hsv (rgb\_map)

: hsv\_img = rgb2hsv (rgb\_img)

将颜色图或图像从RGB转换为HSV颜色空间。

RGB空间中的颜色由红、绿、蓝三种强度组成。

在HSV空间中，颜色由色相、饱和度和值(亮度)在圆柱坐标系中表示。色相是方位角，描述主色调。饱和度是径向距离，给出了混合到颜色中的色调的数量。值是高度，是颜色中的光量。

输出类和大小将与输入相同。

参见:hsv2rgb, rgb2ind, rgb2gray。

: rgb\_map = hsv2rgb (hsv\_map)

: rgb\_img = hsv2rgb (hsv\_img)

将颜色图或图像从HSV转换为RGB颜色空间。

在HSV空间中，颜色由色相、饱和度和值(亮度)在圆柱坐标系中表示。色相是方位角，描述主色调。饱和度是径向距离，给出了混合到颜色中的色调的数量。值是高度，是颜色中的光量。

输入既可以是彩色图，也可以是RGB图像。在浮点输入的情况下，值应该在[0 1]范围内。在hue(方位角)的情况下，由于值对应于一个角度，所以使用mod (h, 1)。

>> hsv2rgb ([0.5 1 1])

⇒ ans = 0 1 1

>> hsv2rgb ([2.5 1 1])

⇒ ans = 0 1 1

>> hsv2rgb ([3.5 1 1])

⇒ ans = 0 1 1

输出类和大小将与输入相同。

参见:rgb2hsv, ind2rgb。

: I = rgb2gray (rgb\_img)

: gray\_map = rgb2gray (rgb\_map)

将图像或色图从红绿蓝(RGB)色彩空间转换为灰度强度图像。

输入类型可以是uint8、int8、uint16、int16、single或double。输出与输入属于同一类。

实现说明:灰度强度计算为

I = 0.298936\*R + 0.587043\*G + 0.114021\*B

当RGB被转换成YIQ时对应的亮度通道，如https://en.wikipedia.org/wiki/YIQ所记录的。

参见:rgb2hsv, rgb2ind。