**33音频处理**

**33.1音频文件实用程序**

以下函数允许您读取、写入和检索音频文件的信息。支持各种格式，包括wav, flac和ogg vorbis。

: info = audioinfo (filename)

返回由filename指定的音频文件的信息。

输出信息是一个包含以下字段的结构:

‘Filename’

音频文件的名称。

‘CompressionMethod’

音频压缩方法。未使用，仅用于与MATLAB兼容。

‘NumChannels’

声道数

‘SampleRate’

音频的采样率，单位为赫兹。

‘TotalSamples’

文件中的样本数。

‘Duration’

音频持续时间，以秒为单位。

‘BitsPerSample’

每个样本的位数。

‘BitRate’

音频比特率。未使用，仅用于与MATLAB兼容。

‘Title’

“Title”音频元数据值为字符串，如果不存在则为空。

‘Artist’

“Artist”音频元数据值为字符串，如果不存在则为空。

‘Comment’

“注释”音频元数据值为字符串，如果不存在则为空。

参见:audioread, audiowrite。

: [y, fs] = audioread (filename)

: [y, fs] = audioread (filename, samples)

: [y, fs] = audioread (filename, datatype)

: [y, fs] = audioread (filename, samples, datatype)

读取音频文件文件名并返回音频数据y和采样率fs。

音频数据存储为矩阵，行对应于音频帧，列对应于通道。

可选的双元素向量参数samples指定开始帧和结束帧。

可选参数datatype指定要返回的数据类型。如果它是“本机”，那么数据的类型取决于数据如何存储在音频文件中。

参见:audiowrite, audioformats, audioinfo。

: audiowrite (filename, y, fs)

: audiowrite (filename, y, fs, name, value, …)

以采样率fs从矩阵y写入音频数据到文件名，文件格式由文件扩展名决定。

额外的名称/值参数对可以用来指定以下选项:

‘BitsPerSample’

每个样本的位数。取值为8、16、24、32。默认为16。

‘BitRate’

有效的参数名称，但被忽略。为与MATLAB兼容而留下。

‘Quality’

Ogg Vorbis压缩机的质量设置。取值范围为0到100，其中100为最高质量设置。默认值是75。

‘Title’

音频文件的标题。

‘Artist’

艺术家姓名。

‘Comment’

注释。

参见:audioread, audioformats, audioinfo。

: audioformats ()

: audioformats (format)

显示所有支持的音频格式信息。

如果给出了可选参数format，则只显示名称以format开头的格式。

参见:audioread, audiowrite。

**33.2音频设备信息**

: devinfo = audiodevinfo ()

: devs = audiodevinfo (io)

: name = audiodevinfo (io, id)

: id = audiodevinfo (io, name)

: driverversion = audiodevinfo (io, id, "DriverVersion")

: id = audiodevinfo (io, rate, bits, chans)

: supports = audiodevinfo (io, id, rate, bits, chans)

返回描述可用音频输入和输出设备的结构。

devinfo结构有“input”和“output”两个字段。每个字段的值是一个结构数组，其中“Name”，“DriverVersion”和“ID”字段描述音频设备。

如果可选参数io为1，则只返回有关输入设备的信息。如果为0，则只返回输出设备的信息。如果io是提供的唯一参数，则返回可用输入或输出设备的数量。

如果提供了可选参数id，则返回相应设备的信息。

如果提供了可选参数name，则返回指定设备的ID。

如果给出了可选参数“DriverVersion”，则返回指定设备的驱动程序名称。

给定一个输入或输出设备的采样率、每个采样的比特数和通道数，返回使用指定参数支持播放或记录的第一个设备的ID。

如果还给定一个设备ID，如果设备支持使用这些参数播放或录制，则返回true。

**33.3音频播放器**

下面的方法用于创建和使用audioplayer对象。这些对象可用于回放存储在Octave矩阵和数组中的音频数据。audioplayer对象支持从系统可用的各种设备播放，阻塞和非阻塞播放，方便的暂停和恢复等等。

: player = audioplayer (y, fs)

: player = audioplayer (y, fs, nbits)

: player = audioplayer (y, fs, nbits, id)

: player = audioplayer (recorder)

: player = audioplayer (recorder, id)

创建一个audioplayer对象，它将以采样率fs播放数据y。

信号y可以是矢量(单声道音频)或二维数组(多声道音频)。

可选参数nbits和id分别指定每个样本和播放器设备id的位数。设备id可以通过audiodevinfo函数找到。

给定一个audiorecorder对象，使用该对象的数据来初始化播放器。

audioplayer对象的操作列表如下所示。所有方法都需要audioplayer对象作为第一个参数。

| **Method** | **Description** |
| --- | --- |
| get | Read audioplayer property values |
| isplaying | Return true if audioplayer is playing |
| pause | Pause audioplayer playback |
| play | Play audio stored in audioplayer object w/o blocking |
| playblocking | Play audio stored in audioplayer object |
| resume | Resume playback after pause |
| set | Write audioplayer property values |
| stop | Stop playback |

例子

创建一个audioplayer对象，它将以44100采样率播放一秒钟的白噪声，每个采样使用8位。

y = 0.25 \* randn (2, 44100);

player = audioplayer (y, 44100, 8);

play (player);

参见:@audioplayer/get， @audioplayer/isplaying， @audioplayer/pause， @audioplayer/play， @audioplayer/playblocking， @audioplayer/resume， @audioplayer/set， @audioplayer/stop, audiodevinfo， @audiorecorder/audiorecorder, sound, soundsc。

**33.3.1回放**

以下方法用于控制播放器播放。

: play (player)

: play (player, start)

: play (player, [start, end])

播放存储在audioplayer对象播放器中的音频而不阻塞。

如果提供了可选参数start，则开始在录音中播放start样本。

如果提供了可选参数end，则在录制结束时停止播放采样。

参见:@audioplayer/播放阻塞，@audioplayer/暂停，@audioplayer/停止，@audioplayer/audioplayer。

: playblocking (player)

: playblocking (player, start)

: playblocking (player, [start, end])

播放音频存储在audioplayer对象播放器与阻塞(同步I/O)。

如果提供了可选参数start，则开始在录音中播放start样本。

如果提供了可选参数end，则在录制结束时停止播放采样。

参见:@audioplayer/播放，@audioplayer/暂停，@audioplayer/停止，@audioplayer/audioplayer。

: pause (player)

暂停播放音频播放器播放器。

参见:@audioplayer/resume， @audioplayer/stop， @audioplayer/audioplayer。

: resume (player)

为暂停的audioplayer对象播放器恢复播放。

参见:@audioplayer/pause， @audioplayer/stop， @audioplayer/audioplayer。

: stop (player)

停止audioplayer播放器的播放，并将相关变量重置为初始值。

参见:@audioplayer/暂停，@audioplayer/恢复，@audioplayer/audioplayer。

: tf = isplaying (player)

如果audioplayer对象播放器当前正在播放音频，则返回true，否则返回false。

参见:@audioplayer/pause， @audioplayer/audioplayer。

**33.3.2属性**

剩下的两个方法用于获取和设置audioplayer对象的各种属性。

: value = get (player, name)

: values = get (player, {name1, name2, …})

: values = get (player)

返回由名称标识的属性的值。

如果name是单元格数组，则返回由单元格数组的元素标识的属性值。只给定播放器对象，返回一个带有播放器所有属性值的标量结构。结构的字段名对应于属性名。

参见:@audioplayer/set， @audioplayer/audioplayer。

: set (player, name, value)

: set (player, name\_cell, value\_cell)

: set (player, properties\_struct)

: properties = set (player)

将名称指定的属性的值设置为给定值。

如果名称和值是单元格数组，则将每个属性设置为相应的值。给定一个属性结构，其中的字段与属性名相对应，将这些属性的值设置为相应的字段值。给定一个audioplayer对象，返回一个可配置属性(即可写属性)的结构。

参见:@audioplayer/get， @audioplayer/audioplayer。

**33.4录音机**

下面的方法用于创建和使用audiorecorder对象。这些对象可用于记录来自系统可用的各种设备的音频数据。您可以使用方便的方法来检索该数据或从该数据创建的audioplayer对象。阻塞和非阻塞记录、暂停和恢复记录以及更多可用的方法。

: recorder = audiorecorder ()

: recorder = audiorecorder (fs, nbits, nchannels)

: recorder = audiorecorder (fs, nbits, nchannels, id)

创建一个音频记录对象，记录8000hz采样率的8位单声道音频。

可选参数fs、nbits、nchannels和id分别指定采样率、每个采样的比特数、通道数和记录设备id。设备id可以通过audiodevinfo函数找到。

audiorecorder对象的操作列表如下所示。所有方法都需要audiorecorder对象作为第一个参数。

| **Method** | **Description** |
| --- | --- |
| get | Read audiorecorder property values |
| getaudiodata | Return audio data as a numeric matrix |
| getplayer | Return audioplayer loaded with data from audiorecorder |
| isrecording | Return true if audiorecorder is recording |
| pause | Pause recording |
| play | Play audio stored in audiorecorder object |
| record | Record audio in audiorecorder object w/o blocking |
| recordblocking | Record audio in audiorecorder object |
| resume | Resume recording after pause |
| set | Write audiorecorder property values |
| stop | Stop recording |

参见:@audiorecorder/get， @audiorecorder/getaudiodata， @audiorecorder/getplayer， @audiorecorder/isrecording， @audiorecorder/pause， @audiorecorder/play， @audiorecorder/record， @audiorecorder/recordblocking， @audioplayer/resume， @audiorecorder/set， @audiorecorder/stop, audiodevinfo， @audioplayer/audioplayer, record。

**33.4.1记录**

以下方法控制记录过程。

: record (recorder)

: record (recorder, length)

使用audiorecorder对象录制音频而不阻塞，直到暂停或停止方法暂停或停止。

给定可选参数length，记录长度秒。

参见:@audiorecorder/recordblocking， @audiorecorder/audiorecorder。

: recordblocking (recorder, length)

记录音频阻塞(同步I/O)。

必须指定录音的长度，单位为秒(length)。

参见:@audiorecorder/record， @audiorecorder/audiorecorder。

: pause (recorder)

暂停录音。

参见:@audiorecorder/resume， @audiorecorder/stop， @audiorecorder/audiorecorder。

: resume (recorder)

使用暂停的音频记录器对象记录器恢复录音。

参见:@audiorecorder/pause， @audiorecorder/stop， @audiorecorder/audiorecorder。

: stop (recorder)

停止录音与audiorecorder对象记录器和清理任何音频流。

参见:@audiorecorder/暂停，@audiorecorder/恢复，@audiorecorder/audiorecorder。

: tf = isrecording (recorder)

如果audiorecorder对象recorder当前正在录制音频，则返回true，否则返回false。

参见:@audiorecorder/pause， @audiorecorder/audiorecorder。

33.4.2数据检索

以下方法允许您以各种方式检索录制的音频数据。

: data = getaudiodata (recorder)

: data = getaudiodata (recorder, datatype)

从audiorecorder对象recorder返回音频数据，作为双矩阵，其值在-1.0和1.0之间，并且与记录器中的通道一样多列。

给定可选参数datatype，将记录的数据转换为指定的类型，该类型可能是“double”，“single”，“int16”，“int8”或“uint8”之一。

参见:@audiorecorder/audiorecorder。

: player = getplayer (recorder)

返回一个audioplayer对象，其中包含由audiorecorder对象recorder记录的数据。

参见:@audioplayer/audioplayer， @audiorecorder/audiorecorder。

: player = play (recorder)

: player = play (recorder, start)

: player = play (recorder, [start, end])

播放录制在记录器中的音频而不阻塞并返回一个相应的audioplayer对象。

如果提供了可选参数start，则开始播放录制的开始秒。

如果提供了可选参数end，则在录制结束时停止播放。

参见:@audiorecorder/getplayer， @audioplayer/audioplayer， @audiorecorder/audiorecorder。

33.4.3属性

剩下的两个方法允许您读取或更改audiorecorder对象的属性。

: value = get (recorder, name)

: values = get (recorder, {name1, name2, …})

: values = get (recorder)

返回由名称标识的属性的值。

如果name是单元格数组，则返回由单元格数组的元素标识的属性值。仅给定记录器对象，返回一个带有记录器所有属性值的标量结构。结构的字段名对应于属性名。

参见:@audiorecorder/set， @audiorecorder/audiorecorder。

: set (recorder, name, value)

: set (recorder, name\_cell, value\_cell)

: set (recorder, properties\_struct)

: properties = set (recorder)

将名称指定的属性的值设置为给定值。

如果名称和值是单元格数组，则将每个属性设置为相应的值。给定一个属性结构，其中的字段与属性名相对应，将这些属性的值设置为相应的字段值。只给定一个记录器对象，返回一个可配置属性(即可写属性)的结构。

参见:@audiorecorder/get， @audiorecorder/audiorecorder。

33.5音频数据处理

Octave提供了一些处理音频数据的函数。音频“样本”是a /D转换器的单个输出值，即一个小整数(通常为8或16位)，音频数据只是一系列这样的样本。它可以用三个参数来表征:采样率(以每秒采样数或Hz为单位测量，例如，8000或44100)，每个样本的位数(例如，8或16)和通道数(1为单声道，2为立体声等)。

有许多不同的格式来表示这类数据。目前，Octave只支持两种最流行的编码，线性编码和mu-law编码。Guido van Rossum有一个关于音频格式的优秀FAQ guido@cwi.nl，可以在任何FAQ ftp站点找到，特别是在目录/pub/usenet/news中。档案网站rtfm.mit.edu的答案/音频文件。

Octave简单地将音频数据视为样本向量(非单声道数据尚不支持)。假设使用线性编码的音频文件具有扩展名lin或raw之一，并且以mu-law编码保存数据的文件以au、mu或snd结尾。

: y = lin2mu (x)

: y = lin2mu (x, n)

将音频数据从线性转换为mu-law。

如果n为0(默认)，线性值使用范围为-1≤x≤1的浮点值，如果n为8或16，则使用n位有符号整数。mu律值是8位无符号整数，取值范围为0≤y≤255。

参见:mu2lin。

: y = mu2lin (x)

: y = mu2lin (x, n)

将音频数据从mu-law转换为线性。

mu律值是8位无符号整数，取值范围为0≤y≤255。当n为零(默认值)时，线性值使用-linmax≤x linmax(其中linmax = 32124/32768 =~ 0.98)范围内的浮点值。如果n是8或16，则使用n位有符号整数。

编程注意:mu2lin将最大mu律输入映射到略低于可用线性比例尺表示的最大值([-0.98，+0.98])的值。因此，mu2lin (lin2mu (x))可能无法重现原始输入。

参见:lin2mu。

: data = record (sec)

: data = record (sec, fs)

以每秒8000个采样率记录系统默认音频输入的秒秒音频。

如果给出了可选参数fs，它将指定记录的采样率。

要对音频录制进行更多控制，请使用audiorecorder类。

参见:@audiorecorder/audiorecorder, sound, soundsc。

: sound (y)

: sound (y, fs)

: sound (y, fs, nbits)

以采样率fs播放音频数据到默认音频设备。

音频信号y可以是矢量或两列数组，分别表示单声道或立体声音频。

如果没有给出fs，则使用每秒8000个样本的默认采样率。

可选参数nbits指定要播放到音频设备的位深度，默认为8位。

要对音频播放进行更多控制，请使用audioplayer类。

参见:soundsc， @audioplayer/audioplayer, record。

: soundsc (y)

: soundsc (y, fs)

: soundsc (y, fs, nbits)

: soundsc (…, [ymin, ymax])

缩放音频数据y并以采样率fs播放到默认音频设备。

音频信号y可以是矢量或两列数组，分别表示单声道或立体声音频。

如果没有给出fs，则使用每秒8000个样本的默认采样率。

可选参数nbits指定要播放到音频设备的位深度，默认为8位。

默认情况下，y被自动归一化为[- 1,1]的范围。如果给定范围[ymin, ymax]，则y中处于ymin≤y≤ymax范围内的元素将被缩放到[- 1,1]范围。

要对音频播放进行更多控制，请使用audioplayer类。

参见:声音，@audioplayer/audioplayer，记录。

**34面向对象编程**

Octave能够创建用户定义的类——包括操作符和函数重载的功能。类可以保护内部属性，这样它们就不会被意外地改变，这有利于数据封装。此外，还可以创建规则来解决混合类操作中的类优先级问题。

本章讨论构造用户类的方法，如何查询和设置类的属性，以及如何重载操作符和函数。在本章中，使用为多项式设计的类给出了实际代码示例。

**34.1创建类**

本章通过一个为多项式设计的自定义类来说明用户定义类和面向对象编程。选择这个类是因为它的简单性，它不会不必要地分散对Octave编程特性的讨论。即便如此，在介绍Octave面向对象编程的语法和技术之前，有必要对多项式类的目标有一些背景知识。

多项式类用于表示如下形式的多项式

a0 + a1 \* x + a2 \* x^2 + … + an \* x^n

其中a0 a1等都是实标量。因此，多项式可以用一个向量表示

a = [a0, a1, a2, …, an];

这是开始为多项式类编写构造函数的充分规范。Octave中的所有面向对象类必须位于一个目录中，该目录是以类的名称加上“@”符号。例如，多项式类将在@多项式目录中定义其所有方法。

类的构造函数必须是类本身的名称;在本例中，构造函数驻留在文件@多项式/多项式.m中。理想情况下，即使在没有参数的情况下调用构造函数，也应该返回一个有效的对象。多项式类的构造函数可能如下所示

## -\*- texinfo -\*-

## @deftypefn {} {} polynomial ()

## @deftypefnx {} {} polynomial (@var{a})

## Create a polynomial object representing the polynomial

##

## @example

## a0 + a1 \* x + a2 \* x^2 + @dots{} + an \* x^n

## @end example

##

## @noindent

## from a vector of coefficients [a0 a1 a2 @dots{} an].

## @end deftypefn

function p = polynomial (a)

if (nargin == 0)

p.poly = 0;

p = class (p, "polynomial");

else

if (isa (a, "polynomial"))

p = a;

elseif (isreal (a) && isvector (a))

p.poly = a(:).'; # force row vector

p = class (p, "polynomial");

else

error ("@polynomial: A must be a real vector");

endif

endif

endfunction

注意，构造函数的返回值必须是类函数的输出。类函数的第一个参数是一个结构，第二个参数是类本身的名称。调用类构造函数来创建实例的示例如下

p = polynomial ([1, 0, 1]);

方法由类目录中的m-files定义，并且可以具有与任何其他m-file相同的嵌入式文档。构造函数的帮助可以通过单独使用构造函数名称获得，也就是说，对于多项式构造函数，帮助多项式将返回帮助字符串。通过使用类目录名后跟方法，可以将帮助限制为特定的类。例如，help @多项式/多项式是为多项式构造函数显示帮助字符串的另一种方式。第二种方法是为类的重载方法和函数获取帮助的唯一方法。

在Octave需要函数名的任何地方都可以使用相同的规范机制。例如，type @多项式/disp将把多项式类的disp方法的代码打印到屏幕上，dbstop @多项式/disp将在多项式类的disp方法的第一个可执行行设置一个断点。

要检查变量是否属于用户类，可以使用isobject和isa函数。例如:

p = polynomial ([1, 0, 1]);

isobject (p)

⇒ 1

isa (p, "polynomial")

⇒ 1

: tf = isobject (x)

如果x是一个类对象，则返回true。

参见:class, typeinfo, isa, ismethod, isprop。

类的可用方法可以通过方法函数显示。

: methods (obj)

: methods ("classname")

: methods (…, "-full")

: mtds = methods (…)

列出对象obj或命名类classname的公共方法的名称。

obj可以是Octave类对象或Java对象。classname可以是Octave类或Java类的名称。

如果给出了可选参数"-full"，则Octave返回完整的方法签名，其中包括输出类型，方法名称以及输入的数量和类型。

在没有输出参数的情况下调用方法时，方法会将方法名称列表打印到屏幕上。否则，输出参数mtds将该列表包含在字符串的单元格数组中。

参见:ismethod, properties, fieldnames。

要查询用户类的特定方法是否存在，可以使用ismethod函数。

: tf = ismethod (obj, method)

: tf = ismethod (class\_name, method)

如果string方法是对象obj或类clsname的有效方法，则返回true。

参见:isprop, isobject, isjava, methods。

对于多项式类，有一个方法来计算它的根是有意义的。

function r = roots (p)

r = roots (fliplr (p.poly));

endfunction

我们可以通过调用root方法来检查是否存在:

p = polynomial ([1, 0, 1]);

ismethod (p, "roots")

⇒ 1

**34.2类方法**

可以(也应该)定义许多基本的类方法，以允许查询和设置类的内容。其中最基本的是碟子法。每当一个类需要显示在屏幕上时，Octave就会使用disp方法。通常这是一个没有以分号结尾的Octave表达式的结果。如果没有定义这个方法，那么Octave在显示类的内容时不会打印任何东西，这可能会令人困惑。

多项式类的disp方法的一个示例如下

function disp (p)

a = p.poly;

first = true;

for i = 1 : length (a);

if (a(i) != 0)

if (first)

first = false;

elseif (a(i) > 0 || isnan (a(i)))

printf (" +");

endif

if (a(i) < 0)

printf (" -");

endif

if (i == 1)

printf (" %.5g", abs (a(i)));

elseif (abs (a(i)) != 1)

printf (" %.5g \*", abs (a(i)));

endif

if (i > 1)

printf (" X");

endif

if (i > 2)

printf (" ^ %d", i - 1);

endif

endif

endfor

if (first)

printf (" 0");

endif

printf ("\n");

endfunction

为了与Octave图形句柄类保持一致，类还应该定义get和set方法。get方法接受一个或两个参数。第一个参数是相应类的对象。如果没有给出第二个参数，那么该方法应该返回一个包含类所有属性的结构体。如果给出了可选的第二个参数，它应该是一个属性名，并且应该检索指定的属性。

function val = get (p, prop)

if (nargin < 1)

print\_usage ();

endif

if (nargin == 1)

val.poly = p.poly;

else

if (! ischar (prop))

error ("@polynomial/get: PROPERTY must be a string");

endif

switch (prop)

case "poly"

val = p.poly;

otherwise

error ('@polynomial/get: invalid PROPERTY "%s"', prop);

endswitch

endif

endfunction

类似地，set方法的第一个参数应该是一个对象，任何其他参数应该是属性/值对。

function pout = set (p, varargin)

if (numel (varargin) < 2 || rem (numel (varargin), 2) != 0)

error ("@polynomial/set: expecting PROPERTY/VALUE pairs");

endif

pout = p;

while (numel (varargin) > 1)

prop = varargin{1};

val = varargin{2};

varargin(1:2) = [];

if (! ischar (prop) || ! strcmp (prop, "poly"))

error ("@polynomial/set: invalid PROPERTY for polynomial class");

elseif (! (isreal (val) && isvector (val)))

error ("@polynomial/set: VALUE must be a real vector");

endif

pout.poly = val(:).'; # force row vector

endwhile

endfunction

注意，Octave没有实现引用传递;因此，要修改对象，需要使用set方法的返回值执行赋值语句。

p = set (p, "poly", [1, 0, 0, 0, 1]);

set方法使用类的subsassign方法，因此也必须定义这个方法。下一节将更详细地讨论子分配方法(请参阅索引对象)。

最后，用户类可以被认为是结构体的一种特殊类型，它们可以像结构体一样保存到文件中。例如:

p = polynomial ([1, 0, 1]);

save userclass.mat p

clear p

load userclass.mat

支持保存和加载所支持的所有文件格式。在某些情况下，用户类可能包含保存没有意义的字段，或者需要在保存之前初始化的字段。这可以通过类的saveobj方法来完成。

: b = saveobj (a)

方法，以便在将对象保存到文件之前对其进行操作。

当使用save函数保存对象a时调用saveobj函数。使用saveobj的一个例子可能是删除对象中不值得保存的字段，或者可以使用它来确保在保存对象之前初始化对象的某些字段。例如:

function b = saveobj (a)

b = a;

if (isempty (b.field))

b.field = initfield (b);

endif

endfunction

参见:loadobj, class。

Saveobj在将类保存到文件之前被调用。类似地，loadobj方法仅在从文件加载类之后调用，可用于确保将任何删除的字段重新插入到用户对象中。

: b = loadobj (a)

方法，以便在从文件加载对象后对其进行操作。

当使用load函数加载对象a时调用loadobj函数。使用saveobj的一个例子可能是向对象中添加不需要保存的字段。例如:

function b = loadobj (a)

b = a;

b.addmissingfield = addfield (b);

endfunction

参见:saveobj, class。

**34.3索引对象**

34.3.1定义索引和索引赋值

对象可以用括号或大括号进行索引，像obj(idx)或obj{idx}，甚至像obj(idx).field。然而，由程序员来决定这个索引的实际含义。在多项式类的情况下，p(n)可能意味着多项式的n次幂的系数，也可能是多项式在n处的值。这个带下标的引用的含义由subsref方法决定。

: newval = subsref (val, idx)

根据idx指定的下标对val执行下标元素选择操作。

下标idx必须是一个具有' type '和' subs '字段的结构数组。type的有效值为"()"、"{}"和"。"。' subs '字段可以是":"或索引值的单元格数组。

下面的示例展示了如何提取矩阵的前两列

val = magic (3)

⇒ val = [ 8 1 6

3 5 7

4 9 2 ]

idx.type = "()";

idx.subs = {":", 1:2};

subsref (val, idx)

⇒ [ 8 1

3 5

4 9 ]

注意，这与写入val(:， 1:2)是一样的。

如果idx是一个带有字段' type '和' subs '的空结构数组，则返回val。

关键字end不能在subsref中用于索引分配。

参见:subsassign, substruct。

例如，这个类使用约定，用“()”索引计算多项式，用“{}”索引返回(n次幂的)第n个系数。subsref方法的代码如下所示

function r = subsref (p, s)

if (isempty (s))

error ("@polynomial/subsref: missing index");

endif

switch (s(1).type)

case "()"

idx = s(1).subs;

if (numel (idx) != 1)

error ("@polynomial/subsref: need exactly one index");

endif

r = polyval (fliplr (p.poly), idx{1});

case "{}"

idx = s(1).subs;

if (numel (idx) != 1)

error ("@polynomial/subsref: need exactly one index");

endif

if (isnumeric (idx{1}))

r = p.poly(idx{1}+1);

else

r = p.poly(idx{1});

endif

case "."

fld = s.subs;

if (! strcmp (fld, "poly"))

error ('@polynomial/subsref: invalid property "%s"', fld);

endif

r = p.poly;

otherwise

error ("@polynomial/subsref: invalid subscript type");

endswitch

if (numel (s) > 1)

r = subsref (r, s(2:end));

endif

endfunction

下标赋值的等效功能使用subassign方法。

: newval = subsasgn (val, idx, rhs)

按照idx指定的下标进行下标赋值操作。

下标idx必须是一个具有' type '和' subs '字段的结构数组。type的有效值为"()"、"{}"和"。"。' subs '字段可以是":"或索引值的单元格数组。

下面的示例显示如何将3 × 3矩阵的前两列设置为零。

val = magic (3);

idx.type = "()";

idx.subs = {":", 1:2};

val = subsasgn (val, idx, 0)

⇒ [ 0 0 6

0 0 7

0 0 2 ]

注意，这与写val(:， 1:2) = 0是一样的。

如果idx是一个包含字段' type '和' subs '的空结构数组，则返回rhs。

关键字end不能在子分配中用于索引分配。

参见:subsref, substruct, optimize\_subsasgn\_calls。

: val = optimize\_subsasgn\_calls ()

: old\_val = optimize\_subsasgn\_calls (new\_val)

: old\_val = optimize\_subsasgn\_calls (new\_val, "local")

查询或设置子分配方法调用优化的内部标志。

如果为true，在调用用户定义类的subsassign方法时，Octave将尝试消除冗余复制。

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:subssign。

注意，subsref和subsassign方法总是接收整个索引链，而它们通常只处理第一个元素。这些方法负责处理链的其余部分(如果需要)，通常将其再次转发给subsref或subsassign。

如果希望在对象的下标表达式中使用end关键字，则必须定义end方法。例如，多项式类的end方法可能看起来像

function r = end (obj, index\_pos, num\_indices)

if (num\_indices != 1)

error ("polynomial object may only have one index");

endif

r = length (obj.poly) - 1;

endfunction

这是一个相当通用的end方法，其行为类似于Octave Array类的end关键字。一个使用多项式类的例子是

p = polynomial ([1,2,3,4]);

p{end-1}

⇒ 3

对象本身也可以用作下标表达式中的索引，这是由subsindex函数控制的。

: idx = subsindex (obj)

将对象转换为索引向量。

当obj是用类构造函数定义的类对象时，subsindex是允许将该类对象转换为有效索引向量的重载方法。重要的是要注意subsindex必须返回一个“double”类的从零开始的实整数向量。例如，如果类构造函数为

function obj = myclass (a)

obj = class (struct ("a", a), "myclass");

endfunction

然后是subsindex函数

function idx = subsindex (obj)

idx = double (obj.a) - 1.0;

endfunction

可以这样使用吗

a = myclass (1:4);

b = 1:10;

b(a)

⇒ 1 2 3 4

参见:class, subsref, subsassign。

最后，通过提供冒号方法，可以像使用范围一样使用对象。

: r = colon (base, limit)

: r = colon (base, increment, limit)

返回与基数、极限和可选的增量对应的冒号表达式的结果。

该函数等价于base: limit或base: increment: limit操作符语法。

参见:linspace。

34.3.2索引分配优化

Octave无处不在的惰性复制按值传递语义意味着用户定义的子分配方法的性能问题。想象一下下面对subsassign的调用

ss = substruct ("()", {1});

x = subsasgn (x, ss, 1);

其中对应的方法看起来像这样:

function x = subsasgn (x, ss, val)

…

x.myfield (ss.subs{1}) = val;

endfunction

问题是，在进入subassign方法时，x仍然是从调用者的作用域引用的，这意味着在执行赋值之前，该方法首先需要取消共享(复制)x和x.myfield。在完成调用后，除非发生错误，否则结果将立即赋值给调用者作用域中的x，因此x.myfield的先前值将被遗忘。

因此，Octave语言意味着N个元素的副本(N是x.myfield的大小)，其中只需修改单个元素实际上就足够了。换句话说，一个恒定时间的操作被降级为线性时间的操作。对于本质上存储大数组的用户类来说，这可能是一个真正的问题。

为了部分解决这个问题，Octave对用户定义的子分配方法进行了特殊的优化，这些方法编码为m-files。当方法作为内置赋值语法的结果被调用时(不是上面所示的直接subsasgn调用)，即x(1) = 1，并且如果subsasgn方法声明为相同的输入和输出参数，如上面的例子所示，那么Octave将忽略调用者作用域中x的副本;因此，在方法执行期间对x所做的任何更改也将直接影响调用方的副本。例如，这允许定义一个多项式类，其中修改单个元素需要常数时间。

理解这种优化带来的含义是很重要的。因为在调用者的作用域中不会存在x的额外副本，所以如果在执行过程中发生错误，不让x处于无效状态完全是被调用者的责任。此外，如果方法部分地更改了x，然后又出错，则这些更改将影响调用方作用域中的x。删除或完全替换subassign中的x不会做任何事情，但是，只有索引赋值才重要。

由于这种优化可能会改变代码的工作方式(特别是如果编写得不好的话)，因此提供了一个函数optimize\_subsasgn\_calls来控制它。默认情况下启用该特性。另一种避免优化的方法是声明具有不同输出和输入参数的子分配方法，如下所示:

function y = subsasgn (x, ss, val)

…

endfunction

**34.4对象重载**

**34.4.1函数重载**

任何Octave函数都可以重载，这允许在需要时调用函数的特定对象版本。多项式类的一个相关示例可能是重载polyval函数。

function [y, dy] = polyval (p, varargin)

if (nargout > 1)

[y, dy] = polyval (fliplr (p.poly), varargin{:});

else

y = polyval (fliplr (p.poly), varargin{:});

endif

endfunction

这个函数只是把工作交给普通的八度多值函数。多项式类重载函数的另一个有趣的例子是plot函数。

function h = plot (p, varargin)

n = 128;

rmax = max (abs (roots (p.poly)));

x = [0 : (n - 1)] / (n - 1) \* 2.2 \* rmax - 1.1 \* rmax;

if (nargout > 0)

h = plot (x, polyval (p, x), varargin{:});

else

plot (x, polyval (p, x), varargin{:});

endif

endfunction

它允许多项式在多项式根区域附近的定义域内绘制。

对重载特别感兴趣的函数是类转换函数，如double。重载这些函数允许强制转换函数与用户类一起工作。它还可以帮助将类对象与来自其他类的方法和函数一起使用，因为对象可以转换为新函数所需的输入形式。多项式类的双函数示例如下

function a = double (p)

a = p.poly;

endfunction

**34.4.2操作符重载**

下表显示了为用户类提供重载方法时要使用的每个内置数值操作的对应函数名。

| **Operation** | **Method** | **Description** |
| --- | --- | --- |
| *a + b* | *plus (a, b)* | Binary addition |
| *a - b* | *minus (a, b)* | Binary subtraction |
| *+a* | *uplus (a)* | Unary addition |
| *-a* | *uminus (a)* | Unary subtraction |
| *a .\* b* | *times (a, b)* | Element-wise multiplication |
| *a \* b* | *mtimes (a, b)* | Matrix multiplication |
| *a ./ b* | *rdivide (a, b)* | Element-wise right division |
| *a / b* | *mrdivide (a, b)* | Matrix right division |
| *a .\ b* | *ldivide (a, b)* | Element-wise left division |
| *a \ b* | *mldivide (a, b)* | Matrix left division |
| *a .^ b* | *power (a, b)* | Element-wise power |
| *a ^ b* | *mpower (a, b)* | Matrix power |
| *a < b* | *lt (a, b)* | Less than |
| *a <= b* | *le (a, b)* | Less than or equal to |
| *a > b* | *gt (a, b)* | Greater than |
| *a >= b* | *ge (a, b)* | Greater than or equal to |
| *a == b* | *eq (a, b)* | Equal to |
| *a != b* | *ne (a, b)* | Not equal to |
| *a & b* | *and (a, b)* | Logical and |
| *a | b* | *or (a, b)* | Logical or |
| *!a* | *not (a)* | Logical not |
| *a'* | *ctranspose (a)* | Complex conjugate transpose |
| *a.'* | *transpose (a)* | Transpose |
| *a:b* | *colon (a, b)* | Two element range |
| *a:b:c* | *colon (a, b, c)* | Three element range |
| *[a, b]* | *horzcat (a, b)* | Horizontal concatenation |
| *[a; b]* | *vertcat (a, b)* | Vertical concatenation |
| *a(s\_1,…,s\_n)* | *subsref (a, s)* | Subscripted reference |
| *a(s\_1,…,s\_n) = b* | *subsasgn (a, s, b)* | Subscripted assignment |
| *b(a)* | *subsindex (a)* | Convert object to index |
| *disp* | *disp (a)* | Object display |

表34.1:可用的重载操作符及其对应的类方法

多项式类的mtimes方法示例如下

function p = mtimes (a, b)

p = polynomial (conv (double (a), double (b)));

endfunction

**34.4.3对象优先级**

许多函数和操作符接受两个或多个参数，很容易出现用不同类的对象调用这些函数的情况。因此，当给函数或操作符提供混合对象时，有必要确定从哪个类调用哪个方法的优先级。为此，可以使用superior和inferorto函数

: superiorto (class\_name, …)

从类构造函数调用时，将当前构造的对象标记为具有比class\_name更高的优先级。

在一次调用中可以指定多个这样的类。此函数只能从类构造函数调用。

参见:inferorto。

: inferiorto (class\_name, …)

当从类构造函数调用时，将当前构造的对象标记为具有低于class\_name的优先级。

在一次调用中可以指定多个这样的类。此函数只能从类构造函数调用。

参见:superiorto。

对于多项式类，考虑这种情况

2 \* polynomial ([1, 0, 1]);

这将类“double”的对象与类“多项式”的对象混合。在这种情况下，返回类型应该是“多项式”，因此在类构造函数中使用superiorto函数。特别是多项式类构造函数将被修改为

## -\*- texinfo -\*-

## @deftypefn {} {} polynomial ()

## @deftypefnx {} {} polynomial (@var{a})

## Create a polynomial object representing the polynomial

##

## @example

## a0 + a1 \* x + a2 \* x^2 + @dots{} + an \* x^n

## @end example

##

## @noindent

## from a vector of coefficients [a0 a1 a2 @dots{} an].

## @end deftypefn

function p = polynomial (a)

if (nargin == 0)

p.poly = [0];

p = class (p, "polynomial");

else

if (strcmp (class (a), "polynomial"))

p = a;

elseif (isreal (a) && isvector (a))

p.poly = a(:).'; # force row vector

p = class (p, "polynomial");

else

error ("@polynomial: A must be a real vector");

endif

endif

superiorto ("double");

endfunction

请注意，用户类总是比内置Octave类型具有更高的优先级。因此，实际上没有必要将多项式类标记为高于“double”类。

当遇到两个具有相同优先级的对象时，Octave将使用在参数列表中最先出现的对象的方法。

**34.5继承和聚合**

通过使用继承和聚合，Octave支持使用类来构建新类。

类继承由Octave使用类构造函数中的类函数提供。与多项式类的情况一样，Octave程序员将创建一个包含类所需数据字段的结构，然后调用类函数来指示要从该结构创建对象。创建现有对象的子对象是通过创建父类的对象并将该对象作为类函数的第三个参数提供来完成的。

这一点最容易用例子来证明。假设程序员需要一个FIR滤波器，即一个分子多项式但分母为1的滤波器。在传统的Octave编程中，这将按照以下方式执行。

>> x = [some data vector];

>> n = [some coefficient vector];

>> y = filter (n, 1, x);

等效的行为可以通过类@FIRfilter来实现。该类的构造函数是文件FIRfilter。m在类目录@FIRfilter中。

## -\*- texinfo -\*-

## @deftypefn {} {} FIRfilter ()

## @deftypefnx {} {} FIRfilter (@var{p})

## Create a FIR filter with polynomial @var{p} as coefficient vector.

## @end deftypefn

function f = FIRfilter (p)

if (nargin == 0)

p = @polynomial ([1]);

elseif (! isa (p, "polynomial"))

error ("@FIRfilter: P must be a polynomial object");

endif

f.polynomial = [];

f = class (f, "FIRfilter", p);

endfunction

和前面一样，前面的注释为类构造函数提供了文档。这个构造函数与多项式类构造函数非常相似，除了一个多项式对象作为类函数的第三个参数传递，告诉Octave FIRfilter类将从多项式类派生。FIR过滤器类本身没有任何数据字段，但是它必须为类函数提供一个结构体。假设@多项式构造函数将向对象结构中添加一个名为多项式的元素，那么@FIRfilter只是初始化一个结构，其中包含一个稍后将被覆盖的虚拟字段多项式。

请注意，示例代码总是为不提供参数的情况提供支持。这一点很重要，因为Octave在从保存的文件加载对象时将调用一个不带参数的构造函数，以确定继承结构。

一个类可以是多个类的子类(参见类)，继承可以是嵌套的。除了内存或其他物理问题外，对父节点的数量或嵌套级别没有限制。

对于FIRfilter类，需要对对象显示进行更多的控制。因此，重载的是display方法而不是disp方法(参见类方法)。一个简单的例子可能是

function display (f)

printf ("%s.polynomial", inputname (1));

disp (f.polynomial);

endfunction

请注意，FIRfilter的显示方法依赖于多项式类中的disp方法来实际显示过滤器系数。此外，请注意，在display方法中，以printf ("%s ="， inputname(1))行开始方法是有意义的，它与Octave的其余部分保持一致，Octave打印要显示的变量名后跟值。一般不建议将显示功能过载。

: display (obj)

显示以其名称前缀的对象obj的内容。

每当需要在屏幕上显示一个类时，Octave解释器就调用display函数。通常，这将是一个不以分号结束以抑制输出的语句。例如:

myclass (…)

或者：

myobj = myclass (…)

通常，用户定义的类应该重载disp方法以避免默认输出:

myobj = myclass (…)

⇒ myobj =

<class myclass>

当重载display方法时，必须注意正确显示对象的名称。这可以通过使用inputname函数来完成。

参见:disp, class, subsref, subsassign。

一旦构造函数和显示方法存在，就可以创建类的实例。还可以检查类类型和检查底层结构。

octave:1> f = FIRfilter (polynomial ([1 1 1]/3))

f.polynomial = 0.33333 + 0.33333 \* X + 0.33333 \* X ^ 2

octave:2> class (f)

ans = FIRfilter

octave:3> isa (f, "FIRfilter")

ans = 1

octave:4> isa (f, "polynomial")

ans = 1

octave:5> struct (f)

ans =

scalar structure containing the fields:

polynomial = 0.33333 + 0.33333 \* X + 0.33333 \* X ^ 2

使这个类可用的唯一剩下的东西是处理数据的方法。但在此之前，通常还需要有一种方法来更改存储在类中的数据。由于底层结构中的字段在默认情况下是私有的，因此有必要提供一种访问字段的机制。subsref方法可以用于这两个任务。

function r = subsref (f, x)

switch (x.type)

case "()"

n = f.polynomial;

r = filter (n.poly, 1, x.subs{1});

case "."

fld = x.subs;

if (! strcmp (fld, "polynomial"))

error ('@FIRfilter/subsref: invalid property "%s"', fld);

endif

r = f.polynomial;

otherwise

error ("@FIRfilter/subsref: invalid subscript type for FIR filter");

endswitch

endfunction

"()"的情况允许我们使用提供给构造函数的多项式来过滤数据。

octave:2> f = FIRfilter (polynomial ([1 1 1]/3));

octave:3> x = ones (5,1);

octave:4> y = f(x)

y =

0.33333

0.66667

1.00000

1.00000

1.00000

“。”的情况允许我们查看多项式字段的内容。

octave:1> f = FIRfilter (polynomial ([1 1 1]/3));

octave:2> f.polynomial

ans = 0.33333 + 0.33333 \* X + 0.33333 \* X ^ 2

为了更改对象的内容，需要使用子分配方法。例如，下面的代码使多项式字段可公开写入

function fout = subsasgn (f, index, val)

switch (index.type)

case "."

fld = index.subs;

if (! strcmp (fld, "polynomial"))

error ('@FIRfilter/subsasgn: invalid property "%s"', fld);

endif

fout = f;

fout.polynomial = val;

otherwise

error ("@FIRfilter/subsasgn: Invalid index type")

endswitch

endfunction

以便

octave:1> f = FIRfilter ();

octave:2> f.polynomial = polynomial ([1 2 3])

f.polynomial = 1 + 2 \* X + 3 \* X ^ 2

将FIRfilter类定义为多项式类的子类意味着FIRfilter对象可以在任何可能使用多项式对象的地方使用。这不是过滤器的正常用法。使用聚合而不是继承可能是更明智的设计方法。在这种情况下，多项式只是类结构中的一个字段。聚合情况的类构造函数可以是

## -\*- texinfo -\*-

## @deftypefn {} {} FIRfilter ()

## @deftypefnx {} {} FIRfilter (@var{p})

## Create a FIR filter with polynomial @var{p} as coefficient vector.

## @end deftypefn

function f = FIRfilter (p)

if (nargin == 0)

f.polynomial = @polynomial ([1]);

else

if (! isa (p, "polynomial"))

error ("@FIRfilter: P must be a polynomial object");

endif

f.polynomial = p;

endif

f = class (f, "FIRfilter");

endfunction

对于本例，只需要更改构造函数，其他所有类方法保持不变。

**34.6 classdef类**

从4.0版本开始，Octave对classdef类的支持有限。与前面提到的类(本节称为旧式类)不同，classdef类可以在单个m文件中定义。classdef类的其他创新有:

* access rights for properties and methods,
* static methods, i.e., methods that are independent of an object, and
* the distinction between value and handle classes.

为了与MATLAB完全兼容，Octave的未来版本必须添加几个特性。关于缺失内容的概述可以在https://wiki.octave.org/Classdef上找到。

**34.6.1创建classdef Class**

一个非常基本的classdef value类(参见值类与句柄类)由以下方式定义:

classdef some\_class

properties

endproperties

methods

endmethods

endclassdef

与旧式的类相比，properties-endproperties块以及methods-endmethods块可用于定义类的属性和方法。因为两个块都是空的，所以在这种特殊情况下可以省略它们。

为简单起见，再次使用多项式示例展示了classdef类的更高级实现(参见创建类):

classdef polynomial2

properties

poly = 0;

endproperties

methods

function p = polynomial2 (a)

if (nargin == 1)

if (isa (a, "polynomial2"))

p.poly = a.poly;

elseif (isreal (a) && isvector (a))

p.poly = a(:).'; # force row vector

else

error ("polynomial2: A must be a real vector");

endif

endif

endfunction

function disp (p)

a = p.poly;

first = true;

for i = 1 : length (a);

if (a(i) != 0)

if (first)

first = false;

elseif (a(i) > 0 || isnan (a(i)))

printf (" +");

endif

if (a(i) < 0)

printf (" -");

endif

if (i == 1)

printf (" %.5g", abs (a(i)));

elseif (abs (a(i)) != 1)

printf (" %.5g \*", abs (a(i)));

endif

if (i > 1)

printf (" X");

endif

if (i > 2)

printf (" ^ %d", i - 1);

endif

endif

endfor

if (first)

printf (" 0");

endif

printf ("\n");

endfunction

endmethods

endclassdef

通过调用类构造函数来创建类多项式2的对象:

>> p = polynomial2 ([1, 0, 1])

⇒ p =

1 + X ^ 2

**34.6.2属性**

所有的类属性都必须在属性块中定义。属性默认值的定义是可选的，可以省略。每个类属性的默认初始值是[]。

属性块可以有额外的属性来指定访问权限或定义常量:

classdef some\_class

properties (Access = mode)

prop1

endproperties

properties (SetAccess = mode, GetAccess = mode)

prop2

endproperties

properties (Constant = true)

prop3 = pi ()

endproperties

properties

prop4 = 1337

endproperties

endclassdef

模式可以是其中之一:

public

The properties can be accessed from everywhere.

private

The properties can only be accessed from class methods. Subclasses of that class cannot access them

protected

The properties can only be accessed from class methods and from subclasses of that class.

在创建some\_class对象时，prop1具有默认值[]，并且对prop1的读写由单一模式定义。对于prop2，读写访问可以设置不同。最后，prop3是一个常量属性，只能在属性块中初始化一次。

默认情况下，在示例prop4中，属性不是常量，并且具有公共读写访问权限。

: properties (obj)

: properties (class\_name)

: proplist = properties (…)

显示或返回classdef object obj或命名类class\_name的公共属性。

如果请求输出值，则返回单元格数组中的属性名称列表。

编程注意:如果GetAccess属性是公共的，并且如果Hidden属性为false，则返回属性名。

参见:方法。

**34.6.3方法**

所有类方法必须在方法块中定义。在本小节的末尾描述了该规则的一个例外。这些方法块可以有额外的属性来指定访问权限或方法是否为静态的，即不需要创建该类的对象就可以调用的方法。

classdef some\_class

methods

function obj = some\_class ()

disp ("New instance created.");

endfunction

function disp (obj)

disp ("Here is some\_class.");

endfunction

endmethods

methods (Access = mode)

function r = func (obj, r)

r = 2 \* r;

endfunction

endmethods

methods (Static = true)

function c = circumference (radius)

c = 2 \* pi () .\* radius;

endfunction

endmethods

endclassdef

类的构造函数在方法块中声明，必须具有与类相同的名称，并且只有一个输出参数，该参数是其类的对象。

也可以重载内置或继承的方法，就像上面例子中的disp函数告诉Octave应该如何显示some\_class的对象(参见类方法)。

通常，方法定义中的第一个参数总是调用它的对象。类方法可以通过将对象作为该方法的第一个参数传递来调用，也可以通过在调用对象后加上点(".")和方法名以及随后的参数来调用:

>> obj = some\_class ();

New instance created.

>> disp (obj); # both are

>> obj.disp (); # equal

在some\_class中，方法func是在一个方法块中定义的，将Access属性设置为mode，它是:

public

The methods can be accessed from everywhere.

private

The methods can only be accessed from other class methods. Subclasses of that class cannot access them.

protected

The methods can only be accessed from other class methods and from subclasses of that class.

方法的默认访问是公共的。

最后，方法圆周在静态方法块中定义，无需创建some\_class对象即可使用。这对于不依赖于任何类属性的方法很有用。类名和静态方法的名称(用点(".")分隔)调用这个静态方法。与非静态方法相比，即使使用some\_class的对象调用，对象也不会作为第一个参数传递。

>> some\_class.circumference (3)

⇒ ans = 18.850

>> obj = some\_class ();

New instance created.

>> obj.circumference (3)

⇒ ans = 18.850

此外，类方法可以定义为与类同名的文件夹中的函数，类的前面加了“@”符号(参见创建类)。主classdef文件也必须存储在这个类文件夹中。

**34.6.4继承**

类可以从其他类继承。在这种情况下，超类的所有属性和方法都被继承到子类，考虑到它们的访问权限。使用这个语法来继承超类:

classdef subclass < superclass

…

endclassdef

**34.6.5值类与句柄类**

classdef类有两种本质上不同的类型，它们的主要区别在于变量赋值的行为。第一类是值类:

classdef value\_class

properties

prop1

endproperties

methods

function obj = set\_prop1 (obj, val)

obj.prop1 = val;

endfunction

endmethods

endclassdef

将该类的对象赋值给另一个变量实际上创建了一个新对象:

>> a = value\_class ();

>> a.prop1 = 1;

>> b = a;

>> b.prop1 = 2;

>> b.prop1

⇒ ans = 2

>> a.prop1

⇒ ans = 1

但这也意味着你可能不得不将一个方法的输出手动赋值回对象:

>> a = value\_class ();

>> a.prop1 = 1;

>> a.set\_prop1 (3);

⇒ ans =

<object value\_class>

>> ans.prop1

⇒ ans = 3

>> a.prop1

⇒ ans = 1

第二种类型是句柄类。这些类必须从抽象句柄类派生:

classdef handle\_class < handle

properties

prop1

endproperties

methods

function set\_prop1 (obj, val)

obj.prop1 = val;

endfunction

endmethods

endclassdef

在下面的例子中，变量a和b引用了类handle\_class的同一个对象:

>> a = handle\_class ();

>> a.prop1 = 1;

>> b = a;

>> b.prop1 = 2;

>> b.prop1

⇒ ans = 2

>> a.prop1

⇒ ans = 2

由句柄类的方法修改的对象属性将被持久地更改:

>> a.set\_prop1 (3);

>> a.prop1

⇒ ans = 3

**35 GUI开发**

Octave主要是一种批处理或命令行语言。但是，它确实提供了一些用于构建与用户交互的图形界面的特性。

可用的GUI元素包括I/O对话框、进度条和绘图窗口的UI元素。例如，脚本可以打开一个对话框，允许用户选择一个文件，而不是为输出结果硬编码一个文件名。类似地，如果预计计算需要很长时间，脚本可以显示进度条。各种UI元素可用于完全定制具有菜单栏、工具栏、上下文菜单、按钮、滑块等的绘图窗口。

有几个实用程序函数可以存储私有数据以供GUI使用，而不会污染用户的变量空间。

最后，用Octave编写的程序可能需要长期存储首选项或状态变量。这可以通过用户定义的首选项来实现。

**35.1 I/O对话框**

简单的对话框菜单可用于选择目录或文件。它们返回一个字符串变量，可以用于任何需要文件名的命令。

: dirname = uigetdir ()

: dirname = uigetdir (init\_path)

: dirname = uigetdir (init\_path, dialog\_name)

打开一个GUI对话框来选择一个目录。

如果未指定init\_path，则使用当前工作目录。

Dialog\_name可用于自定义对话框标题。

输出的dirname是一个字符串，包含所选目录的名称。但是，如果单击“Cancel”按钮，则输出类型为double，值为0。

参见:uietfile, uiputfile。

: [fname, fpath, fltidx] = uigetfile ()

: […] = uigetfile (flt)

: […] = uigetfile (flt, dialog\_name)

: […] = uigetfile (flt, dialog\_name, default\_file)

: […] = uigetfile (…, "MultiSelect", mode)

打开一个用于选择文件的GUI对话框，并返回文件名fname、该文件的路径fpath和过滤器索引fltidx。

FLT包含以下格式之一的文件过滤器字符串(列表):

"/path/to/filename.ext"

如果给出了文件名，则提取文件扩展名并将其用作过滤器。此外，在对话框中路径被选择为当前路径，文件名被选择为默认文件。示例:uigetfile ("myfcn.m")

单个文件扩展名“\*.ext”

示例:uigetfile ("\*.ext")

一个两列单元格数组

在第一列中包含文件扩展名，在第二列中包含简要描述。示例:uigetfile({"\*。ext”，“我的描述”;xyz”、“XYZ-Format”})

过滤器字符串还可以包含一个以分号分隔的过滤器扩展列表。示例:uigetfile ({"\*.gif;\*.png;\*.jpg"， "支持的图片格式"})

目录名或路径名

如果path name的文件夹名包含末尾的文件分隔符，则将显示该文件夹的内容。如果没有尾随文件分隔符，则列出父目录。最右边的文件分隔符(如果有的话)右边的子字符串将被解释为文件或目录名，如果该文件或目录存在，它将被突出显示。如果路径名或目录名完全或部分不存在，则显示当前工作目录。没有过滤器将被激活。

Dialog\_name可用于自定义对话框标题。

如果给出default\_file，那么它将在GUI对话框中被选中。另外，如果给定了一个路径，它也被用作当前路径。

当将“MultiSelect”键设置为“on”时，可以选择两个或多个文件。在这种情况下，fname是一个包含文件的单元格数组。

输出fname和fpath分别是返回所选名称和路径的字符串。但是，如果单击“Cancel”按钮，则输出类型为double，值为0。Fltidx是所选过滤器扩展FLT列表中的索引。

参见:uiputfile, uigetdir。

: [fname, fpath, fltidx] = uiputfile ()

: [fname, fpath, fltidx] = uiputfile (flt)

: [fname, fpath, fltidx] = uiputfile (flt, dialog\_name)

: [fname, fpath, fltidx] = uiputfile (flt, dialog\_name, default\_file)

打开一个GUI对话框来选择文件。

FLT包含以下格式之一的文件过滤器字符串(列表):

"/path/to/filename.ext"

如果给定了文件名，则提取文件扩展名并将其用作过滤器。此外，对话框中的路径被选择为当前路径，文件名被选择为默认文件。示例:uiputfile ("myfcn.m")

"\*.ext"

单个文件扩展名。示例:uiputfile ("\*.ext")

{"\*.ext", "My Description"}

一个两列单元格数组，在第一列中包含文件扩展名，在第二列中包含简要描述。示例:uiputfile({"\*。ext”、“我的”;“\*描述。xyz”、“XYZ-Format”})

过滤器字符串还可以包含一个以分号分隔的过滤器扩展列表。示例:uiputfile ({"\*.gif;\*.png;\*.jpg"， "支持的图片格式"})

Dialog\_name可用于自定义对话框标题。如果给出default\_file，则在GUI对话框中预先选择它。另外，如果给定了一个路径，它也被用作当前路径。

Fname和fpath分别返回所选的名称和路径。Fltidx是所选过滤器扩展FLT列表中的索引。

参见:uigetfile, uigetdir。

此外，还有用于显示帮助消息、警告或错误的对话框，以及用于从用户获取文本输入的对话框。

: errordlg ()

: errordlg (msg)

: errordlg (msg, title)

: errordlg (msg, title, opt)

: h = errordlg (…)

显示带有错误消息msg和标题的错误对话框。

默认的错误信息是“这是默认的错误字符串”，默认的标题是“错误对话框”。

错误消息可能有多行，由换行字符(“\n”)分隔，或者它可能是每行一个元素的cellstr数组。

第三个可选参数opt控制对话框的行为。详细信息请参见信息提示框。

返回值h是用于构建对话框的图形对象的句柄。

例子:

errordlg ("Some fancy error occurred.");

errordlg ("Some fancy error\nwith two lines.");

errordlg ({"Some fancy error", "with two lines."});

errordlg ("Some fancy error occurred.", "Fancy caption");

参见:helpdlg, wardlg, msgbox, inputdlg, listdlg, questdlg。

: helpdlg ()

: helpdlg (msg)

: helpdlg (msg, title)

: h = helpdlg (…)

显示帮助对话框，显示帮助信息和标题。

默认的帮助信息是“这是默认的帮助字符串”，默认的标题是“帮助对话框”。

帮助消息可能有多行，用换行符(“\n”)分隔，或者它可能是每行有一个元素的cellstr数组。

返回值h是用于构建对话框的图形对象的句柄。

例子:

helpdlg ("Some helpful text for the user.");

helpdlg ("Some helpful text\nwith two lines.");

helpdlg ({"Some helpful text", "with two lines."});

helpdlg ("Some helpful text for the user.", "Fancy caption");

另见:errorlg, warndlg, msgbox, inputdlg, listdlg, questdlg。

: cstr = inputdlg (prompt)

: cstr = inputdlg (prompt, title)

: cstr = inputdlg (prompt, title, rowscols)

: cstr = inputdlg (prompt, title, rowscols, defaults)

: cstr = inputdlg (prompt, title, rowscols, defaults, options)

从多文本字段对话框中返回用户输入的字符串单元格数组，如果对话框被取消按钮关闭，则返回空单元格数组。

Inputs:

prompt

一个单元格数组，每个文本字段都有字符串标记。这个输入是必需的。

title

用于对话框标题的字符串。默认为“输入对话框”。

rowscols

指定文本字段的大小，可以采用三种形式:

定义用于每个文本字段的行数的标量值。

定义用于每个文本字段的单独行数的向量。

定义用于每个文本字段的单独行数和列数的矩阵。在矩阵中，每行描述一个文本字段。第一列指定要使用的输入行数，第二列指定文本字段宽度。

defaults

要放置在每个文本字段中的默认值列表。它必须是与提示符大小相同的字符串单元格数组。

options

不支持，仅用于MATLAB兼容性。

示例：

prompt = {"Width", "Height", "Depth"};

defaults = {"1.10", "2.20", "3.30"};

rowscols = [1,10; 2,20; 3,30];

dims = inputdlg (prompt, "Enter Box Dimensions", ...

rowscols, defaults);

参见:errorlg, helpdlg, listdlg, msgbox, questdlg, wardlg。

: [sel, ok] = listdlg (key, value, …)

从列表对话框返回一个选择索引向量(sel)的用户输入，并返回一个指示用户如何关闭对话框的标志(ok)。

sel中的索引是基于1的。

如果用户使用ok按钮关闭框，ok的值为1，否则为0,sel为空。

输入参数以键、值对的形式指定。必须指定"ListString"参数对。

有效的键和值对是:

"ListString"

a cell array of strings specifying the items to list in the dialog.

"SelectionMode"

can be either "Single" (only one item may be selected at a time) or "Multiple" (default).

"ListSize"

a two-element vector [width, height] specifying the size of the list field in pixels. The default is [160, 300].

"InitialValue"

a vector containing 1-based indices of elements which will be pre-selected when the list dialog is first displayed. The default is 1 (first item).

"Name"

a string to be used as the dialog caption. Default is "".

"PromptString"

a cell array of strings to be displayed above the list of items. Default is {}.

"OKString"

a string used to label the OK button. Default is "OK".

"CancelString"

a string used to label the Cancel button. Default is "Cancel".

示例：

my\_options = {"An item", "another", "yet another"};

[sel, ok] = listdlg ("ListString", my\_options,

"SelectionMode", "Multiple");

if (ok == 1)

disp ("You selected:");

for i = 1:numel (sel)

disp (sprintf ("\t%s", my\_options{sel(i)}));

endfor

else

disp ("You cancelled.");

endif

参见:菜单，错误，帮助，输入，msgbox，查询，提示。

: h = msgbox (msg)

: h = msgbox (msg, title)

: h = msgbox (msg, title, icon)

: h = msgbox (msg, title, "custom", cdata)

: h = msgbox (msg, title, "custom", cdata, colormap)

: h = msgbox (…, opt)

通过消息对话框显示消息。

消息可以有多行，用换行符(“\n”)分隔，也可以是每行有一个元素的cellstr数组。

可选的输入标题(字符串)可用于修饰对话框标题。

可选参数图标选择一个对话框图标。它可以是none(默认)、error、help、warn或custom中的一个。后者必须后跟一个图像数组cdata，对于已索引的图像，必须后跟相关的colormap。

最后一个可选参数opt控制对话框的行为。如果opt是一个字符串，它可能是

"non-modal" (default)

The dialog is normal.

"modal"

If any dialogs already exist with the same title, the most recent is reused and all others are closed. The dialog is displayed "modal" which means it prevents users from interacting with any other GUI element until the dialog has been closed.

"replace"

If any dialogs already exist with the same title, the most recent is reused and all others are closed. The resulting dialog is set "non-modal".

If opt is a structure, it must contain fields "WindowStyle" and "Interpreter":

"WindowStyle"

The value must be "non-modal", "modal", or "replace". See above.

"Interpreter"

Controls the "interpreter" property of the text object used for displaying the message. The value must be "tex" (default), "none", or "latex".

返回值h是用于构建对话框的图形对象的句柄。

例子:

msgbox ("Some message for the user.");

msgbox ("Some message\nwith two lines.");

msgbox ({"Some message", "with two lines."});

msgbox ("Some message for the user.", "Fancy caption");

## A message dialog box with error icon

msgbox ("Some message for the user.", "Fancy caption", "error");

参见:errdlg, helpdlg, inputdlg, listdlg, questdlg, wardlg。

: btn = questdlg (msg)

: btn = questdlg (msg, title)

: btn = questdlg (msg, title, default)

: btn = questdlg (msg, title, btn1, btn2, default)

: btn = questdlg (msg, title, btn1, btn2, btn3, default)

使用问题对话框显示msg，并返回激活按钮的标题。

消息可以有多行，用换行符(“\n”)分隔，也可以是每行有一个元素的cellstr数组。

可选的标题(字符串)可用于指定对话框标题。默认为“Question Dialog”。

对话框可能包含两个或三个按钮，它们都将关闭对话框。

字符串default标识默认按钮，该按钮通过按ENTER键激活。它必须匹配btn1、btn2或btn3中给定的字符串之一。

如果只指定msg和title，则使用三个默认标题为“是”、“否”和“取消”的按钮。

如果只指定了两个按钮标题btn1和btn2，则对话框将只包含这两个按钮。

例子:

btn = questdlg ("Close Octave?", "Some fancy title", ...

"Yes", "No", "No");

if (strcmp (btn, "Yes"))

exit ();

endif

参见:errorlg, helplg, inputdlg, listdlg, msgbox, warndlg。

: warndlg ()

: warndlg (msg)

: warndlg (msg, title)

: warndlg (msg, title, opt)

: h = warndlg (…)

显示一个带有警告信息msg和标题的警告对话框。

默认的警告信息是“这是默认的警告字符串”，默认的标题是“警告对话框”。

警告消息可能有多行，用换行符(“\n”)分隔，或者它可能是一个每行有一个元素的cellstr数组。

第三个可选参数opt控制对话框的行为。详细信息请参见信息提示框。

返回值h是用于构建对话框的图形对象的句柄。

例子:

warndlg ("Some warning text for the user.");

warndlg ("Some warning text\nwith two lines.");

warndlg ({"Some warning text", "with two lines."});

warndlg ("Some warning text for the user.", "Fancy caption");

参见:errorlg, helplg, msgbox, inputdlg, listdlg, questdlg。

: uisetfont ()

: uisetfont (h)

: uisetfont (fontstruct)

: uisetfont (…, title)

: fontstruct = uisetfont (…)

打开字体选择对话框。

如果第一个参数是文本、轴或uicontrol对象的句柄，按下OK按钮将改变对象的字体属性。

第一个参数也可以是一个结构体，包含字段FontName、FontWeight、FontAngle、FontUnits和FontSize，表示最初选择的字体。

对话框窗口的标题可以使用最后一个参数title来指定。

如果请求输出参数fontstruct，则返回所选字体结构。否则，在屏幕上显示字体信息。

编程注意:在不使用本机FontConfig的系统上(除了Linux)，字体缓存是在安装Octave时构建的。安装新字体后，您需要手动运行system(“fc-cache -fv”)。

参见:listfonts, text, axes, uicontrol。

为了创建新的对话框类型，有一个dialog函数。

: h = dialog ()

: h = dialog ("property", value, …)

创建一个空的模态对话框窗口，可以向其中添加其他控件。

该对话框是一个图形对象，具有对话框的推荐属性。

与图形不同的默认属性是:

buttondownfcn

if isempty (allchild(gcbf)), close (gcbf), endif

colormap

[]

color

defaultuicontrolbackgroundcolor

dockcontrols

off

handlevisibility

callback

integerhandle

off

inverthardcopy

off

menubar

none

numbertitle

off

paperpositionmode

auto

resize

off

windowstyle

modal

可以为对话框对象指定多个属性值对，但它们必须成对出现。完整的属性列表记录在图属性中。

返回值h是所创建图形的图形句柄。

例子:

## create an empty dialog window titled "Dialog Example"

h = dialog ("name", "Dialog Example");

## create a button (default style)

b = uicontrol (h, "string", "OK",

"position", [10 10 150 40],

"callback", "delete (gcf)");

## wait for dialog to resume or close

uiwait (h);

参见:错误，消息框，询问，警告，图形，等等。

**35.2进度条**

: h = waitbar (frac)

: h = waitbar (frac, msg)

: h = waitbar (…, "createcancelbtn", fcn, …)

: h = waitbar (…, prop, val, …)

: waitbar (frac)

: waitbar (frac, h)

: waitbar (frac, h, msg)

返回一个句柄h给一个新的进度指示器(“waitbar”)对象。

等待条被填充到分数分数，分数分数必须在[0,1]范围内。

可选消息msg居中并显示在等候栏上方。

可以使用等候栏图形的“createcancelbtn”属性将取消按钮添加到等候栏的底部。当用户按下按钮时要执行的动作使用字符串或函数句柄fcn指定。

等待栏图形窗口的外观可以通过向函数传递prop/val对来配置。完整的属性列表记录在图属性中。

当使用单个输入调用当前等待条时，如果它存在，则更新为新的值frac。如果有多个未完成的等待条，它们可以通过传递要修改的特定等待条的句柄h来单独更新。

参见:删除。

**35.3 UI元素**

ui\*系列函数在qt图形工具包中工作得最好，尽管有些功能在fltk工具包中可用。不支持gnuplot工具包。

: h = uifigure ()

: h = uifigure ("property", value, …)

为应用程序创建一个新的图形窗口。

可以为图形对象指定多个属性值对，但它们必须成对出现。

返回值h是创建的图形对象的图形句柄。

编程注意:完整的属性列表记录在图属性中。这个函数与图形的不同之处在于，所创建的图形是针对应用程序开发而不是绘图进行优化的。这意味着菜单栏和工具栏等功能被关闭。

参见:uipanel, uibuttongroup。

: hui = uipanel ()

: hui = uipanel (property, value, …)

: hui = uipanel (parent)

: hui = uipanel (parent, property, value, …)

创建一个uipanel对象。

ui面板用作容器来对其他ui控件对象进行分组。

如果省略parent，则为当前图形创建一个uipanel。如果没有可用的图，则首先创建一个新图。

如果给出了parent，则创建一个相对于parent的uipanel。

任何提供的属性值对将覆盖创建的uipanel对象的默认值。

完整的属性列表记录在Uipanel属性中。

可选的返回值hui是创建的uipanel对象的图形句柄。

例子:

## create figure and panel on it

f = figure;

p = uipanel ("title", "Panel Title", "position", [.25 .25 .5 .5]);

## add two buttons to the panel

b1 = uicontrol ("parent", p, "string", "A Button", ...

"position", [18 10 150 36]);

b2 = uicontrol ("parent", p, "string", "Another Button", ...

"position",[18 60 150 36]);

参见:图形，uicontrol。

: hui = uibuttongroup ()

: hui = uibuttongroup (property, value, …)

: hui = uibuttongroup (parent)

: hui = uibuttongroup (parent, property, value, …)

: uibuttongroup (h)

创建一个uibuttongroup对象并返回一个句柄。

uibuttongroup用于对uicontrol对象进行分组。

如果省略parent，则为当前图形创建一个uibuttongroup。如果没有可用的图，则首先创建一个新图。

如果parent被给定，则创建一个相对于parent的uibuttongroup。

任何提供的属性值对都将覆盖创建的uibuttongroup对象的默认值。

完整的属性列表记录在Uibuttongroup properties中。

例子:

## Create figure and panel on it

f = figure;

## Create a button group

gp = uibuttongroup (f, "Position", [ 0 0.5 1 1])

## Create a buttons in the group

b1 = uicontrol (gp, "style", "radiobutton", ...

"string", "Choice 1", ...

"Position", [ 10 150 100 50 ]);

b2 = uicontrol (gp, "style", "radiobutton", ...

"string", "Choice 2", ...

"Position", [ 10 50 100 30 ]);

## Create a button not in the group

b3 = uicontrol (f, "style", "radiobutton", ...

"string", "Not in the group", ...

"Position", [ 10 50 100 50 ]);

当使用单个参数h(一个现有uibuttongroup对象的句柄)调用时，将焦点切换到指定的uibuttongroup。这个功能目前还没有实现。

参见:figure, uipanel。

: hui = uicontrol ()

: hui = uicontrol (property, value, …)

: hui = uicontrol (parent)

: hui = uicontrol (parent, property, value, …)

: uicontrol (h)

创建一个uicontrol对象并返回一个句柄。

uicontrol对象用于创建简单的交互式控件，如按钮、复选框、编辑和列表控件。

如果省略parent，则为当前图形创建一个uicontrol。如果没有可用的图，则首先创建一个新图。

如果给出了parent，则创建一个相对于parent的uicontrol。

任何提供的属性值对都将覆盖创建的uicontrol对象的默认值。

完整的属性列表记录在Uicontrol properties中。

创建的uicontrol的类型由style属性指定。如果没有提供样式属性，则将创建一个按钮。

uicontrol的有效样式有:

"checkbox"

Create a checkbox control that allows user on/off selection.

"edit"

Create an edit control that allows user input of single or multiple lines of text.

"listbox"

Create a listbox control that displays a list of items and allows user selection of single or multiple items.

"popupmenu"

Create a popupmenu control that displays a list of options that can be selected when the user clicks on the control.

"pushbutton"

Create a push button control that allows user to press to cause an action.

"radiobutton"

Create a radio button control intended to be used for mutually exclusive input in a group of radiobutton controls.

"slider"

Create a slider control that allows user selection from a range of values by sliding knob on the control.

"text"

Create a static text control to display single or multiple lines of text.

"togglebutton"

Create a toggle button control that appears like a push button but allows the user to select between two states.

注意:对于“edit”和“listbox”样式，单行或多行/选择行为由“Min”和“Max”属性决定，当值设置为Max - Min > 1时，允许多行/选择。

示例：

## Create figure and panel on it

f = figure;

## Create a button (default style)

b1 = uicontrol (f, "string", "A Button", ...

"position", [10 10 150 40]);

## Create an edit control

e1 = uicontrol (f, "style", "edit", "string", "editable text", ...

"position", [10 60 300 40]);

## Create a checkbox

c1 = uicontrol (f, "style", "checkbox", "string", "a checkbox", ...

"position", [10 120 150 40]);

当使用单个参数h(一个现有uicontrol对象的句柄)调用时，将键盘焦点切换到指定的uicontrol。因此，uicontrol对象将接收可以使用“keypressfcn”回调处理的键盘事件。

参见:figure, uipanel。

: hui = uitable (property, value, …)

: hui = uitable (parent, property, value, …)

创建一个合适的对象并返回一个句柄。

合适的对象用于在图形窗口中显示数据表。

如果省略parent，则创建一个适合当前图形的对象。如果没有可用的图，则首先创建一个新图。

如果给出了parent，则创建一个适合于parent的相对对象。

任何提供的属性值对都将覆盖所创建的uitable对象的默认值。

属性的完整列表记录在Uitable properties中。

例子:

## Create figure and place a table on it

f = figure ();

m = magic (8);

t = uitable (f, "Data", m, "ColumnWidth", { 40 });

## Create a table with labeled rows and columns

f = figure ();

d = reshape (1:9, [3, 3]);

row\_names = { "Row1", "Row2", "Row3" };

col\_names = { "Col1", "Col2", "Col3" };

t = uitable (f, "Data", d, ...

"RowName", row\_names, "ColumnName", col\_names);

p = get (t, "Position");

e = get (t, "Extent");

p(3:4) = e(3:4);

set (t, "Position", p);

## Long demo with callbacks

function uitable\_demo ()

f = figure ("Name", "uitable Demo", "Menu", "none", ...

"Position", [10 10 1000 680]);

## A basic example

d = { "char" , "A string";

"double" , 12.3456789;

"complex", 1+2i;

"bool" , true;

"single" , single(12.3456789);

"int8" , int8(-128);

"uint8" , uint8(128);

"int16" , int16(-32768);

"uint16" , uint16(32768);

"int32" , int32(-2147483648);

"uint32" , uint32(2147483648);

"int64" , int64(-2147483649);

"uint64" , uint64(2147843649)};

popup\_options = {"A", "B", "C", "D", "E"};

columnformat\_options = { "[]", "char", "pop-up", "numeric", ...

"short", "short e", "short eng", ...

"short g", "long", "long e", ...

"long eng", "long g", "bank", "+", ...

"rat", "logical"};

columnformat\_values = columnformat\_options;

columnformat\_values{1} = "";

columnformat\_values{3} = popup\_options;

default\_data = repmat (d(:,2), 1, columns (columnformat\_options));

b\_add = uicontrol (f, "Position", [285 630 600 50], ...

"UserData", [rows(d), 1], ...

"Style", "pushbutton", ...

"String", "Set data at selected point to selected datatype");

l\_type\_table = uicontrol (f, "Position", [ 0 603 120 25 ], ...

"String", "Datatype Table:", ...

"Style", "text");

t\_type\_table = uitable (f, "Position", [ 0 530 1000 70 ], ...

"Data", transpose (d(:, 2)), ...

"ColumnName", transpose (d(:, 1)), ...

"RowName", "Value", ...

"CellSelectionCallback", ...

@(x, y) set (b\_add, "UserData", y.Indices ));

l\_point\_table = uicontrol (f, "Position", [ 0 640 60 25 ], ...

"String", "Point:", ...

"Style", "text");

t\_point\_table = uitable (f, "Position", [ 80 630 160 42 ], ...

"RowName", [], ...

"ColumnName", {"x", "y"}, ...

"Data", [ 1, 1 ], ...

"ColumnEditable", true);

l\_editable\_table = uicontrol (f, "Position", [ 0 502 200 25 ], ...

"Style", "text", ...

"String", "Set Data Columns Editable:");

t\_editable\_table = ...

uitable (f, "Position", [ 0 434 1000 65 ], ...

"Data", repmat (false, 1, columns (default\_data)), ...

"ColumnEditable", true);

l\_format\_table = uicontrol (f, "Position", [ 0 406 200 25 ], ...

"Style", "text", ...

"String", "Set Data Column Format:");

t\_format\_table = ...

uitable (f, "Position", [ 0 338 1000 65 ], ...

"Data", columnformat\_options, ...

"ColumnEditable", true, ...

"ColumnFormat", arrayfun (@(x) {columnformat\_options}, ...

1:columns (columnformat\_options)));

l\_data\_table = uicontrol (f, "Style", "text", ...

"String", "Data:", ...

"Position", [ 0 310 60 25 ]);

t\_data\_table = uitable (f, "Position", [ 0 15 1000 290 ], ...

"Data", default\_data, ...

"ColumnFormat", columnformat\_values);

set (t\_format\_table, ...

"CellEditCallback", ...

@(x, y) update\_column\_format (y.NewData, y.Indices, ...

t\_data\_table, popup\_options));

set (t\_point\_table, "CellEditCallback", ...

@(x, y) validate\_point\_table (x, y, t\_data\_table));

set (t\_editable\_table, "CellEditCallback", ...

@(x,y) set (t\_data\_table, ...

"ColumnEditable", get (t\_editable\_table, "Data")));

set (b\_add, ...

"Callback", @(x, y) update\_data (b\_add, t\_point\_table, ...

t\_type\_table, t\_data\_table));

set (t\_data\_table, "CellSelectionCallback", ...

@(x, y) update\_point\_table (y.Indices, t\_point\_table));

endfunction

function validate\_point\_table (h, dat, t\_data\_table)

if (! (dat.NewData > 0 && ...

dat.NewData < size (get (t\_data\_table, "Data"), dat.Indices(1, 1)) + 1))

d = get (h, "Data");

d(dat.Indices) = 1;

set (h, "Data", d);

endif

endfunction

function update\_column\_format (format, indices, t\_data\_table, ...

popup\_options)

cf = get (t\_data\_table, "ColumnFormat");

if (strcmp (format, "[]"))

format = "";

elseif (strcmp (format, "pop-up"))

format = popup\_options;

endif

cf{indices(1,2)} = format;

set (t\_data\_table, "ColumnFormat", cf);

endfunction

function update\_point\_table (indices, t\_point\_table)

if (isempty (indices))

indices = [1, 1];

endif

set (t\_point\_table, "Data", indices(1,:));

endfunction

function update\_data (b\_add, t\_point\_table, t\_type\_table, ...

t\_data\_table)

indices = get (b\_add, "UserData");

if (isempty (indices))

indices = [1, 1];

endif

d = get (t\_data\_table, "Data");

t\_type\_table\_data = get (t\_type\_table, "Data");

p = get (t\_point\_table, "Data");

d(p(1,2), p(1,1)) = t\_type\_table\_data(indices(1,2));

set (t\_data\_table, "Data", d);

endfunction

参见:图形，uicontrol。

: hui = uimenu (property, value, …)

: hui = uimenu (h, property, value, …)

创建一个uimenu对象并返回一个句柄。

如果省略h，则为当前图创建一个顶级菜单。如果给定h，则创建一个相对于h的子菜单。

Uimenu对象具有以下特定属性:

"accelerator"

A string containing the key, together with CTRL, to execute this menu entry (e.g., "x" for CTRL+x).

"checked"

Can be set "on" or "off". Sets a mark at this menu entry.

"enable"

Can be set "on" or "off". If disabled then the menu entry cannot be selected and is grayed out.

"foregroundcolor"

A color value for the text of the menu entry.

"menuselectedfcn"

The function called when this menu entry is executed. It can be either a function string (e.g., "myfcn"), a function handle (e.g., @myfcn) or a cell array containing the function handle and arguments for the callback function (e.g., {@myfcn, arg1, arg2}).

"position"

A scalar value containing the relative menu position. The first position has value 1 and will be either the left or top depending on the orientation of the uimenu.

"separator"

Can be set "on" or "off". If enabled, a separator line is drawn above the current position. This property is ignored for top-level entries.

"text"

A string containing the text for this menu entry. A "&"-symbol can be used to mark the "accelerator" character (e.g., "E&xit").

完整的属性列表记录在Uimenu properties中。

例子:

f = uimenu ("text", "&File", "accelerator", "f");

e = uimenu ("text", "&Edit", "accelerator", "e");

uimenu (f, "text", "Close", "accelerator", "q", ...

"menuselectedfcn", "close (gcf)");

uimenu (e, "text", "Toggle &Grid", "accelerator", "g", ...

"menuselectedfcn", "grid (gca)");

参见:figure。

: hui = uicontextmenu (property, value, …)

: hui = uicontextmenu (h, property, value, …)

创建一个uicontextmenu对象并返回一个句柄。

如果省略h，则为当前图形创建一个uicontextmenu。如果没有可用的图，则首先创建一个新图。

如果给定h，则创建一个相对于h的uicontextmenu。

任何提供的属性值对都将覆盖创建的uicontextmenu对象的默认值。

完整的属性列表记录在Uicontextmenu properties中。

例子:

## create figure and uicontextmenu

f = figure ();

c = uicontextmenu (f);

## create menus in the context menu

m1 = uimenu ("parent", c, "label", "Menu item 1", ...

"callback", "disp('menu item 1')");

m2 = uimenu ("parent", c, "label", "Menu item 2", ...

"callback", "disp('menu item 2')");

## set the context menu for the figure

set (f, "uicontextmenu", c);

参见:figure, ui菜单。

: hui = uitoolbar ()

: hui = uitoolbar (property, value, …)

: hui = uitoolbar (parent)

: hui = uitoolbar (parent, property, value, …)

创建工具栏对象。工具栏显示工具栏和工具栏按钮。

如果省略parent，则为当前图形创建一个工具栏。如果没有可用的图，则首先创建一个新图。

如果给出了parent，则创建一个相对于parent的工具栏。

任何提供的属性值对都将覆盖创建的工具栏对象的默认值。

完整的属性列表记录在Uitoolbar properties中。

可选的返回值hui是创建的工具栏对象的图形句柄。

例子:

% create figure without a default toolbar

f = figure ("toolbar", "none");

% create empty toolbar

t = uitoolbar (f);

参见:figure, uitoggletool, uipushtool。

: hui = uipushtool ()

: hui = uipushtool (property, value, …)

: hui = uipushtool (parent)

: hui = uipushtool (parent, property, value, …)

创建一个upushtool对象。

upushtools是出现在图形工具栏上的按钮。当用户将鼠标悬停在按钮上时，该按钮将显示边框。可以使用cdata属性设置图像。

如果省略parent，则为当前图形创建一个uppushtool。如果没有可用的图，则首先创建一个新图。如果有图形，但不包含工具栏，则创建工具栏。

如果指定了parent，则在父工具栏上创建一个upushtool。

任何提供的属性值对都将覆盖创建的uipushtool对象的默认值。

完整的属性列表记录在Uipushtool properties中。

可选的返回值hui是创建的uipushtool对象的图形句柄。

例子:

% create figure without a default toolbar

f = figure ("toolbar", "none");

% create empty toolbar

t = uitoolbar (f);

% create a 19x19x3 black square

img=zeros(19,19,3);

% add pushtool button to toolbar

b = uipushtool (t, "cdata", img);

参见:图形，工具栏，工具栏。

: hui = uitoggletool ()

: hui = uitoggletool (property, value, …)

: hui = uitoggletool (parent)

: hui = uitoggletool (parent, property, value, …)

创建一个uitoggletool对象。

工具栏是出现在图形工具栏上的切换按钮。当用户将鼠标悬停在按钮上时，该按钮将显示边框。可以使用cdata属性设置图像。

如果省略parent，则为当前图形创建一个工具。如果没有可用的图，则首先创建一个新图。如果有图形，但不包含工具栏，则创建工具栏。

如果指定了parent，则在父工具栏上创建一个工具栏。

任何提供的属性值对都将覆盖创建的uitoggletool对象的默认值。

完整的属性列表记录在Uitoggletool properties。

可选的返回值hui是创建的uitoggletool对象的图形句柄。

例子:

% create figure without a default toolbar

f = figure ("toolbar", "none");

% create empty toolbar

t = uitoolbar (f);

% create a 19x19x3 black square

img=zeros(19,19,3);

% add uitoggletool button to toolbar

b = uitoggletool (t, "cdata", img);

参见:图，工具栏，工具栏。

35.4 GUI实用功能

这些函数不实现GUI元素，但在开发实现GUI元素的程序时非常有用。函数uiwait、uiresume和waitfor只能在qt或fltk工具包中使用。

: data = guidata (h)

: guidata (h, data)

查询或设置用户自定义GUI数据。

GUI数据存储在图形句柄h中。如果h不是图形句柄，那么它的父图形将用于存储。

数据必须是单个对象，这意味着它通常是一个数据容器，如单元数组或结构体，这样可以很容易地添加额外的数据项。

参见:getappdata, setappdata, get, set, getpref, setpref。

: hdata = guihandles (h)

: hdata = guihandles

返回与句柄h相关联的图形的对象句柄结构。

如果没有指定句柄，则使用gcf返回的当前图形。

hdata的每个条目的字段名取自图形对象的“tag”属性。如果标记为空，则不返回句柄。如果有多个具有相同标记的图形对象，那么hdata中的条目将是句柄向量。guihandles包括所有可能的句柄，包括那些“HandleVisibility”为“off”的句柄。

参见:guidata, findobj, findall, allchild。

: tf = have\_window\_system ()

如果有可用的窗口系统(X11、Windows或Apple OS X)，返回true，否则返回false。

参见:isguirrunning。

: tf = isguirunning ()

如果Octave在GUI模式下运行返回true，否则返回false。

参见:have\_window\_system。

: pos = getpixelposition (h)

: pos = getpixelposition (h, rel\_to\_fig)

以像素为单位返回用户界面组件的位置。

第一个参数h必须是一个有效图形对象的句柄，类型为uibuttongroup、uicontrol、uipanel、uitable、axes或figure。对于其他对象类型，该函数返回零。

默认情况下，返回相对于对象父对象的位置。如果第二个参数rel\_to\_fig在逻辑上为真，则计算相对于封闭图形对象的位置。

返回值pos是一个值为[lower\_left\_X, lower\_left\_Y, width, height]的4元素向量。

参见:get。

: fonts = listfonts ()

: fonts = listfonts (h)

列出系统字体。

如果提供了图形对象h的句柄，也要在列表中包含该对象的“FontName”属性中的字体。

编程注意:在不使用本机FontConfig的系统上(除了Linux)，字体缓存是在安装Octave时构建的。安装新字体后，您需要手动运行system(“fc-cache -fv”)。

参见:uisetfont, text, axes, uicontrol。

: movegui

: movegui (h)

: movegui (pos)

: movegui (h, pos)

: movegui (h, event)

: movegui (h, event, pos)

将图形手柄h指定的图形移动到屏幕上由pos定义的位置。

H是图形句柄，或图形对象的句柄。在后一种情况下，将使用它的父图。如果未指定，如果正在执行回调(gcbf)，则h将被设置为相关图的句柄，否则将被设置为当前图的句柄(gcf)。

Pos可以是双值数字向量，也可以是字符串。如果pos是数字，那么它必须是[h, v]形式，指定图形相对于屏幕的水平和垂直偏移量。正值表示屏幕的左侧(垂直组件的底部)与图形的左侧(底部)之间的偏移量。负值表示屏幕的右侧(或顶部)与图形的右侧(或顶部)之间的偏移量。

pos作为字符串的可能值有

north

Top center of the screen.

south

Bottom center of the screen.

east

Right center of the screen.

west

Left center of the screen.

northeast

Top right of the screen.

northwest

Top left of the screen.

southeast

Bottom right of the screen.

southwest

Bottom left of the screen.

center

Center of the screen.

onscreen (default)

The figure will be minimally moved to be entirely visible on the screen, with a 30 pixel extra padding from the sides of the screen. This is the default value if none is provided.

Event包含将被忽略的事件数据。这个构造有助于从回调调用movegui。

: openvar (name)

在图形化变量编辑器中打开变量名。

: uiwait

: uiwait (h)

: uiwait (h, timeout)

挂起程序执行，直到带有句柄h的图形被删除或uiresume被调用。

当没有指定图形句柄时，此函数使用当前图形。如果图形句柄无效或当前没有图形，则此函数立即返回。

当指定时，timeout定义了等待数字删除或uresume调用的秒数。超时时间至少为1。如果指定的值较小，则发出警告，并使用超时值1代替。如果指定了一个非整数值，它将被截断为0。如果未指定timeout，则程序执行将被无限期挂起。

参见:简历，等待。

: uiresume (h)

用uiwait暂停程序执行。

句柄h必须与uiwait中指定的on相同。如果句柄无效，或者对于句柄为h的图形没有挂起的uiwait调用，则此函数不执行任何操作。

参见:uiwait。

: waitfor (h)

: waitfor (h, prop)

: waitfor (h, prop, value)

: waitfor (…, "timeout", timeout)

暂停当前程序的执行，直到满足图形句柄h上的条件。

当程序挂起时，图形事件仍然被正常处理，允许回调来修改图形对象的状态。这个函数是可重入的，可以从回调调用，而另一个等待调用在顶层挂起。

在第一种形式中，程序执行被挂起，直到图形对象h被销毁。如果图形句柄无效，或者h是根图形句柄，并且没有提供属性prop，则该函数立即返回。

在第二种形式中，执行被挂起，直到图形对象被销毁或名为prop的属性被修改。如果图形句柄无效或属性不存在，则该函数立即返回。

在第三种形式中，执行被暂停，直到图形对象被销毁或名为prop的属性被设置为value。isequal函数用于比较属性值。如果图形句柄无效、属性不存在或属性已设置为value，则该函数立即返回。

一个可选的超时可以使用属性"timeout"来指定。此超时值是等待条件为真所需的秒数。Timeout必须至少为1。如果指定的值较小，则发出警告并使用值1代替。如果超时值不是整数，则将其截断为0。

要在名为"timeout"的属性上定义条件，请使用字符串'\timeout'代替。

在所有情况下，键入CTRL-C会立即停止程序执行。

参见:waitforbuttonpress, isequal。

35.5自定义首选项

: val = getpref ("group", "pref")

: val = getpref ("group", "pref", default)

: {val1, val2, …} = getpref ("group", {"pref1", "pref2", …})

: prefstruct = getpref ("group")

: prefstruct = getpref ()

返回与首选项组group中的命名首选项pref相对应的首选项值。

命名首选项组必须是字符串。

如果组中不存在pref，并且指定了default，则创建值为default的首选项并返回default。

首选项pref可以是字符串或字符串的单元格数组。如果它是字符串的单元格数组，则返回首选项的单元格数组。

对应的默认值default可以是任何Octave值，例如。、double、struct、cell array、object等。或者，如果pref是字符串的单元格数组，那么default必须是与pref大小相同的值的单元格数组。

如果没有指定pref和default，则返回首选项组group的首选项结构。

如果未指定参数，则返回一个包含所有首选项组及其值的结构。

参见:addpref, setpref, ispref, rmpref。

: setpref ("group", "pref", val)

: setpref ("group", {"pref1", "pref2", …}, {val1, val2, …})

将首选项pref设置为指定首选项组group中的给定val。

命名首选项组必须是字符串。

首选项pref可以是字符串或字符串的单元格数组。

对应的值val可以是任何Octave值，例如。、double、struct、cell array、object等。或者，如果pref是字符串的单元格数组，那么val必须是与pref大小相同的值的单元格数组。

如果指定的首选项或组不存在，则添加它。

参见:addpref, getpref, ispref, rmpref。

: addpref ("group", "pref", val)

: addpref ("group", {"pref1", "pref2", …}, {val1, val2, …})

将首选项pref和关联值val添加到命名的首选项组group。

命名首选项组必须是字符串。

首选项pref可以是字符串或字符串的单元格数组。如果首选项已经存在，则会发出错误。

对应的值val可以是任何Octave值，例如。、double、struct、cell array、object等。或者，如果pref是字符串的单元格数组，那么val必须是与pref大小相同的值的单元格数组。

参见:setpref, getpref, ispref, rmpref。

: rmpref ("group", "pref")

: rmpref ("group", {"pref1", "pref2", …})

: rmpref ("group")

从首选项组group中删除命名首选项pref。

命名首选项组必须是字符串。

首选项pref可以是字符串或字符串的单元格数组。

如果不指定pref，则删除优先级组group。

删除不存在的首选项或组是错误的。

参见:addpref, ispref, setpref, getpref。

: tf = ispref ("group", "pref")

: tf = ispref ("group", {"pref1", "pref2", …})

: tf = ispref ("group")

如果指定的首选项存在于首选项组group中，则返回true。

命名首选项组必须是字符串。

首选项pref可以是字符串或字符串的单元格数组。

如果pref未指定，则如果存在首选项组group，则返回true。

参见:getpref, addpref, setpref, rmpref。

: dir = prefdir

: dir = prefdir (1)

返回保存Octave首选项的目录。

例子:

显示首选项目录

prefdir

更改到preferences文件夹

cd (prefdir)

如果带参数调用，如果preferences目录不存在，则创建该目录。

参见:getpref, setpref, addpref, rmpref, ispref。

: preferences

显示Octave的GUI首选项对话框窗口。

**35.6 Octave工作区窗口**

下面的函数使作为Octave自己的GUI界面一部分的窗口可见，并将键盘焦点移动到所选窗口。它们的实用性在于能够从脚本中调用这些函数并在不使用鼠标进行选择的情况下突出显示窗口。

: commandhistory ()

显示GUI命令历史窗口，并给它键盘焦点。

参见:命令窗口、文件浏览器、工作区。

: commandwindow ()

显示GUI命令窗口并给它键盘焦点。

参见:commandhistory, filebrowser, workspace。

: filebrowser ()

显示GUI文件浏览器窗口并给它键盘焦点。

参见:命令窗口、命令历史、工作空间。

: workspace ()

显示GUI工作区窗口并给它键盘焦点。

参见:commandwindow, commandhistory, filebrowser。

**36系统实用程序**

本章描述了可用的函数，这些函数允许您在Octave仍在运行时获取有关Octave外部发生的事情的信息，并在程序中使用这些信息。例如，您可以获取有关环境变量、当前时间的信息，甚至可以从Octave提示符启动其他程序。

**36.1定时实用程序**

Octave用于操作时间值的核心函数集是按照标准C库中相应函数的模式设计的。其中一些函数使用了包含以下元素的时间数据结构:

usec

Microseconds after the second (0-999999).

sec

Seconds after the minute (0-60). This number can be 60 to account for leap seconds.

min

Minutes after the hour (0-59).

hour

Hours since midnight (0-23).

mday

Day of the month (1-31).

mon

Months since January (0-11).

year

Years since 1900.

wday

Days since Sunday (0-6).

yday

Days since January 1 (0-365).

isdst

Daylight saving time flag.

gmtoff

Seconds offset from UTC.

zone

Time zone.

在以下函数的描述中，这个结构被称为tm\_struct。

: seconds = time ()

返回当前时间，作为自epoch以来的秒数。

纪元参照1970年1月1日00:00:00 UTC(协调世界时)。例如，在1997年2月17日星期一07:15:06 UTC, time返回的值是856163706。

参见:strftime, strptime, localtime, gmtime, mktime, now, date, clock, datenum, datestr, datevec, calendar, weekday。

: t = now ()

返回当前的本地日期/时间作为一个连续的天数(参见datenum)。

积分部分，楼层(现在)对应于从今天到1月10日之间的天数。

小数部分，rem(现在，1)对应于当前时间。

参见:时钟，日期，日期。

: str = ctime (t)

将time(或任何其他非负整数)返回的值转换为本地时间，并返回与asctime相同形式的字符串。

函数ctime (time)等价于asctime (localtime (time))。例如:

ctime (time ())

⇒ "Mon Feb 17 01:15:06 1997\n"

参见:asctime, time, localtime。

: tm\_struct = gmtime (t)

给定从time返回的值，或任何非负整数，返回一个与UTC(协调世界时)相对应的时间结构。

例如:

gmtime (time ())

⇒ {

usec = 0

sec = 6

min = 15

hour = 7

mday = 17

mon = 1

year = 97

wday = 1

yday = 47

isdst = 0

gmtoff = 0

zone = GMT

}

参见:strftime, strptime, localtime, mktime, time, now, date, clock, datenum, datestr, datevec, calendar, weekday。

: tm\_struct = localtime (t)

给定从time返回的值，或任何非负整数，返回与本地时区对应的时间结构。

localtime (time ())

⇒ {

usec = 0

sec = 6

min = 15

hour = 1

mday = 17

mon = 1

year = 97

wday = 1

yday = 47

isdst = 0

gmtoff = -21600

zone = CST

}

参见:strftime, strptime, gmtime, mktime, time, now, date, clock, datenum, datestr, datevec, calendar, weekday。

: seconds = mktime (tm\_struct)

将与本地时间相对应的时间结构转换为自epoch以来的秒数。

例如:

mktime (localtime (time ()))

⇒ 856163706

参见:strftime, strptime, localtime, gmtime, time, now, date, clock, datenum, datestr, datevec, calendar, weekday。

: str = asctime (tm\_struct)

将时间结构转换为使用以下格式的字符串:“ddd mmm mm HH:mm:SS yyyy \ n”。

例如:

asctime (localtime (time ()))

⇒ "Mon Feb 17 01:15:06 1997\n"

这相当于ctime (time())。

参见:ctime, localtime, time。

: str = strftime (fmt, tm\_struct)

使用格式字符串fmt以灵活的方式格式化时间结构tm\_struct，其中包含类似于printf中的' % '替换。

除特别说明外，替换字段具有固定大小;如有必要，将填充数字字段。默认情况下填充为零;对于显示单个数字的字段，可以通过在' % '后面加上下面描述的修饰符之一来更改或禁止填充。未知字段说明符作为普通字符复制。所有其他字符都不加更改地复制到输出中。例如:

strftime ("%r (%Z) %A %e %B %Y", localtime (time ()))

⇒ "01:15:06 AM (CST) Monday 17 February 1997"

Octave的strftime函数支持ANSI C字段说明符的超集。

文字字符字段:

%%

% character.

%n

Newline character.

%t

Tab character.

数字修饰符(非标准扩展):

- (dash)

Do not pad the field.

\_ (underscore)

Pad the field with spaces.

Time fields:

%H

Hour (00-23).

%I

Hour (01-12).

%k

Hour (0-23).

%l

Hour (1-12).

%M

Minute (00-59).

%p

Locale’s AM or PM.

%r

Time, 12-hour (hh:mm:ss [AP]M).

%R

Time, 24-hour (hh:mm).

%s

Time in seconds since 00:00:00, Jan 1, 1970 (a nonstandard extension).

%S

Second (00-61).

%T

Time, 24-hour (hh:mm:ss).

%X

Locale’s time representation (%H:%M:%S).

%z

Offset from UTC (±hhmm), or nothing if no time zone is determinable.

%Z

Time zone (EDT), or nothing if no time zone is determinable.

Date fields:

%a

Locale’s abbreviated weekday name (Sun-Sat).

%A

Locale’s full weekday name, variable length (Sunday-Saturday).

%b

Locale’s abbreviated month name (Jan-Dec).

%B

Locale’s full month name, variable length (January-December).

%c

Locale’s date and time (Sat Nov 04 12:02:33 EST 1989).

%C

Century (00-99).

%d

Day of month (01-31).

%e

Day of month ( 1-31).

%D

Date (mm/dd/yy).

%h

Same as %b.

%j

Day of year (001-366).

%m

Month (01-12).

%U

Week number of year with Sunday as first day of week (00-53).

%w

Day of week (0-6).

%W

Week number of year with Monday as first day of week (00-53).

%x

Locale’s date representation (mm/dd/yy).

%y

Last two digits of year (00-99).

%Y

Year (1970-).

参见:strptime, localtime, gmtime, mktime, time, now, date, clock, datenum, datestr, datevec, calendar, weekday。

: [tm\_struct, nchars] = strptime (str, fmt)

在格式字符串fmt的控制下，将字符串str转换为时间结构tm\_struct。

如果fmt匹配失败，nchars为0;否则，它将被设置为最后一个匹配字符的位置加1。总是检查这个，除非你绝对确定日期字符串将被正确解析。

参见:strftime, localtime, gmtime, mktime, time, now, date, clock, datenum, datestr, datevec, calendar, weekday。

本节中描述的其余大多数函数都不是按照标准C库的模式设计的。其中一些是为了与MATLAB兼容而提供的，而另一些则是因为它们很有用。

: datevec = clock ()

: [datevec, isdst] = clock ()

返回当前本地日期和时间作为日期向量。

日期向量包含以下字段:当前年、月(1-12)、日(1-31)、小时(0-23)、分钟(0-59)和秒(0-61)。秒字段在小数点后有一个小数部分，以提高精度。

如果系统的时区实行夏令时，则可选的第二个输出isdst为true。

例如:

fix (clock ())

⇒ 1993 8 20 4 56 1

时钟在具有gettimeofday功能的系统上更准确。

参见:now, date, datevec。

: str = date ()

以DD-MMM-YYYY格式的字符串形式返回当前日期。

例如:

date ()

⇒ 20-Aug-1993

参见:现在，时钟，日期，本地时间。

: secs = etime (t2, t1)

返回从时钟(t2 - t1)返回的两个时间值之间的秒差。

例如:

t0 = clock ();

# many computations later…

elapsed\_time = etime (clock (), t0);

将变量elapsed\_time设置为自设置变量t0以来的秒数。

参见:tic, toc, clock, cputime, addtodate。

: [total, user, system] = cputime ();

返回你的Octave会话使用的CPU时间。

第一个输出是执行进程所花费的总时间，它等于第二个和第三个输出的总和，这两个输出分别是在用户模式下执行所花费的CPU秒数和在系统模式下执行所花费的CPU秒数。

如果您的系统没有报告CPU时间使用情况的方法，那么cputime将为它的每个输出值返回0。

请注意，由于Octave使用了一些CPU时间来启动，因此通过检查所使用的总CPU时间是否为非零来检查cputime是否正常工作是合理的。

参见:tic, toc。

: tf = is\_leap\_year ()

: tf = is\_leap\_year (year)

如果year是闰年返回true，否则返回false。

如果不指定年份，is\_leap\_year使用当前年份。

例如:

is\_leap\_year (2000)

⇒ 1

参见:工作日，eomday，日历。

: tic ()

: id = tic ()

初始化一个挂钟计时器。

不带输出参数调用tic将重置内部计时器。对toc的后续调用返回自计时器设置以来的秒数。

如果使用一个输出参数调用，tic将创建一个新的计时器实例并返回一个计时器标识符id。id是一个类型为uint64的标量，可以传递给toc来检查该定时器的运行时间，而不是默认的内部定时器。

示例1:使用内部计时器对代码进行基准测试

tic;

# many computations later…

elapsed\_time = toc;

例2:混合定时器id和内部定时器

tic;

pause (1);

toc

⇒ Elapsed time is 1.0089 seconds.

id = tic;

pause (2);

toc (id)

⇒ Elapsed time is 2.01142 seconds.

toc

Elapsed time is 3.02308 seconds.

以这种方式调用tic和toc允许嵌套计时调用。

如果您对进程使用的CPU时间更感兴趣，则应该使用cputime函数。tic和toc函数报告调用之间经过的实际挂钟时间。这可能包括花在处理其他工作或什么都不做上的时间。

参见:toc, cputime。

: toc ()

: toc (id)

: elapsed\_time = toc (…)

测量挂钟计时器上经过的时间。

在不带参数的情况下，返回自上次调用tic以来内部计时器经过的秒数。

当给定特定计时器的标识符id时，返回自计时器id初始化以来经过的秒数。

参见tic，例如tic/toc的用法。

参见tic, cputime。

: pause ()

: pause (n)

: old\_state = pause ("on")

: old\_state = pause ("off")

: old\_state = pause ("query")

暂停程序的执行或改变暂停函数的状态。

如果在没有输入参数的情况下调用，则程序将挂起，直到输入一个字符。如果参数n为正实数，则表示程序应暂停的秒数，例如:

tic; pause (0.05); toc

-| Elapsed time is 0.05039 seconds.

下面的示例打印一个消息,然后等待5秒,然后清除屏幕。

disp ("wait please...");

pause (5);

clc;

如果使用字符串参数“on”、“off”或“query”调用暂停函数，则会更改或查询暂停函数的状态。当状态为“off”时，暂停函数立即返回。可选的返回值包含暂停函数的先前状态。在下面的示例中，本地禁用pause:

old\_state = pause ("off");

tic; pause (0.05); toc

-| Elapsed time is 3.00407e-05 seconds.

pause (old\_state);

当程序挂起时，Octave仍然处理图形绘画和图形回调执行。

参见:kbhit。

: days = datenum (datevec)

: days = datenum (year, month, day)

: days = datenum (year, month, day, hour)

: days = datenum (year, month, day, hour, minute)

: days = datenum (year, month, day, hour, minute, second)

: days = datenum ("datestr")

: days = datenum ("datestr", f)

: days = datenum ("datestr", p)

: [days, secs] = datenum (…)

将日期/时间输入返回为一个连续的日期号，将Jan 10000定义为第1天。

整数部分，floor (days)计算日期输入中完整的天数。

小数部分rem (days, 1)对应于给定日期的时间。

输入可以是日期向量(见datevec)、日期字符串(见datestr)，或者直接指定为输入。

在处理输入日期字符串时，f是用于解释日期字符串的格式字符串(参见datestr)。如果没有指定格式f，则通过各种格式执行相对较慢的搜索。如果格式字符串f已知，最好指定它。没有指定特定时间分量的格式将其值设置为零。未指定日期的格式将默认为当前年份的1月1日。

当传递单独的年、月、日等参数时，每个参数可以是标量或非标量数组。非标量输入必须具有相同的大小。标量输入将被扩展为非标量输入的大小。

P是本世纪开始的年份，两位数年份将被引用。如果未指定，则默认为当前年份减去50。

可选输出secs保存指定日期的时间，其精度高于days。

注:

年份可以是负数和/或小数。

低于1的月份被认为是1月。

每个月的天数从1开始。

过了月底的日子转到以后的月份。

月初的前几天到前一个月。

天可以是小数。

示例：

Convert from datestrs:

d = datenum ("1966-06-14")

⇒ d = 718232

d = datenum ({"1966-06-14", "1966-06-15", "1966-06-16"})

⇒ d =

718232

718233

718234

Convert from datevec:

d = datenum ([1966 06 14])

⇒ d = 718232

d = datenum ([1966 06 14 23 59 59])

⇒ d = 718232.9999884259

Specify date components separately:

d = datenum (1966, 6, 14)

⇒ d = 718232

d = datenum (1966, magic(3), 1)

⇒ d =

718280 718068 718219

718127 718188 718249

718158 718311 718099

注意:日基准代表整个地球的特定时间。它们没有考虑时区(基于位置的时间变化)，也没有考虑由于夏令时(基于当地法规的时间变化)而产生的季节变化。请注意，有可能创建的数据，在由一个考虑时区和DST变化的函数(如datestr)解释时，是不存在的或有歧义的。

注意:此函数不尝试处理儒略历，因此1582年10月15日之前的日期最多会有11天的误差。此外，要知道只有罗马天主教国家在1582年采用了这种日历。直到1924年，它才被广泛采用。请参阅维基百科关于公历的条目了解更多细节。

警告:闰秒被忽略。在维基百科的闰秒条目中有一个闰秒表。

参见:datestr, datevec, now, clock, date。

算法:Peter Baum (http://vsg.cape.com/~pbaum/date/date0.htm)

: str = datestr (date)

: str = datestr (date, f)

: str = datestr (date, f, p)

根据格式f格式化给定的日期/时间，并以str格式返回结果。

Date是一个序列号(参见datenum)，一个日期向量(参见datevec)，或者一个字符串或字符串的单元格数组。在后一种情况下，它被传递给datevec来猜测输入日期格式。

F可以是一个整数，对应于下表中的一个代码，也可以是一个日期格式字符串。

P是世纪开始的年份，两位数字的年份将被解释为年份。如果未指定，则默认为当前年份减去50。

例如，日期730736.65149(2000-09-07 15:38:09.0934)的格式如下:

| **Code** | **Format** | **Example** |
| --- | --- | --- |
| 0 | dd-mmm-yyyy HH:MM:SS | 07-Sep-2000 15:38:09 |
| 1 | dd-mmm-yyyy | 07-Sep-2000 |
| 2 | mm/dd/yy | 09/07/00 |
| 3 | mmm | Sep |
| 4 | m | S |
| 5 | mm | 09 |
| 6 | mm/dd | 09/07 |
| 7 | dd | 07 |
| 8 | ddd | Thu |
| 9 | d | T |
| 10 | yyyy | 2000 |
| 11 | yy | 00 |
| 12 | mmmyy | Sep00 |
| 13 | HH:MM:SS | 15:38:09 |
| 14 | HH:MM:SS PM | 3:38:09 PM |
| 15 | HH:MM | 15:38 |
| 16 | HH:MM PM | 3:38 PM |
| 17 | QQ-YY | Q3-00 |
| 18 | QQ | Q3 |
| 19 | dd/mm | 07/09 |
| 20 | dd/mm/yy | 07/09/00 |
| 21 | mmm.dd,yyyy HH:MM:SS | Sep.07,2000 15:38:08 |
| 22 | mmm.dd,yyyy | Sep.07,2000 |
| 23 | mm/dd/yyyy | 09/07/2000 |
| 24 | dd/mm/yyyy | 07/09/2000 |
| 25 | yy/mm/dd | 00/09/07 |
| 26 | yyyy/mm/dd | 2000/09/07 |
| 27 | QQ-YYYY | Q3-2000 |
| 28 | mmmyyyy | Sep2000 |
| 29 | yyyy-mm-dd | 2000-09-07 |
| 30 | yyyymmddTHHMMSS | 20000907T153808 |
| 31 | yyyy-mm-dd HH:MM:SS | 2000-09-07 15:38:08 |

如果f是格式字符串，则识别以下符号:

| **Symbol** | **Meaning** | **Example** |
| --- | --- | --- |
| yyyy | Full year | 2005 |
| yy | Two-digit year | 05 |
| mmmm | Full month name | December |
| mmm | Abbreviated month name | Dec |
| mm | Numeric month number (padded with zeros) | 01, 08, 12 |
| m | First letter of month name (capitalized) | D |
| dddd | Full weekday name | Sunday |
| ddd | Abbreviated weekday name | Sun |
| dd | Numeric day of month (padded with zeros) | 11 |
| d | First letter of weekday name (capitalized) | S |
| HH | Hour of day, padded with zeros, | 09:00 |
|  | or padded with spaces if PM is set | 9:00 AM |
| MM | Minute of hour (padded with zeros) | 10:05 |
| SS | Second of minute (padded with zeros) | 10:05:03 |
| FFF | Milliseconds of second (padded with zeros) | 10:05:03.012 |
| AM | Use 12-hour time format | 11:30 AM |
| PM | Use 12-hour time format | 11:30 PM |

如果f未指定或为-1，则根据date的日期部分或时间部分是否为空，使用0、1或16。

如果未指定p，则默认为当前年份减去50。

如果给出日期的矩阵或单元格数组，则返回日期字符串的列向量。

参见:datenum、datevec、date、now、clock。

: v = datevec (date)

: v = datevec (date, f)

: v = datevec (date, p)

: v = datevec (date, f, p)

: [y, m, d, h, mi, s] = datevec (…)

将串行日期数(参见datenum)或日期字符串(参见datestr)转换为日期向量。

日期向量是一个包含六个成员的行向量，分别表示年、月、日、时、分和秒。

日期数字输入可以是标量数组或非标量数组。日期字符串输入可以是单个日期字符串、二维日期字符数组(每行为一个日期字符串)或任何维度的单元格字符串数组(每个单元格元素包含一个日期字符串)。

V是日期向量的二维数组，每行一个日期向量。对于数组输入，v的排序基于data中日期的列主顺序。

F是用于解释日期字符串的格式字符串(参见datestr)。如果date是字符串或字符串的单元格数组，但未指定格式，则使用启发式方法猜测输入格式。这些启发式方法可能会导致与用户期望的结果不同的匹配。此外，这涉及到通过各种格式进行相对较慢的搜索。如果格式字符串f已知，最好指定它。没有指定特定时间分量的格式将其值设置为零。没有指定特定日期组件的格式将默认该组件为当前年份的1月1日。为了计算日期向量，即使字符包含额外的时间/日期信息，也会忽略尾随字符。

P是本世纪开始的年份，两位数年份将被引用。如果未指定，则默认为当前年份减去50。

参见:datenum,datestr,clock,now,date。

: d = addtodate (d, q, f)

将q个时间量(单位为f)添加到串行数据d中。

F必须是“年”、“月”、“日”、“时”、“分”、“秒”或“毫秒”中的一个。

参见:datenum, datevec, etime。

: c = calendar ()

: c = calendar (d)

: c = calendar (y, m)

: calendar (…)

以6x7矩阵的形式返回当前的月日历。

如果指定了d，则返回包含日期d的月份的日历，该日期d必须是一个序列号或日期字符串。

如果指定了y和m，则返回y年和m月的日历。

如果没有指定输出参数，则在屏幕上打印日历，而不是返回矩阵。

参见:datenum, datestr。

: [n, s] = weekday (d)

: [n, s] = weekday (d, format)

在n中以数字形式返回星期几，在s中以字符串形式返回星期几。

一星期的日子按1-7编号，第一天是星期天。

D是一个序列号或日期字符串。

如果字符串格式不存在或等于"short"，则s将包含工作日的缩写名称。如果format为"long"，则s将包含全名。

基于格式的返回值表:

| ***n*** | ***"short"*** | ***"long"*** |
| --- | --- | --- |
| 1 | Sun | Sunday |
| 2 | Mon | Monday |
| 3 | Tue | Tuesday |
| 4 | Wed | Wednesday |
| 5 | Thu | Thursday |
| 6 | Fri | Friday |
| 7 | Sat | Saturday |

参见:eomday, is\_leap\_year, calendar, datenum, datevec。

: e = eomday (y, m)

返回y年m月的最后一天。

参见:weekday, datenum, datevec, is\_leap\_year, calendar。

: datetick ()

: datetick (axis\_str)

: datetick (date\_format)

: datetick (axis\_str, date\_format)

: datetick (…, "keeplimits")

: datetick (…, "keepticks")

: datetick (hax, …)

向轴添加日期格式的刻度标签。

将刻度应用到的轴由axis\_str确定，它可以接受值“x”、“y”或“z”。默认值为“x”。

标签的格式由变量date\_format决定，date\_format可以是datestr接受的字符串或正整数。

如果第一个参数hax是一个轴句柄，那么绘制到这个轴，而不是gca返回的当前轴。

参见:datenum, datestr。

**36.2文件系统实用程序**

Octave包含许多用于复制、移动、重命名和删除文件的实用函数;用于创建、读取和删除目录;用于检索文件的状态信息;以及操作文件和路径名。

: movefile f1

: movefile f1 f2

: movefile f1 f2 f

: movefile (f1)

: movefile (f1, f2)

: movefile (f1, f2, 'f')

: [status] = movefile (…)

: [status, msg] = movefile (…)

: [status, msg, msgid] = movefile (…)

将源文件或目录f1移动到目标f2。

名称f1可以包含globing模式，也可以是字符串的单元格数组。如果f1展开为多个文件名，则f2必须是一个目录。

如果没有指定目标f2，则目标是当前工作目录。如果f2是文件名，则f1重命名为f2。

当强制标志'f'给定时，任何现有文件将被覆盖而不提示。

如果成功，status为逻辑1,msg、msgstr为空字符串("")。否则，status为逻辑0,msg包含系统相关的错误消息，msgstr包含唯一的消息标识符。请注意，状态码与系统命令的状态码完全相反。

参见:rename, copyfile, unlink, delete, glob。

: rename old new

: [status, msg] = rename (old, new)

将旧文件的名称改为新文件。

如果成功，status为0,msg为空字符串。否则，status为-1,msg包含与系统相关的错误消息。

参见:movefile, copyfile, ls, dir。

: copyfile f1 f2

: copyfile f1 f2 f

: copyfile (f1, f2)

: copyfile (f1, f2, 'f')

: [status, msg, msgid] = copyfile (…)

将源文件或目录f1复制到目标f2。

名称f1可以包含globing模式，也可以是字符串的单元格数组。如果f1展开为多个文件名，则f2必须是一个目录。

当强制标志'f'给定时，任何现有文件将被覆盖而不提示。

如果成功，status为逻辑1,msg、msgstr为空字符串("")。否则，status为逻辑0,msg包含系统相关的错误消息，msgstr包含唯一的消息标识符。请注意，状态码与系统命令的状态码完全相反。

参见:移动文件，重命名，取消链接，删除，glob。

: unlink (file)

: [status, msg] = unlink (file)

删除名为file的文件。

如果成功，status为0,msg为空字符串。否则，status为-1,msg包含与系统相关的错误消息。

请参见:delete, rmdir。

: link old new

: [status, msg] = link (old, new)

创建到现有文件的新链接(也称为硬链接)。

如果成功，status为0,msg为空字符串。否则，status为-1,msg包含与系统相关的错误消息。

参见:符号链接，解除链接，读链接，lstat。

:符号连接新旧

: [status, msg] = symlink(旧的，新的)

创建一个包含字符串old的符号链接new。

如果成功，status为0,msg为空字符串。否则，status为-1,msg包含与系统相关的错误消息。

参见:链接，取消链接，读链接，lstat。

: result = readlink symlink

: [result, err, msg] = readlink (symlink)

读取符号链接的值。

如果成功，result包含符号链接的内容，err为0,msg为空字符串。否则，err是非零的，并且msg包含一个系统相关的错误消息。

参见:lstat, symlink, link, unlink, delete。

: mkdir dirname

: mkdir parent dirname

: mkdir (dirname)

: mkdir (parent, dirname)

: [status, msg, msgid] = mkdir (…)

在父目录中创建一个名为dirname的目录，必要时创建任何中间目录。

如果dirname是一个相对路径，并且没有指定父目录，则使用当前工作目录。

如果成功，status为逻辑1,msg、msgstr为空字符串("")。否则，status为逻辑0,msg包含系统相关的错误消息，msgstr包含唯一的消息标识符。请注意，状态码与系统命令的状态码完全相反。

创建目录时将权限设置为0777 - UMASK。

参见:rmdir, pwd, cd, umask。

: rmdir dir

: rmdir (dir, "s")

: [status, msg, msgid] = rmdir (…)

删除名为dir的目录。

如果第二个可选参数提供的值为"s"，则也递归地删除所有子目录。

如果成功，status为逻辑1,msg、msgstr为空字符串("")。否则，status为逻辑0,msg包含系统相关的错误消息，msgstr包含唯一的消息标识符。

参见:mkdir, confirm\_recursive\_rmdir, pwd。

: val = confirm\_recursive\_rmdir ()

: old\_val = confirm\_recursive\_rmdir (new\_val)

: old\_val = confirm\_recursive\_rmdir (new\_val, "local")

查询或设置控制Octave在递归删除目录树之前是否要求确认的内部变量。

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。

请参见:rmdir。

: mkfifo (name, mode)

: [status, msg] = mkfifo (name, mode)

创建一个名为name的FIFO特殊文件，文件模式为mode。

Mode被解释为八进制数，并受到掩码处理的影响。最终计算模式为mode - umask。

如果成功，status为0,msg为空字符串。否则，status为-1,msg包含与系统相关的错误消息。

参见:pipe, umask。

: oldmask = umask (mask)

设置创建文件的权限掩码。

参数mask是一个整数，解释为八进制数。

如果成功，返回前一个掩码的值(作为一个将被解释为八进制数的整数);否则将打印错误信息。

权限掩码是在文件系统(如文件、目录或命名的fifo)上创建新对象时使用的UNIX概念。要创建的对象具有八进制模式的基本权限，根据mask的八进制值进行修改。新对象的最终权限是mode - mask。

参见:fopen, mkdir, mkfifo。

: [info, err, msg] = stat (file)

: [info, err, msg] = stat (fid)

: [info, err, msg] = lstat (file)

: [info, err, msg] = lstat (fid)

返回一个结构信息，其中包含关于文件或文件标识符fid的以下信息。

dev

ID of device containing a directory entry for this file.

ino

File number of the file.

mode

File mode, as an integer. Use the functions S\_ISREG, S\_ISDIR, S\_ISCHR, S\_ISBLK, S\_ISFIFO, S\_ISLNK, or S\_ISSOCK to extract information from this value.

modestr

File mode, as a string of ten letters or dashes as would be returned by ls -l.

nlink

Number of links.

uid

User ID of file’s owner.

gid

Group ID of file’s group.

rdev

ID of device for block or character special files.

size

Size in bytes.

atime

Time of last access in the same form as time values returned from time. See Timing Utilities.

mtime

Time of last modification in the same form as time values returned from time. See Timing Utilities.

ctime

Time of last file status change in the same form as time values returned from time. See Timing Utilities.

blksize

Size of blocks in the file.

blocks

Number of blocks allocated for file.

如果调用成功，err为0,msg为空字符串。如果文件不存在或出现其他错误，则info为空矩阵，err为−1,msg为对应的系统错误信息。

如果file是一个符号链接，stat将返回链接所引用的实际文件的信息。如果需要有关符号链接本身的信息，请使用lstat。

例如:

[info, err, msg] = stat ("/vmlinuz")

⇒ info =

{

atime = 855399756

rdev = 0

ctime = 847219094

uid = 0

size = 389218

blksize = 4096

mtime = 847219094

gid = 6

nlink = 1

blocks = 768

mode = -rw-r--r--

modestr = -rw-r--r--

ino = 9316

dev = 2049

}

⇒ err = 0

⇒ msg =

参见:lstat, ls, dir, isfile, isfolder。

: tf = S\_ISBLK (mode)

如果模式对应于块设备则返回true。

假定mode的值是从调用stat返回的。

参见:stat, lstat。

: tf = S\_ISCHR (mode)

如果模式对应于字符设备则返回true。

假定mode的值是从调用stat返回的。

参见:stat, lstat。

: tf = S\_ISDIR (mode)

如果模式对应一个目录，则返回true。

假定mode的值是从调用stat返回的。

参见:stat, lstat。

: tf = S\_ISFIFO (mode)

如果模式对应于fifo，则返回true。

假定mode的值是从调用stat返回的。

参见:stat, lstat。

: tf = S\_ISLNK (mode)

如果模式对应于符号链接则返回true。

假定mode的值是从调用stat返回的。

参见:stat, lstat。

: tf = S\_ISREG (mode)

如果模式对应于一个常规文件，则返回true。

假定mode的值是从调用stat返回的。

参见:stat, lstat。

: tf = S\_ISSOCK (mode)

如果模式对应于套接字则返回true。

假定mode的值是从调用stat返回的。

参见:stat, lstat。

: fileattrib

: fileattrib file

: fileattrib (file)

: [status, attrib] = fileattrib (…)

: [status, msg, msgid] = fileattrib (…)

报告文件属性信息。

如果未指定文件或目录，则报告当前工作目录的信息。

如果成功，输出是一个包含以下字段的结构:

Name

Full name of file.

archive

True if file is an archive (Windows).

system

True if file is a system file (Windows).

hidden

True if file is a hidden file (Windows).

directory

True if file is a directory.

UserRead

GroupRead

OtherRead

True if the user (group; other users) has read permission for file.

UserWrite

GroupWrite

OtherWrite

True if the user (group; other users) has write permission for file.

UserExecute

GroupExecute

OtherExecute

True if the user (group; other users) has execute permission for file.

如果某个属性不适用(例如，Unix系统上的archive)，则该字段被设置为NaN。

如果file包含通配符，则以结构数组的形式返回所有匹配文件的信息。

如果请求输出，第一个是status，当操作成功时其值为1，否则为0。如果操作成功，第二个输出包含上面描述的结构(属性);否则，第二个输出是与系统相关的错误消息(msg)。第三个输出是操作成功时的空字符串("")，失败时的唯一消息标识符(msgid)。

参见:stat, glob。

: tf = isfile (f)

如果f是普通文件则返回true，否则返回false。

如果f是字符串的单元格数组，则tf是相同大小的逻辑数组。

参见:isfolder, exist, stat, is\_absolute\_filename, is\_rooted\_relative\_filename。

: tf = isdir (f)

不推荐使用该功能。使用isfolder或file\_in\_loadpath代替。

如果f是目录返回true，否则返回false。

兼容性注意:同名MATLAB函数也会在加载路径目录中搜索f。要模拟此行为，请使用

tf = ! isempty (file\_in\_loadpath (f))

参见:isfolder, file\_in\_loadpath, exist, stat, is\_absolute\_filename, is\_rooted\_relative\_filename。

: tf = isfolder (f)

如果f是目录返回true，否则返回false。

如果f是字符串的单元格数组，则tf是相同大小的逻辑数组。

参见:isfile, exist, stat, is\_absolute\_filename, is\_rooted\_relative\_filename。

: files = readdir (dir)

: [files, err, msg] = readdir (dir)

以字符串的单元格数组的形式返回目录dir中文件的名称。

如果发生错误，则返回文件中的空单元格数组。如果成功，err为0,msg为空字符串。否则，err是非零的，并且msg包含一个系统相关的错误消息。

参见:ls, dir, glob, what。

: cstr = glob (pattern)

给定pattern中的模式字符串数组(作为字符数组或单元格数组)，返回与其中任何一个匹配的文件名的单元格数组，如果没有匹配的模式则返回空的单元格数组。

模式字符串被解释为文件名globbing模式(正如Unix shell使用的那样)。

在一个模式内

\*

匹配任何字符串，包括空字符串，

?

匹配任何单个字符，和

[…]

匹配任何包含的字符。

在查找匹配的文件名之前，对每个模式执行波浪展开。例如:

ls

⇒

file1 file2 file3 myfile1 myfile1b

glob ("\*file1")

⇒

{

[1,1] = file1

[2,1] = myfile1

}

glob ("myfile?")

⇒

{

[1,1] = myfile1

}

glob ("file[12]")

⇒

{

[1,1] = file1

[2,1] = file2

}

注意:在Windows上，不支持包含非ascii字符的模式。

参见:ls, dir, readdir等等。

: fname = file\_in\_path (path, file)

: fname = file\_in\_path (path, file, "all")

如果可以在path中找到，则返回文件的绝对名称。

path的值应该是一个以冒号分隔的目录列表，格式为path所描述的格式。如果没有找到文件，则返回一个空字符串。例如:

file\_in\_path (EXEC\_PATH, "sh")

⇒ "/bin/sh"

如果第二个参数是字符串的单元格数组，则在路径的每个目录中搜索单元格数组的元素，并返回第一个匹配的元素。

如果提供了第三个可选参数"all"，则返回一个单元格数组，其中包含路径中具有相同名称的所有文件的列表。如果没有找到文件，则返回一个空单元格数组。

参见:file\_in\_loadpath, dir\_in\_loadpath, path。

: sep = filesep ()

: filesep ("all")

返回用于分隔目录名的系统相关字符。

如果给出"all"，该函数将以字符串的形式返回所有有效的文件分隔符。文件分隔符列表依赖于系统。在UNIX或Mac OS X下为' / '(正斜杠)，在Windows下为' / '和' \ '(正斜杠和反斜杠)。

参见:pathsep。

: [dir, name, ext] = fileparts (filename)

返回filename的目录、名称和扩展名组件。

输入文件名是一个经过解析的字符串。没有尝试检查指定的文件名或目录是否实际存在。

参见:fullfile, filesep。

: filename = fullfile (dir1, dir2, …, file)

从单独的部分构建完整的文件名。

该函数可以智能地连接任意数量的路径组件。返回值是每个组件的串联，在路径的每个部分之间只有一个文件分隔符，最多有一个前导和/或尾随文件分隔符。

输入参数可以是字符串或单元格字符串。任何单元格字符串的输入参数必须包含一个字符串，或者大小必须相等。在这种情况下，该函数返回一个与输入单元字符串具有相同维度的文件路径的单元字符串，例如:

fullfile ("/home/username", "data", {"f1.csv", "f2.csv", "f3.csv"})

⇒

{

[1,1] = /home/username/data/f1.csv

[1,2] = /home/username/data/f2.csv

[1,3] = /home/username/data/f3.csv

}

在Windows系统上，虽然正斜杠文件分隔符可以工作，但它们被反斜杠取代。此外，驱动器号被剥离前导文件分隔符以获得有效的文件路径。

注意:fullfile不会对生成的完整文件名执行任何验证。

参见:fileparts, filesep。

: newstr = tilde\_expand (string)

: newcstr = tilde\_expand (cellstr)

在弦上进行波浪展开。

如果字符串以波浪字符(' ~ ')开头，则第一个斜杠之前的所有字符(或所有字符，如果没有斜杠)都被视为可能的用户名，并且波浪和以下字符直到斜杠被指定用户的主目录所替换。如果波浪后面紧跟着斜杠，则波浪将被运行Octave的用户的主目录所替换。

如果输入是字符串cellstr的单元数组，则对每个字符串元素执行波浪展开。

例子:

tilde\_expand ("~joeuser/bin")

⇒ "/home/joeuser/bin"

tilde\_expand ("~/bin")

⇒ "/home/jwe/bin"

: [cname, status, msg] = canonicalize\_file\_name (fname)

返回文件fname的规范名称。

如果文件不存在，则返回空字符串("")。不对fname进行波浪展开。

参见:make\_absolute\_filename, is\_absolute\_filename, is\_rooted\_relative\_filename, is\_same\_file, tilde\_expand。

: abs\_fname = make\_absolute\_filename (file)

返回从文件系统根目录开始的文件的全名。

不检查文件是否存在。不执行文件的波浪扩展操作。

参见:canonicalize\_file\_name, is\_absolute\_filename, is\_rooted\_relative\_filename, isfolder, tilde\_expand。

: tf = is\_absolute\_filename (file)

如果file是绝对文件名则返回true。

参见:is\_rooted\_relative\_filename, make\_absolute\_filename, isfolder。

: same = is\_same\_file (filepath1, filepath2)

如果filepath1和filepath2引用同一个文件，则返回true。

如果filepath1或filepath2是字符串的单元格数组，则返回大小相同的数组，其中包含单元格数组中每个成员的上述值。另一个参数也可以是字符串的单元格数组(大小相同)或字符串。

编程注意事项:根据不同的操作系统和文件系统，可以通过不同的路径引用相同的文件或文件夹。特别是，Windows平台上的路径可能在大小写(file1 vs. file1)、文件分隔符(' \ ' vs. ' / ')和格式(A~spaces.txt(8.3约定)vs.带有spaces.txt的文件名)上有所不同。如果filepath1和filepath2中的路径实际上指向同一个文件或文件夹，则此函数返回true，否则返回false。

注意，与strcmp不同，这个函数要求filepath1和filepath2存在，并且指向相同的位置，以便返回true。

参见:canonicalize\_file\_name, strcmp。

: tf = is\_rooted\_relative\_filename (file)

如果文件是根相对文件名则返回true。

参见:is\_absolute\_filename, make\_absolute\_filename, isfolder。

: val = recycle ()

: old\_val = recycle (new\_val)

查询或设置已删除文件的回收优先级。

启用回收后，将永久擦除文件的命令会将它们移动到临时位置(例如标记为Trash的目录)。

编程注意:这个函数是为了与MATLAB兼容而提供的，但是在Octave中没有实现回收。为避免意外数据丢失，如果尝试启用文件回收，将引发错误。

请参见:delete, rmdir。

**36.3 File Archiving Utilities**

: bunzip2 (bzfile)

: bunzip2 (bzfile, dir)

: filelist = bunzip2 (…)

解压缩bzip2归档文件bzfile。

如果指定了dir，则文件将在该目录中解压缩，而不是在bzfile所在的目录中解压缩。

可选的输出文件列表是未压缩文件的列表。

参见:bzip2, unpack, gunzip, unzip, untar。

: filelist = gzip (files)

: filelist = gzip (files, dir)

压缩文件中指定的文件和目录列表。

Files是字符串的字符数组或单元格数组。文件名中的Shell通配符，如' \* '或' ?被接受和扩展。每个文件都被单独压缩，并创建一个扩展名为“。gz”的新文件。原始文件不会被修改，但现有的压缩文件将被静默覆盖。如果指定了一个目录，那么gzip将递归地压缩该目录中的所有文件。

如果定义了dir，则压缩文件将放在此目录中，而不是未压缩文件所在的原始目录中。注意，这不会在dir中复制目录树，如果有多个同名的文件，可能会导致文件相互覆盖。

如果dir不存在，则创建它。

可选的输出文件列表是压缩文件的列表。

参见:gunzip, unpack, bzip2, zip, tar。

: gunzip (gzfile)

: gunzip (gzfile, outdir)

: filelist = gunzip (…)

解压缩gzip文件gzfile。

如果gzfile是一个目录，则该目录下的所有gzfile将被递归解压缩。

如果指定了outdir，则文件将在该目录中解压缩，而不是在gzfile所在的目录中解压缩。

可选的输出文件列表是未压缩文件的列表。

参见:gzip, unpack, bunzip2, unzip, untar。

: filelist = tar (tarfile, files)

: filelist = tar (tarfile, files, rootdir)

将文件中指定的文件和目录列表打包到TAR存档tarfile中。

Files是字符串的字符数组或单元格数组。文件名中的Shell通配符，如' \* '或' ?被接受和扩展。递归遍历目录，并将所有文件添加到存档中。

如果定义了rootdir，那么任何没有绝对路径名的文件都相对于rootdir而不是当前目录定位。

可选的输出文件列表是归档文件中包含的文件列表。

参见:untar, unpack, bzip2, gzip, zip。

: untar (tarfile)

: untar (tarfile, dir)

: filelist = untar (…)

解压缩TAR存档文件。

如果指定了dir，则文件将在此目录而不是当前目录中解压缩。

可选的输出文件列表是未压缩文件的列表。

参见:tar, unpack, bunzip2, gunzip, unzip。

: filelist = zip (zipfile, files)

: filelist = zip (zipfile, files, rootdir)

将files中指定的文件和目录列表压缩到ZIP归档文件zipfile中。

Files是字符串的字符数组或单元格数组。文件名中的Shell通配符，如' \* '或' ?被接受和扩展。目录被递归地遍历，所有文件被压缩并添加到存档中。

如果定义了rootdir，那么任何没有绝对路径名的文件都相对于rootdir而不是当前目录定位。

可选的输出文件列表是归档文件中包含的文件列表。

参见:unzip, unpack, bzip2, gzip, tar。

: unzip (zipfile)

: unzip (zipfile, dir)

: filelist = unzip (…)

解压缩ZIP归档文件。

如果指定了dir，则文件将在此目录而不是当前目录中解压缩。

可选的输出文件列表是未压缩文件的列表。

参见:zip, unpack, bunzip2, gunzip, untar。

: files = unpack (file)

: files = unpack (file, dir)

: files = unpack (file, dir, filetype)

根据扩展名将归档文件解压缩到目录dir。

如果file是字符串列表，则每个文件单独解压缩。文件名中的Shell通配符，如' \* '或' ?被接受和扩展。

如果dir未指定或为空([])，则默认为当前目录。如果目录在文件列表中，则还必须指定文件类型。

特定的归档文件类型是从文件的扩展名推断出来的。文件类型也可以直接使用与已知扩展名对应的字符串指定。

有效的文件类型扩展名:

bz

bz2

bzip archive

gz

gzip archive

tar

tar archive

tarbz

tarbz2

tbz

tbz2

tar + bzip archive

targz

tgz

tar + gzip archive

z

compress archive

zip

zip archive

可选的返回值是未打包的文件列表。

参见:bunzip2, gunzip, unzip, untar, bzip2, gzip, zip, tar。

: filelist = bzip2 (files)

: filelist = bzip2 (files, dir)

压缩文件中指定的文件列表。

Files是字符串的字符数组或单元格数组。文件名中的Shell通配符，如' \* '或' ?被接受和扩展。每个文件都被单独压缩，新文件带有“。创建“Bz2”扩展。原始文件不会被修改，但现有的压缩文件将被静默覆盖。

如果定义了dir，则压缩文件将放在此目录中，而不是未压缩文件所在的原始目录中。注意，这不会在dir中复制目录树，如果有多个同名的文件，可能会导致文件相互覆盖。

如果dir不存在，则创建它。

可选的输出文件列表是压缩文件的列表。

参见:bunzip2, unpack, gzip, zip, tar。

**36.4组网实用程序**

: name = gethostname ()

返回运行Octave的系统的主机名。

**36.4.1 FTP对象**

Octave通过面向对象的接口支持FTP协议。使用ftp函数创建一个表示连接的ftp对象。所有FTP函数都以FTP对象作为第一个参数。

: f = ftp (host)

: f = ftp (host, username, password)

使用用户名和密码连接FTP服务器主机。

如果不指定用户名和密码，则使用不带密码的“anonymous”用户。返回的FTP对象f表示已建立的FTP连接。

FTP对象的操作列表如下所示。所有函数都需要一个FTP对象作为第一个参数。

| **Method** | **Description** |
| --- | --- |
| ascii | Set transfer type to ascii |
| binary | Set transfer type to binary |
| cd | Change remote working directory |
| close | Close FTP connection |
| delete | Delete remote file |
| dir | List remote directory contents |
| mget | Download remote files |
| mkdir | Create remote directory |
| mput | Upload local files |
| rename | Rename remote file or directory |
| rmdir | Remove remote directory |

参见:@ftp/ascii， @ftp/二进制，@ftp/cd， @ftp/close， @ftp/delete， @ftp/dir， @ftp/mget， @ftp/mkdir， @ftp/mput， @ftp/rename， @ftp/rmdir。

: close (f)

关闭FTP对象f表示的FTP连接。

f是FTP函数返回的FTP对象。

参见:@ftp/ftp。

: mget (f, file)

: mget (f, dir)

: mget (f, remote\_name, target)

f.通过FTP连接下载远程文件file或目录dir到本地目录。

f是FTP函数返回的FTP对象。

参数file和dir可以包含通配符，并且将下载远程服务器上匹配的任何文件或目录。

如果给出了第三个字符串参数target，那么它必须指示到本地目标目录的路径。目标可以是相对路径或绝对路径。

参见:@ftp/mput， @ftp/ftp。

: mput (f, file)

: file\_list = mput (f, file)

f.通过FTP连接将本地文件上传至远程当前目录。

f是FTP函数返回的FTP对象。

参数file通过glob函数传递，任何与file中的通配符匹配的文件都将被上传。

可选的输出参数file\_list包含一个包含上传文件名称的字符串单元格数组。

参见:@ftp/mget， @ftp/mkdir， @ftp/ftp。

: cwd = cd (f)

: cd (f, path)

: new\_cwd = cd (f, path)

获取或设置FTP连接上的远程目录。

f是FTP函数返回的FTP对象。

如果未指定path，则返回远程当前工作目录。否则，将远程目录设置为path，并返回新的远程工作目录。

如果目录不存在，则打印错误信息，并且不更改工作目录。

参见:@ftp/dir， @ftp/ftp。

: dir (f)

: lst = dir (f)

以详细形式列出FTP连接的当前目录。

f是FTP函数返回的FTP对象。

如果请求可选输出列表，则返回一个struct数组，每个文件有一个条目，包含字段name、date、bytes、isdir、datenum。

参见:@ftp/cd， @ftp/mkdir， @ftp/rmdir， @ftp/ftp。

: ascii (f)

设置FTP连接使用ASCII方式传输。

ASCII模式只适用于文本文件，因为它将远程主机的换行符表示转换为本地主机的换行符表示。

f是FTP函数返回的FTP对象。

参见:@ftp/binary， @ftp/ftp。

: binary (f)

设置FTP连接使用二进制传输方式。

在二进制模式下，没有从远程表示到本地表示的换行符转换。

f是FTP函数返回的FTP对象。

参见:@ftp/ascii， @ftp/ftp。

: delete (f, file)

删除通过FTP连接的远程文件file。

f是FTP函数返回的FTP对象。

请参见:@ftp/rmdir， @ftp/rename， @ftp/ftp。

: rename (f, oldname, newname)

通过FTP连接将远程文件或目录oldname重命名或移动到newname。

f是FTP函数返回的FTP对象。

参见:@ftp/delete， @ftp/rmdir， @ftp/ftp。

: mkdir (f, path)

通过FTP连接创建远程目录路径。

f是FTP函数返回的FTP对象。

参见:@ftp/rmdir， @ftp/ftp。

: rmdir (f, path)

通过FTP连接删除远程目录路径。

f是FTP函数返回的FTP对象。

参见:@ftp/delete， @ftp/mkdir， @ftp/rename， @ftp/ftp。

**36.4.3 Base64和二进制数据传输**

有些传输通道不能接受二进制数据。习惯上用Base64编码二进制数据进行传输，并在接收时解码数据。

: s = base64\_encode (x)

将双精度矩阵或数组x编码为base64格式字符串s。

参见:base64\_decode, matlab.net.base64decode, matlab.net.base64encode。

: x = base64\_decode (s)

: x = base64\_decode (s, dims)

从base64编码的字符串s解码双矩阵或数组x。

可选输入参数dims应该是一个包含解码数组尺寸的向量。

参见:base64\_encode, matlab.net.base64decode, matlab.net.base64encode。

: b64\_str = matlab.net.base64encode (in)

转换为base64编码的字符串b64\_str。

输入in可以是字符串或数字向量。输出b64\_str将根据RFC 4648进行编码。

参见:matlab.net.base64decode, base64\_decode, base64\_encode, unicode2native。

: out\_vec = matlab.net.base64decode (b64\_str)

将base64编码的b64\_str转换为uint8向量out\_vec。

输入b64\_str必须是一个字符串向量。输出out\_vec将是一个uint8向量，根据RFC 4648进行解码。

参见:matlab.net.base64encode, base64\_decode, base64\_encode, native2unicode。

**36.5控制子进程**

Octave包括一些高级命令，如system和popen，用于启动子进程。如果您希望运行另一个程序来执行某些任务，然后查看其输出，那么您可能需要使用这些函数。

Octave还提供了几个非常低级的类unix函数，这些函数也可以用于启动子进程，但是如果您无法找到任何方法来完成您需要的高级函数，则可能应该只使用它们。

: system ("string")

: system ("string", return\_output)

: system ("string", return\_output, type)

: [status, output] = system (…)

执行由字符串指定的shell命令。

如果使用一个或多个输出参数调用system，或者如果可选参数return\_output为true并且同步启动子进程，则命令的输出将作为变量返回。否则，如果同步执行子流程，则将其输出发送到标准输出。要通过分页器发送与system一起执行的命令的输出，可以使用如下命令

[~, text] = system ("cmd");

more on;

disp (text);

或

more on;

printf ("%s\n", nthargout (2, "system", "cmd"));

如果可选参数类型为"async"，则进程在后台启动，并立即返回子进程的进程ID。否则，子进程启动，Octave等待，直到它退出。如果省略type参数，则默认值为"sync"。

系统函数可以返回两个值。第一个是命令的退出状态，第二个是写入标准输出流的命令的任何输出。例如,

[status, output] = system ("echo foo & exit 2");

将变量输出设置为字符串' foo '，并将变量状态设置为整数' 2 '。

对于异步执行的命令，“status”为执行该命令所启动的命令shell的进程号。

执行命令使用的shell因操作系统而异，UNIX系统通常为/bin/sh, Windows系统通常为cmd.exe。

参见:unix, dos。

: unix ("command")

: status = unix ("command")

: [status, text] = unix ("command")

: […] = unix ("command", "-echo")

如果在类unix操作系统下运行，则执行系统命令，否则什么都不做。

Octave等待外部命令完成，然后返回程序的退出状态和任何文本输出。

如果调用时没有输出参数，或者给出了"-echo"参数，则文本也会发送到标准输出。

参见:dos, system, isunix, ismac, ispc。

: dos ("command")

: status = dos ("command")

: [status, text] = dos ("command")

: […] = dos ("command", "-echo")

如果在类似windows的操作系统下运行，则执行系统命令，否则什么都不做。

Octave等待外部命令完成，然后返回程序的退出状态和任何文本输出。

如果调用时没有输出参数，或者给出了"-echo"参数，则文本也会发送到标准输出。

参见:unix, system, isunix, ismac, ispc。

: open file

: output = open (file)

根据文件扩展名确定的文件类型，在Octave或外部应用程序中打开文件文件。

默认情况下，可识别的文件类型为

.m

Open file in the editor. No output value is returned.

.mat

octave-workspace

Open the data file with load. If no return value output is requested, variables are loaded in the base workspace. Otherwise output will be a structure containing loaded data. See load function.

.ofig

Open the figure with hgload. See hgload function.

.fig, .ofig

Load the figure

.exe

Execute the program (on Windows systems only). No output value is returned.

如果在加载路径中找到函数openxxx(其中xxx是扩展名)，也可以处理自定义文件扩展名。函数必须接受文件名作为输入。例如，为了在基本工作区中加载“。dat”数据文件，默认情况下为“。dat”。在文件中，可以定义“open”。M”，内容如下:

function retval = opendat (fname)

evalin ("base", sprintf ("load ('%s');", fname));

endfunction

在适当的外部应用程序中打开其他文件类型。

: output = perl (scriptfile)

: output = perl (scriptfile, argument1, argument2, …)

: [output, status] = perl (…)

调用Perl脚本scriptfile，可能带有命令行参数列表。

在output中返回输出，在status中返回可选状态。如果scriptfile不是绝对文件名，则在当前目录中搜索，然后在Octave加载路径中搜索。

参见:system, python。

: output = python (scriptfile)

: output = python (scriptfile, argument1, argument2, …)

: [output, status] = python (…)

调用Python脚本scriptfile，可能带有命令行参数列表。

在output中返回输出，在status中返回可选状态。如果scriptfile不是绝对文件名，则在当前目录中搜索，然后在Octave加载路径中搜索。

编程注意:在UNIX系统上，脚本将由python3执行，在Windows上由python执行。你可以通过设置PYTHON环境变量来覆盖这些默认值，例如在Octave中使用setenv PYTHON /usr/local/bin/python3.

参见:system, perl。

: fid = popen (command, mode)

启动一个进程并创建一个管道。

要运行的命令的名称由command给出。论证模式可以是

“r”

管道将连接到工艺的标准输出，并打开读取。

“w”

该管道将连接到进程的标准输入，并为写入打开。

与进程的输入或输出流相对应的文件标识符在fid中返回。

例如:

fid = popen ("ls -ltr / | tail -3", "r");

while (ischar (s = fgets (fid)))

fputs (stdout, s);

endwhile

-| drwxr-xr-x 33 root root 3072 Feb 15 13:28 etc

-| drwxr-xr-x 3 root root 1024 Feb 15 13:28 lib

-| drwxrwxrwt 15 root root 2048 Feb 17 14:53 tmp

参见:popen2。

: status = pclose (fid)

关闭由popen打开的文件标识符fid。

如果成功，fclose返回0，否则返回-1。

编程注意:函数fclose也可以用于相同的目的。

参见:关闭，打开。

: [in, out, pid] = popen2 (command, args)

启动具有双向通信的子流程。

进程的名称由command给出，args是包含命令选项的字符串数组或单元格数组。

子流程的输入和输出流的文件标识符在in和out中返回。如果命令执行成功，则pid包含子进程的进程号。否则，pid为−1。

例如:

[in, out, pid] = popen2 ("sort", "-r");

fputs (in, "these\nare\nsome\nstrings\n");

fclose (in);

EAGAIN = errno ("EAGAIN");

done = false;

do

s = fgets (out);

if (ischar (s))

fputs (stdout, s);

elseif (errno () == EAGAIN)

pause (0.1);

fclear (out);

else

done = true;

endif

until (done)

fclose (out);

waitpid (pid);

-| these

-| strings

-| some

-| are

注意，与popen不同，popen2不会“收获”子进程。如果您不使用waitpid来检查子进程的退出状态，它将一直停留直到Octave退出。

参见:popen, waitpid。

: val = EXEC\_PATH ()

: old\_val = EXEC\_PATH (new\_val)

: old\_val = EXEC\_PATH (new\_val, "local")

查询或设置内部变量，该变量指定在执行外部程序时附加到shell PATH的冒号分隔的目录列表。

的初始值取自环境变量OCTAVE\_EXEC\_PATH，但是该值可以被命令行参数——exec-path PATH覆盖。

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:IMAGE\_PATH, OCTAVE\_HOME, OCTAVE\_EXEC\_HOME。

在大多数情况下，以下函数只是解码它们的参数并进行相应的Unix系统调用。有关如何使用它们的完整示例，请查看popen2函数的定义。

: [pid, msg] = fork ()

创建当前进程的副本。

Fork可以返回以下值之一:

> 0

您处于父进程中。从fork返回的值是子进程的进程id。您可能应该安排等待任何子进程退出。

0

您处于子进程中。您可以调用exec来启动另一个进程。如果失败，您可能应该调用exit。

< 0

由于某些原因，对fork的调用失败。你必须采取规避行动。一个系统相关的错误消息将在msg中等待。

: [err, msg] = exec (file, args)

用新流程替换当前流程。

调用exec而不首先调用fork将终止当前的Octave进程，并将其替换为file命名的程序。例如,

exec ("ls", "-l")

将运行ls并将您返回到shell提示符。

如果成功，exec不会返回。如果exec确实返回，err将是非零的，并且msg将包含一个系统相关的错误消息。

: [read\_fd, write\_fd, err, msg] = pipe ()

创建一个管道，并将管道的读端和写端分别返回给read\_fd和write\_fd。

如果成功，err为0,msg为空字符串。否则，err是非零的，并且msg包含一个系统相关的错误消息。

参见:mkfifo。

: [fid, msg] = dup2 (old, new)

复制一个文件描述符。

如果成功，则fid大于零并包含新的文件ID。否则，fid为负值，msg包含系统相关的错误消息。

参见:fopen, fclose, fcntl。

: [pid, status, msg] = waitpid (pid, options)

等待进程pid终止。

pid参数可以是:

−1

等待任何子进程。

0

等待进程组ID等于Octave解释器进程的子进程。

> 0

等待ID为pid的子进程终止。

options参数可以是以下零或多个常量的位或:

0

等待，直到接收到信号或子进程退出(如果缺少options参数，这是默认值)。

WNOHANG

如果状态不能立即可用，不要挂起。

WUNTRACED

报告任何停止的子进程的状态，以及自停止以来尚未报告其状态的子进程的状态。

WCONTINUE

如果已停止的子进程通过SIGCONT的传递恢复，则返回。该值可能对所有系统都没有意义。

如果返回值大于0，则表示退出的子进程的进程号。如果发生错误，pid将小于零，msg将包含一个系统相关的错误消息。status的值包含有关已退出子进程的其他系统相关信息。

参见：WCONTINUE, WCOREDUMP, WEXITSTATUS, WCONTINUE, wifinterrupted, WIFSTOPPED, WNOHANG, WSTOPSIG, WTERMSIG, untrace。

: v = WCONTINUE ()

返回WCONTINUE宏的数值。

WCONTINUE是可以传递给waitpid的选项参数，用于指示如果通过传递SIGCONT信号恢复了被停止的子进程，它也应该返回。

参见:waitpid, WNOHANG, wuntracked。

: tf = WCOREDUMP (status)

给定调用waitpid的状态，如果子进程产生核心转储，则返回true。

这个函数应该只在wifsignal返回true时使用。用于实现此函数的宏在POSIX.1-2001中没有指定，并且在某些Unix实现(例如AIX、SunOS)中不可用。

参见:waitpid, WIFEXITED, WEXITSTATUS, wifsignals, WTERMSIG, WIFSTOPPED, WSTOPSIG, wifcontinue。

: tf = WEXITSTATUS (status)

给定调用waitpid的状态，返回子进程的退出状态。

只有当WIFEXITED返回true时，才应该使用这个函数。

参见:waitpid, WIFEXITED, wifsignals, WTERMSIG, WCOREDUMP, WIFSTOPPED, WSTOPSIG, wifcontinue。

: tf = WIFCONTINUED (status)

给定调用waitpid的状态，如果通过传递SIGCONT恢复子进程，则返回true。

参见:waitpid, WIFEXITED, WEXITSTATUS, wifsignals, WTERMSIG, WCOREDUMP, WIFSTOPPED, WSTOPSIG。

: tf = WIFSIGNALED (status)

给定调用waitpid的状态，如果子进程被信号终止则返回true。

参见:waitpid, WIFEXITED, WEXITSTATUS, WTERMSIG, WCOREDUMP, WIFSTOPPED, WSTOPSIG, wifcontinue。

: tf = WIFSTOPPED (status)

给定调用waitpid的状态，如果子进程通过传递信号而停止，则返回true。

只有当调用是使用wuntracked或子进程被跟踪时才有可能(参见ptrace(2))。

参见:waitpid, WIFEXITED, WEXITSTATUS, wifsignals, WTERMSIG, WCOREDUMP, WSTOPSIG, wifcontinue。

: tf = WIFEXITED (status)

给定调用waitpid的状态，如果子进程正常终止则返回true。

参见:waitpid, WEXITSTATUS, wifsignals, WTERMSIG, WCOREDUMP, WIFSTOPPED, WSTOPSIG, wifcontinue。

: v = WNOHANG ()

返回WNOHANG宏的数值。

WNOHANG是一个选项参数，可以传递给waitpid，以指示它应该立即返回其状态，而不是等待进程退出。

参见:waitpid, wuntracked, WCONTINUE。

: tf = WSTOPSIG (status)

给定调用waitpid的状态，返回导致子进程停止的信号号。

只有当wifstop返回true时，才应该使用这个函数。

参见:waitpid, WIFEXITED, WEXITSTATUS, wifsignals, WTERMSIG, WCOREDUMP, WIFSTOPPED, wifcontinue。

: tf = WTERMSIG (status)

给定调用waitpid的状态，返回导致子进程终止的信号号。

这个函数应该只在wifsignal返回true时使用。

参见:waitpid, WIFEXITED, WEXITSTATUS, wifsignals, WCOREDUMP, WIFSTOPPED, WSTOPSIG, wifcontinue。

: v = WUNTRACED ()

返回wuntracked宏的数值。

wuntracked是一个选项参数，它可以传递给waitpid，表明如果子进程已经停止，但没有通过ptrace系统调用进行跟踪，waitpid也应该返回

参见:waitpid, WNOHANG, WCONTINUE。

: fcntl (fid, request, arg)

: [status, msg] = fcntl (fid, request, arg)

修改打开文件fid的属性。

以下值可以作为请求传递:

F\_DUPFD

Return a duplicate file descriptor.

F\_GETFD

Return the file descriptor flags for fid.

F\_SETFD

Set the file descriptor flags for fid.

F\_GETFL

Return the file status flags for fid. The following codes may be returned (some of the flags may be undefined on some systems).

O\_RDONLY

Open for reading only.

O\_WRONLY

Open for writing only.

O\_RDWR

Open for reading and writing.

O\_APPEND

Append on each write.

O\_CREAT

Create the file if it does not exist.

O\_NONBLOCK

Non-blocking mode.

O\_SYNC

Wait for writes to complete.

O\_ASYNC

Asynchronous I/O.

F\_SETFL

Set the file status flags for fid to the value specified by arg. The only flags that can be changed are O\_APPEND and O\_NONBLOCK.

如果成功，status为0,msg为空字符串。否则，status为-1,msg包含与系统相关的错误消息。

参见:fopen, dup2。

: kill (pid, sig)

: [status, msg] = kill (pid, sig)

发送信号信号到进程pid。

如果pid为正，则向pid发送信号信号。

如果pid为0，则向当前进程所属进程组中的每个进程发送信号信号。

如果pid为-1，则信号信号被发送到除进程1之外的每个进程。

如果pid小于-1，则向进程组-pid中的每个进程发送信号信号。

如果sig为0，则不发送信号，但仍进行错误检查。

如果成功，status为0,msg为空字符串。否则，status为-1,msg包含与系统相关的错误消息。

: S = SIG ()

返回一个包含Unix信号名称及其定义值的结构。

36.6进程、组和用户id

: pgid = getpgrp ()

返回当前进程的进程组id。

: pid = getpid ()

返回当前进程的进程id。

参见:getppid。

: pid = getppid ()

返回父进程的进程id。

参见:getpid。

: euid = geteuid ()

返回当前进程的有效用户id。

参见:getuid。

: uid = getuid ()

返回当前进程的真实用户id。

参见:geteuid。

: egid = getegid ()

返回当前进程的有效组id。

参见:getgid。

: gid = getgid ()

返回当前进程的实际组id。

参见:getegid。

**36.7环境变量**

: val = getenv ("var")

返回环境变量var的值。

例如,

getenv ("PATH")

返回包含路径值的字符串。

参见:setenv, unsetenv。

: tf = isenv (var)

如果变量var是环境变量，则返回true，否则返回false。

例如,

通常在UNIX系统上返回true，因为"PATH"是UNIX的环境变量。

参见:getenv, setenv, unsetenv。

: setenv ("var", value)

: setenv (var)

: putenv (…)

将环境变量var的值设置为value。

如果没有指定值，则该变量将被分配空字符串。

编程注意:putenv是setenv的别名，可以互换使用。

参见:unsetenv, getenv, isenv。

: unsetenv ("var")

: status = unsetenv ("var")

删除环境变量var。

如果变量被删除或不存在，则返回0，如果发生错误则返回-1。

参见:setenv, getenv, isenv。

: homedir = get\_home\_directory ()

返回当前主目录。

在大多数系统上，这相当于getenv ("HOME")。在Windows系统上，如果没有设置环境变量HOME，那么它相当于fullfile (getenv ("HOMEDRIVE")， getenv ("HOMEPATH"))

参见:get。

**36.8当前工作目录**

: cd dir

: cd

: old\_dir = cd

: old\_dir = cd (dir)

: chdir …

将当前工作目录更改为dir。

如果在没有输入或输出参数的情况下调用，则当前目录将更改为用户的主目录(“~”)。

例如,

cd ~/octave

将当前工作目录更改为~/octave。如果目录不存在，则打印错误信息，并且不更改工作目录。

编程注意:chdir是cd的别名，可以用于所有相同的调用格式。

兼容性注意:当不带参数调用时，MATLAB打印当前工作目录，而不是更改到用户的主目录。

参见:pwd, mkdir, rmdir, dir, ls。

: ls

: ls filenames

: ls options

: ls options filenames

: list = ls (…)

列出目录内容。

如果ls命令可用，则ls函数将转发到ls命令。它退回到调用本机操作系统的目录列表命令。可用的选项可能因系统而异。

如果文件名包含任何通配符“\*”，“?””、“[]”。如果这些通配符用反斜杠' \ '(例如' \\* ')转义，则它们不被视为通配符，而是作为相应的文字字符。

如果请求可选的输出列表，则ls返回一个字符数组，每个文件/目录名对应一行。

在类unix系统上的示例用法:

ls -l

-| total 12

-| -rw-r--r-- 1 jwe users 4488 Aug 19 04:02 foo.m

-| -rw-r--r-- 1 jwe users 1315 Aug 17 23:14 bar.m

参见:dir、readdir、glob、what、stat、filesep、ls\_command。

: val = ls\_command ()

: old\_val = ls\_command (new\_val)

查询或设置Octave的ls命令使用的shell命令。

参见:ls。

: dir

: dir directory

: [list] = dir (directory)

显示目录directory的文件列表。

如果未指定directory，则列出当前工作目录。

如果请求返回值，则返回包含字段的结构数组

name

File or directory name.

folder

Location of file or directory

date

Timestamp of file modification (string value).

bytes

File size in bytes.

isdir

True if name is a directory.

datenum

Timestamp of file modification as serial date number (double).

statinfo

Information structure returned from stat.

如果directory是文件名，而不是目录，则返回有关命名文件的信息。目录也可以是一个列表，而不是单个目录或文件。

如果目录包含任何通配符“\*”，“?””、“[]”。如果这些通配符在POSIX平台上使用反斜杠' \ '(例如' \\* ')进行转义，则它们不会被视为通配符，而是作为相应的文字字符。在Windows上，不可能转义通配符，因为反斜杠“\”被视为文件分隔符。在Windows上，对于包含被dir视为通配符的字符的文件或文件夹名，使用ls。

注意，对于符号链接，dir返回符号链接指向的文件的信息，而不是链接本身的信息。但是，如果链接指向一个不存在的文件，dir返回关于该链接的信息。

参见:ls, readdir, glob, what, stat, lstat。

: dir = pwd ()

返回当前工作目录。

参见:cd, dir, ls, mkdir, rmdir。

**36.9密码数据库功能**

Octave的密码数据库函数以以下字段的结构返回信息。

name

The user name.

passwd

The encrypted password, if available.

uid

The numeric user id.

gid

The numeric group id.

gecos

The GECOS field.

dir

The home directory.

shell

The initial shell.

在以下函数的描述中，这个数据结构被称为pw\_struct。

: pw\_struct = getpwent ()

返回一个包含密码数据库条目的结构，必要时打开它。

一旦到达数据的末尾，getpgo返回0。

参见:setpwent, endpwent。

: pw\_struct = getpwuid (uid).

返回一个结构，其中包含密码数据库中用户ID为uid的第一个条目。

如果用户ID在数据库中不存在，getpuid返回0。

参见:getpwnam。

: pw\_struct = getpwnam (name)

返回一个结构，其中包含密码数据库中用户名为name的第一个条目。

如果用户名在数据库中不存在，getpwname返回0。

参见:getpuid。

: [status, msg] = setpwent ()

返回指向密码数据库开头的内部指针。

参见:getpwent, endpwent。

: [status, msg] = endpwent ()

关闭密码数据库。

参见:getpwent, setpwent。

**36.10组数据库功能**

Octave的组数据库函数以以下字段的结构返回信息。

name

The user name.

passwd

The encrypted password, if available.

gid

The numeric group id.

mem

The members of the group.

在以下函数的描述中，这个数据结构被称为grp\_struct。

: grp\_struct = getgrent ()

从组数据库返回一个条目，必要时打开它。

一旦到达数据的末尾，getgrant返回0。

参见:setgreent, endgreent。

: grp\_struct = getgrgid (gid).

返回组数据库中组ID为gid的第一个条目。

如果组ID在数据库中不存在，getgrgid返回0。

参见:getgram。

: grp\_struct = getgrnam (name)

返回组数据库中组名为name的第一个条目。

如果组名在数据库中不存在，getgrnam返回0。

参见:getgrgid。

: [status, msg] = setgrent ()

返回指向组数据库开头的内部指针。

参见:getgrant, endent。

: [status, msg] = endgrent ()

关闭组数据库。

参见:getgrant, setgrant。

**36.11系统信息**

: computer ()

: comp = computer ()

: [comp, maxsize] = computer ()

: [comp, maxsize, endian] = computer ()

: arch = computer ("arch")

打印或返回cpu-vendor-os形式的字符串，该字符串标识Octave正在运行的计算机类型。

如果使用输出参数调用，则返回值而不是打印值。例如:

computer ()

-| x86\_64-pc-linux-gnu

mycomp = computer ()

⇒ mycomp = x86\_64-pc-linux-gnu

如果请求两个输出参数，还返回数组的最大元素数。这将取决于Octave是使用32位还是64位索引向量编译的。

如果请求三个输出参数，也以字符形式返回当前系统的字节顺序(“B”表示大端序，“L”表示小端序)。

如果指定了参数"arch"，则返回一个字符串，指示运行Octave的计算机的体系结构。

如果使用- tradition选项调用Octave，结果可能会有所不同。

参见:isunix, ismac, ispc。

: [uts, err, msg] = uname ()

返回结构中的系统信息。

例如:

uname ()

⇒ {

sysname = x86\_64

nodename = segfault

release = 2.6.15-1-amd64-k8-smp

version = Linux

machine = #2 SMP Thu Feb 23 04:57:49 UTC 2006

}

如果成功，err为0,msg为空字符串。否则，err是非零的，并且msg包含一个系统相关的错误消息。

: n = nproc ()

: n = nproc (query)

返回当前可用(逻辑)处理器的数量。

这将返回逻辑处理器的数量。对于具有超线程的处理器，这比物理内核的数量还要大。

如果用可选参数query调用，修改处理器的计数方式如下:

all

total number of processors.

current

processors available to the current process.

overridable

same as current, but overridable through the OMP\_NUM\_THREADS environment variable.

: tf = ispc ()

如果Octave在Windows系统上运行则返回true，否则返回false。

参见:isunix, ismac。

: tf = isunix ()

还真如果八度是在类unix系统上运行否则,则返回false。

参见:ismac, ispc。

: tf = ismac ()

如果Octave在Mac OS X系统上运行则返回true，否则返回false。

参见:isunix, ispc。

: tf = isieee ()

如果您的计算机声称符合浮点计算的IEEE标准，则返回true。

不执行任何实际测试。

: tf = isdeployed ()

如果当前程序已编译并独立于Octave解释器运行，则返回true;如果在Octave解释器中运行，则返回false。

目前，这个函数在Octave中总是返回false。

: tf = isstudent ()

如果在学生版的MATLAB中运行，返回true。

isstudent在Octave中总是返回false。

参见:false。

: dir = OCTAVE\_HOME ()

返回顶级Octave安装目录的名称。

OCTAVE\_HOME对应于配置变量前缀。

参见:EXEC\_PATH, IMAGE\_PATH, OCTAVE\_EXEC\_HOME。

: dir = OCTAVE\_EXEC\_HOME ()

返回与体系结构相关的文件的顶级Octave安装目录的名称。

如果不单独指定，则与OCTAVE\_HOME相同。OCTAVE\_EXEC\_HOME对应于配置变量exec\_prefix。

参见:EXEC\_PATH, IMAGE\_PATH, OCTAVE\_HOME。

: dir = matlabroot ()

返回顶级Octave安装目录的名称。

这是为兼容性提供的函数OCTAVE\_HOME的别名。

参见:OCTAVE\_HOME。

: cfg\_dir = user\_config\_dir ()

返回用户配置的(特定于平台的)目录。

参见:user\_data\_dir。

: data\_dir = user\_data\_dir ()

返回用户数据的(特定于平台的)目录。

参见:user\_config\_dir。

: verstr = OCTAVE\_VERSION ()

以字符串形式返回Octave的版本号。

参见:ver, version。

: v = version ()

: [v, d] = version ()

: v = version (feature)

获取Octave的版本信息。

如果在没有输入参数的情况下调用，第一个返回值v以字符串形式给出Octave的版本号。第二个返回值d以字符串形式保存发布日期。

以下选项可以传递给feature:

"-date"

for the release date of the running build,

"-description"

for a description of the release (always an empty string),

"-release"

for the name of the running build (always an empty string),

"-java"

for version information of the Java VM,

"-fftw"

for version information for the linked FFTW,

"-blas"

for version information for the linked BLAS,

"-lapack"

for version information for the linked LAPACK.

"-hgid"

the mercurial ID of the sources used to build Octave.

“-blas”和“-lapack”选项返回的信息可能不可靠。它可能会报告在构建Octave时链接的库，而不是当前使用的库。

没有输入和输出参数的变体是为兼容性提供的OCTAVE\_VERSION函数的别名。

参见:OCTAVE\_VERSION, ver。

: ver

: ver Octave

: ver package

: v = ver (…)

显示包含当前Octave版本号、许可证字符串和操作系统的标头。头文件后面是安装包、版本和安装目录的列表。

使用包名称package或Octave查询指定组件。

当带输出参数调用时，返回描述Octave和每个已安装包的结构向量。每个结构包括以下字段。

Name

Package name.

Version

Version of the package.

Release

Release of the package (currently unused, defaults to []).

Date

Date that the version was released.

参见:version, usejava, pkg。

: tf = compare\_versions (v1, v2, operator)

使用给定的操作符比较两个版本字符串。

该函数假设版本v1和v2是任意长字符串，由数字和句点字符组成，可能后跟一个任意字符串(例如，"1.2.3"，"0.3"，"0.1.2+"或"1.2.3.4-test1")。

该版本首先被分成数字和字符部分，然后这些部分被填充为相同的长度(即，当与“1.1.1”进行比较时，“1.1”将被填充为“1.1.0”，并且单独地，字符串的字符部分用null填充)。

运算符可以是集合中的任何逻辑运算符

* "==" equal
* "<" less than
* "<=" less than or equal to
* ">" greater than
* ">=" greater than or equal to
* "!=", "~=" not equal

注意，版本"1.1-test2"将比"1.1-test10"大。此外，由于首先比较数字部分，“a”比“1a”小，因为第二个字符串以数字部分开始，即使双精度(“a”)大于双精度(“1”)。

: tf = verLessThan (package, version)

如果包的安装版本小于version，则返回true。

Package是要检查的包的名称。使用“Octave”作为包来检查Octave本身的版本。

Version是要与之比较的版本。版本是由compare\_versions接受的格式的字符串:由数字和句点字符组成的任意长字符串，可能后跟任意字符串(例如，"1.2.3"，"0.3"，"0.1.2+"或"1.2.3.4-test1")。

例子:

tf = verLessThan ("Octave", "5")

⇒ tf = 0

tf = verLessThan ("io", "2.4.12")

⇒ ...

if (! verLessThan ("Octave", "5"))

## ... use new Octave 5 features ...

endif

参见:compare\_versions, version, ver, pkg。

: license

: license inuse

: license inuse feature

: license ("inuse")

: license\_struct = license ("inuse")

: license\_struct = license ("inuse", feature)

: status = license ("test", feature)

: status = license ("checkout", feature)

: [status, errmsg] = license ("checkout", feature)

获取Octave和Octave软件包的许可信息。

GNU Octave是在GNU通用公共许可证(GPL)下发布的自由软件，许可证管理器没有意义。此函数仅为MATLAB兼容而提供。

在没有额外输入参数的情况下调用时，它返回Octave许可证，否则第一个输入定义了操作模式，并且必须是以下字符串之一:inuse、test和checkout。可选的feature参数可以是"octave" (core)或octave包的名称。

"inuse"

打印已加载的特性列表，例如“octave”和已加载的包列表。如果请求输出，它将返回一个包含“feature”和“user”字段的结构数组。

"test"

如果安装了指定的特性，则返回true，否则返回false。

可选的第三个参数“enable”或“disable”被接受但被忽略。

"checkout"

如果安装了指定的特性，则返回true，否则返回false。如果没有安装包，第二个可选输出将显示错误消息。

参见:pkg, ver, version。

: memory ()

: [userdata, systemdata] = memory ()

显示或返回有关Octave内存使用情况的信息。

如果在没有输出参数的情况下调用该函数，则会显示一个包含当前内存消耗概述的表。

输出参数userdata是一个结构体，包含以下字段，其中包含Octave进程的数据:

MaxPossibleArrayBytes

要分配的数组的最大大小。请注意，这包括所有物理内存和交换空间。分配那么多内存可能会导致系统不稳定、数据损坏和/或文件系统损坏。请注意，根据平台的不同(32位系统)，最大的连续内存块可能会进一步限制可分配数组的最大值。当前未实现此检查。

MemAvailableAllArrays

可用内存的总大小，以字节为单位。

ram\_available\_all\_arrays

可以在物理内存中分配的数组的最大大小(不包括交换空间)。请注意，根据平台的不同(32位系统)，最大的连续内存块可能会进一步限制可分配数组的最大值。当前未实现此检查。

MemUsedMATLAB

mem\_used\_octave

当前Octave使用的内存(包括交换空间)，以字节为单位。

ram\_used\_octave

Octave当前使用的物理内存(不包括交换空间)，以字节为单位。

输出参数systemdata是一个嵌套结构，包含以下字段，其中包含有关系统内存的信息:

PhysicalMemory.Available

当前可用的物理内存，以字节为单位。

PhysicalMemory.Total

总物理内存(以字节为单位)。

SystemMemory.Available

当前可用的内存(包括交换空间)，以字节为单位。

SystemMemory.Total

总内存(包括交换空间)，以字节为单位。

VirtualAddressSpace.Available

当前可用的虚拟地址空间，以字节为单位。

VirtualAddressSpace.Total

总虚拟地址空间(以字节为单位)。

示例#1:打印格式化的内存使用表

memory ()

⇒

System RAM: 3934008 KiB, swap: 4087804 KiB

Octave RAM: 170596 KiB, virt: 1347944 KiB

Free RAM: 1954940 KiB, swap: 4087804 KiB

Available RAM: 2451948 KiB, total: 6042744 KiB

示例#2:返回带有内存使用信息的结构体

[userdata, systemdata] = memory ()

⇒

userdata =

scalar structure containing the fields:

MaxPossibleArrayBytes = 6.1622e+09

MemAvailableAllArrays = 6.1622e+09

ram\_available\_all\_arrays = 2.4883e+09

MemUsedMATLAB = 1.3825e+09

mem\_used\_octave = 1.3825e+09

ram\_used\_octave = 1.7824e+08

systemdata =

scalar structure containing the fields:

PhysicalMemory =

scalar structure containing the fields:

Available = 2.4954e+09

Total = 4.0284e+09

SystemMemory =

scalar structure containing the fields:

Available = 6.6813e+09

Total = 8.2143e+09

VirtualAddressSpace =

scalar structure containing the fields:

Available = 2.8147e+14

Total = 2.8147e+14

编程注意:此函数仅在Linux和Windows上实现。

参见:computer, getpid, getrusage, nproc, uname。

: procstats = getrusage ()

返回一个结构，其中包含有关当前Octave进程的一些统计信息。

并非所有字段在所有系统上都可用。如果无法获得CPU时间统计信息，则将CPU时隙设置为零。其他缺失的数据由NaN替换。可能的字段列表如下:

idrss

Unshared data size.

inblock

Number of block input operations.

isrss

Unshared stack size.

ixrss

Shared memory size.

majflt

Number of major page faults.

maxrss

Maximum data size.

minflt

Number of minor page faults.

msgrcv

Number of messages received.

msgsnd

Number of messages sent.

nivcsw

Number of involuntary context switches.

nsignals

Number of signals received.

nswap

Number of swaps.

nvcsw

Number of voluntary context switches.

oublock

Number of block output operations.

stime

A structure containing the system CPU time used. The structure has the elements sec (seconds) usec (microseconds).

utime

A structure containing the user CPU time used. The structure has the elements sec (seconds) usec (microseconds).

: value = winqueryreg (rootkey, subkey, valuename)

: value = winqueryreg (rootkey, subkey)

: names = winqueryreg ("name", rootkey, subkey)

从Windows注册表查询名称或值。

在Windows操作系统中，从根键rootkey返回注册表键子键的值。可以使用可选参数valuename指定查询的注册表值的名称。否则，如果只带两个参数调用，或者valuename为空，则返回subkey的默认值。如果注册表值的类型为“REG\_DWORD”，则value的类型为int32。如果值的类型为REG\_SZ或REG\_EXPAND\_SZ，则返回字符串。

如果第一个参数是“name”，则返回一个包含该键值名称的字符串单元格数组。

变量rootkey必须是一个具有有效根键标识符的字符串:

HKCR

HKEY\_CLASSES\_ROOT

HKEY\_CURRENT\_CONFIG

HKCU

HKEY\_CURRENT\_USER

HKLM

HKEY\_LOCAL\_MACHINE

HKU

HKEY\_USERS

HKEY\_PERFORMANCE\_DATA

例子:

按“HKCU\Environment”键获取值名称列表:

valuenames = winqueryreg ("name", "HKEY\_CURRENT\_USER", ...

"Environment");

对于每个valuenames，显示其值:

for k = 1:numel (valuenames)

val = winqueryreg ("HKEY\_CURRENT\_USER", "Environment", ...

valuenames{k});

str = sprintf ("%s = %s", valuenames{k}, num2str (val));

disp (str);

endfor

在非windows平台上，此函数失败并出现错误。

**36.12哈希函数**

通常需要查找两个字符串或文件是否相同。这可以通过逐个字符比较它们并寻找差异来实现。但是，这可能很慢，因此比较字符串或文件的散列可以快速发现文件是否不同。

哈希函数的另一个用途是检查文件完整性。用户可以根据已知值检查文件的哈希值，并查找他们拥有的文件是否与生成原始哈希值的文件相同。

Octave提供哈希函数来计算字符串和文件的哈希值，后者与fileread函数结合使用。哈希函数支持许多常用的哈希方法。

: hashval = hash ("hashfcn", str)

使用哈希函数hashfcn计算字符串str的哈希值。

可用的散列函数如下表所示。

‘MD2’

Message-Digest Algorithm 2 (RFC 1319).

‘MD4’

Message-Digest Algorithm 4 (RFC 1320).

‘MD5’

Message-Digest Algorithm 5 (RFC 1321).

‘SHA1’

Secure Hash Algorithm 1 (RFC 3174)

‘SHA224’

Secure Hash Algorithm 2 (224 Bits, RFC 3874)

‘SHA256’

Secure Hash Algorithm 2 (256 Bits, RFC 6234)

‘SHA384’

Secure Hash Algorithm 2 (384 Bits, RFC 6234)

‘SHA512’

Secure Hash Algorithm 2 (512 Bits, RFC 6234)

例如，要计算字符串“abc”的MD5哈希值，哈希函数调用如下:

hash ("md5", "abc")

-| ans = 900150983cd24fb0d6963f7d28e17f72

对于相同的字符串，SHA-1哈希值计算如下:

hash ("sha1", "abc")

-| ans = a9993e364706816aba3e25717850c26c9cd0d89d

要计算一个文件的哈希值，例如file = "file.txt"，可以结合fileread调用hash:

hash ("md5", fileread (file));

**37包**

由于Octave是自由软件，我们鼓励用户与他人分享他们的程序。为了实现这种共享，Octave支持安装额外的软件包。“Octave Forge”项目是一个社区维护的软件包集，可以下载并安装在Octave中。在写作的时候，“八度锻造”项目可以在https://octave.sourceforge.io上找到，但由于互联网是一个不断变化的地方，这在阅读的时候可能不是真的。因此，建议您查看Octave网站以获取更新的参考。

**37.1安装和拆卸软件包**

假设在image-1.0.0.tar.gz文件中有一个可用的包，可以使用下面的命令从Octave提示符安装它

pkg install image-1.0.0.tar.gz

如果包安装成功，提示符上不会显示任何内容，但如果在安装过程中出现警告或错误，则会报告。通过在pkg install命令后写入几个包文件名，可以一次安装几个包。如果已经安装了不同版本的软件包，则在安装新软件包之前将其删除。这使得升级和降级一个包的版本变得容易，但是不可能同时安装同一个包的几个版本。

要查看安装了哪些包，请键入

pkg list

-| Package Name | Version | Installation directory

-| --------------+---------+-----------------------

-| image \*| 1.0.0 | /home/jwe/octave/image-1.0.0

在此例中，安装的是1.0.0版本的镜像包。包名称旁边的“\*”字符表示图像包已加载并准备使用。

可以像这样使用pkg卸载命令从系统中删除一个包

pkg uninstall image

如果包被成功删除，那么在提示符中不会打印任何内容，但是如果出现警告或错误，则会报告。需要注意的是，用于安装的包文件不需要删除，删除包时只需要使用pkg列表中报告的包名。通过在pkg uninstall命令后写入多个包名，可以一次删除多个包。

为了尽量减少包之间的代码重复量，一个包可能依赖于另一个包。如果一个包依赖于另一个包，它将在安装过程中检查该包是否已安装。如果不是，则会报告错误，并且不会安装包。可以通过向pkg install命令传递-nodeps标志来禁用此行为

pkg install -nodeps my\_package\_with\_dependencies.tar.gz

由于已安装的包期望安装其依赖项，因此可能无法正常工作。因此，不建议禁用依赖项检查。

: pkg command pkg\_name

: pkg command option pkg\_name

: [out1, …] = pkg (command, … )

管理或查询Octave的包(附加功能组)。

包可以全局安装(即，为系统的所有用户)或本地安装(即，仅为当前用户)。

默认情况下，全局包安装在系统范围的位置。这通常是安装Octave本身的文件夹的子目录。因此，Octave需要对该文件夹进行写访问以安装全局包，这通常只有在Octave以管理权限运行时才可用，例如在类unix系统上以root(或超级用户)身份运行时，或者在Windows上以提升的权限(“以管理员身份运行”)运行时。

相反，本地包默认安装在用户的主目录(或Windows上的用户配置文件)中，并且仅对该特定用户可用。通常，可以在没有管理权限的情况下安装它们。

当Octave以管理权限运行时，pkg将默认将包安装到全局包位置。否则，包将默认安装到本地位置。用户可以使用可选参数(-local或-global)覆盖此默认安装位置，如下所述。可以使用pkg前缀查询当前使用的默认软件包安装位置。

对于全局包和本地包，有单独的数据库保存有关已安装包的信息。如果某个包既在本地安装又在全局安装，则本地安装优先于(“阴影”)全局安装。使用哪个(全局或本地)包安装也可以通过使用前缀和/或使用' local\_list '输入参数来操纵。使用这些机制，同一个包的几个不同版本也可以并排安装(但不能同时加载)。

软件包可能依赖于外部软件和/或其他软件包。为了能够安装这些包，应该事先安装这些依赖项。依赖于其他包的包仍然可以使用-nodeps标志安装。对外部软件(如库)不满意的依赖关系的影响取决于单个软件包。

包必须在被使用之前被加载。加载包时，Octave执行以下任务:

1. 如果包依赖于其他包(并且调用pkg load时不带-nodeps选项)，则不会立即加载包。相反，这些依赖项首先被加载(如果需要的话递归加载)。
2. 当所有依赖项都满足时，包的子目录被添加到搜索路径中。

这种加载顺序会导致依赖项提供的函数可能会被顶层包提供的同名函数掩盖。

每次将一个包添加到搜索路径时，如果包提供了该包的初始化脚本，则自动执行该包的初始化脚本。

根据command的值和所请求的返回参数的数量，pkg可用于执行若干任务。command的取值包括:

‘install’

安装命名包。例如,

pkg install image-1.0.0.tar.gz

安装image-1.0.0.tar.gz文件中的包。包含包的文件可以是一个URL，例如:

pkg install 'http://somewebsite.org/image-1.0.0.tar.gz'

安装在给定URL中找到的包。这需要互联网连接和cURL库。

安全风险:安装前未对软件包进行校验。它与从给定的URL手动下载软件包并安装它具有相同的安全问题。

不支持:GNU Octave社区不负责从外部站点安装的软件包。要获得支持或报告错误，您需要直接联系已安装包的维护人员(请参阅包的DESCRIPTION文件)。

option变量可以包含影响软件包安装方式的选项。这些选项可以是一个或多个

-nodeps

包管理器将禁用依赖项检查。有了这个选项，就可以安装一个包，即使它依赖于系统上没有安装的另一个包。请谨慎使用此选项。

-local

强制进行本地安装(仅对当前用户可用的包)，即使使用管理权限运行Octave也是如此。

-global

强制进行全局安装(所有用户都可以使用包)，即使Octave没有以管理权限运行。用户必须具有对全局包存储的写访问权限。

-forge

直接从Octave Forge存储库安装一个包。这需要互联网连接和cURL库。

安全风险:安装前未对软件包进行校验。没有包的签名，也没有校验和来确认下载了正确的文件。它具有与从Octave Forge存储库手动下载软件包并安装它相同的安全问题。

-verbose

包管理器将在执行命令时打印所有命令的输出。

‘update’

根据存储库检查已安装的Octave Forge软件包，并更新任何过时的项目。更新的包是全局安装还是本地安装，这取决于Octave是否以提升的权限运行。这需要互联网连接和cURL库。

可以将install命令的选项和要检查更新的单个包的名称指定为update命令后面的列表。如果指定了-local或-global选项，pkg update将更新检查限制为本地或全局已安装的包，并在相同的上下文中安装更新。例如,

更新所有软件包:

pkg update

更新所有本地软件包:

pkg update -local

更新某些包，忽略依赖项，最大篇幅:

pkg update -verbose -nodeps image signal geometry

多个包的更新按字母顺序排序，不会检查受安装顺序影响的依赖项。如果出现与依赖顺序相关的pkg更新失败，使用pkg update -nodeps忽略依赖项，或者使用pkg install -forge <package\_name>手动更新单个包。

‘uninstall’

卸载命名包。例如,

pkg uninstall image

从系统中删除镜像包。如果另一个安装的包依赖于映像包，则会发出错误。无论如何，可以使用-nodeps选项卸载这个包。

‘load’

将命名包添加到路径中。加载包后，可以使用包提供的功能。例如,

pkg load image

将镜像包添加到路径中。

注意:当加载一个包时，pkg也会自动尝试加载任何未加载的依赖项，除非指定了-nodeps标志。例如,

pkg load signal

添加信号包并尝试加载其依赖项:控制包。请注意，使用-nodeps标志可能会影响所加载包的功能。即使稍后加载必要的依赖项，顶级包的功能仍然会受到影响，因为可能没有遵循最佳加载顺序。

‘unload’

从路径中删除命名包。卸载包后，就不能再使用包提供的功能了。试图卸载其他已加载的软件包仍然依赖的软件包将导致错误;在这种情况下，不会卸载任何包。可以使用-nodeps标志强制删除包，但要注意，依赖包的功能可能会受到影响。与加载包时一样，在使用-nodeps标志卸载依赖包后重新加载依赖包可能无法恢复依赖包的所有功能，因为所需的加载顺序可能不正确。

‘list’

显示当前安装的软件包列表。例如,

pkg list

将生成一个简短的报告，其中包含每个已安装包的包名、版本和安装目录。提供包名称以限制报告到特定的包。例如:

pkg list image

如果请求单个返回参数，则pkg返回一个单元格数组，其中每个元素都是一个包含单个包信息的结构。

installed\_packages = pkg ("list")

如果请求两个输出参数，pkg将已安装的包列表分成由当前用户安装的包和由系统管理员安装的包。

[user\_packages, system\_packages] = pkg ("list")

“-forge”选项列出了Octave Forge存储库中可用的包。这需要互联网连接和cURL库。例如:

oct\_forge\_pkgs = pkg ("list", "-forge")

‘describe’

显示已安装包的简短描述。使用“-verbose”选项还可以列出包提供的函数。例如,

pkg describe -verbose

将描述所有已安装的包及其提供的功能。可以将显示限制为一组包:

## describe control and signal packages

pkg describe control signal

如果请求一个输出，一个包含每个包的描述和函数列表的结构单元将作为输出返回，而不是打印在屏幕上:

desc = pkg ("describe", "secs1d", "image")

如果任何请求的包没有安装，pkg返回一个错误，除非请求第二个输出:

[desc, flag] = pkg ("describe", "secs1d", "image")

flag将为每个命名的包取一个值"Not installed"， "Loaded"或"Not Loaded"。

‘prefix’

设置安装前缀目录。例如,

pkg prefix ~/my\_octave\_packages

将安装前缀设置为~/my\_octave\_packages。包将安装在此目录中。

可以通过请求输出参数来获取当前安装前缀。例如:

pfx = pkg ("prefix")

可以使用附加参数独立指定安装体系结构相关文件的位置。例如:

pkg prefix ~/my\_octave\_packages ~/my\_arch\_dep\_pkgs

‘local\_list’

设置查找本地安装包信息的文件。本地安装的包是那些只对当前用户可用的包。例如:

pkg local\_list ~/.octave\_packages

可以使用以下命令获取local\_list的当前值

pkg local\_list

‘global\_list’

设置查找全局安装包信息的文件。全局安装的包是所有用户都可以使用的包。例如:

pkg global\_list /usr/share/octave/octave\_packages

可以使用以下命令获取global\_list的当前值

pkg global\_list

‘build’

构建一个或多个包的二进制形式。生成的二进制文件本身将是一个Octave包，可以使用pkg正常安装

pkg build builddir image-1.0.0.tar.gz …

其中builddir是生成临时安装并找到二进制包的目录名。忽略-verbose和-nodeps选项，而忽略所有其他选项。

‘rebuild’

从安装的目录重新构建包数据库。这可以在包数据库损坏的情况下使用。

‘test’

执行指定包提供的所有函数中包含的内置自我测试。例如:

pkg test image

参见:ver, news。

**37.2使用软件包**

默认情况下，安装的包不能从Octave提示符中获得，但可以使用pkg load和pkg unload命令来控制这一点。可以通过输入命令将包中的函数添加到Octave路径中

pkg load package\_name

其中package\_name是要添加到路径中的包的名称。

与此类似，可以通过输入命令从Octave路径中删除包

pkg unload package\_name

**37.3管理包**

可以对每个用户(本地)和系统范围(全局)安装一个包。如果执行安装的用户是root用户(或者在Windows上具有更高权限的管理员)，那么包默认安装在系统范围的目录中，默认为OCTAVE\_HOME/share/octave/packages/。如果用户不是根用户(或者Octave在没有提升权限的情况下运行)，包将在本地安装。本地包默认安装目录为“user\_data\_dir/octave/OCTAVE\_API\_VERSION/packages”。包将安装在安装目录的子目录中，该子目录将以包命名。可以使用pkg前缀命令更改安装目录:

pkg prefix new\_installation\_directory

当前安装目录可以通过键入

current\_installation\_directory = pkg ("prefix")

包管理器在配置文件中存储有关已安装包的一些信息。对于每个用户(本地)包，默认情况下，此信息存储在文件user\_config\_dir/octave/OCTAVE\_API\_VERSION/octave\_packages中。对于系统范围(全局)安装，它存储在OCTAVE\_HOME/share/octave/octave\_packages中。每个用户文件的路径可以用pkg local\_list命令修改:

pkg local\_list /path/to/new\_file

对于系统范围的安装，可以使用pkg global\_list命令以相同的方式进行更改。如果在没有新路径的情况下调用这些命令，则将返回当前路径。要在会话之间保留这些设置，可以在其中一个启动文件中进行设置，请参见启动文件。

**37.4创建包**

在内部，包只是一个压缩过的tar文件，其中包含任意给定名称的顶级目录。这个目录将在下面被称为包，可能包含以下文件:

package/CITATION

这是一个可选文件，描述了如何引用发布包的说明。它将由函数引用逐字显示。

package/COPYING

这是一个必需的文件，包含包的许可证。一般来说，对许可证没有任何限制。但是，如果软件包包含动态链接的函数，则许可证必须与GNU通用公共许可证兼容。

package/DESCRIPTION

这是一个必需的文件，包含有关包的信息。有关该文件的详细信息，请参见DESCRIPTION文件。

package/ChangeLog

这是一个可选文件，描述对包源文件所做的所有更改。

package/INDEX

这是一个可选文件，描述包提供的函数。如果没有给出此文件，则将从包中的函数和DESCRIPTION文件中的Categories关键字自动创建一个。有关该文件的详细信息，请参阅INDEX文件。

package/NEWS

这是一个可选文件，描述了所有用户可见的值得一提的更改。随着文件大小的增加，旧的条目可以移动到包/ONEWS中。

package/ONEWS

这是一个可选文件，描述来自NEWS文件的旧条目。

package/PKG\_ADD

一个可选文件，其中包含在将包添加到用户路径时运行的命令。注意，包源代码中的PKG\_ADD指令也将由Octave包管理器添加到该文件中。注意要避免在包中使用符号链接，因为符号链接在某些文件系统上不存在，因此该文件的典型用途是替换符号链接

ln -s foo.oct bar.oct

使用自动加载指令，比如

autoload ('bar', which ('foo'));

有关PKG\_ADD指令的详细信息，请参见PKG\_ADD和PKG\_DEL指令。

package/PKG\_DEL

一个可选文件，其中包含从用户路径中删除包时运行的命令。注意，包源代码中的PKG\_DEL指令也将由Octave包管理器添加到该文件中。有关PKG\_DEL指令的详细信息，请参见PKG\_ADD和PKG\_DEL指令。

package/pre\_install.m

这是一个可选的函数，在安装包之前运行。这个函数调用时只有一个参数，一个结构体，在DESCRIPTION中的数据后面加上字段名，以及将安装包函数的路径。

package/post\_install.m

这是一个可选函数，在安装包之后运行。调用这个函数时只有一个参数、一个结构体，在DESCRIPTION中的数据后面加上字段名，以及安装包函数的路径。

package/on\_uninstall.m

这是一个可选函数，在删除包之前运行。这个函数调用时只有一个参数，一个结构体，在DESCRIPTION中的数据后面加上字段名，安装包函数的路径，以及当前是否加载包。

除了上面提到的文件，一个包还可以包含以下一个或多个目录:

package/inst

一个可选目录，包含包直接安装的所有文件。通常，这将包括所有m文件。

package/src

一个可选目录，其中包含在包安装之前必须构建的代码。如果这个脚本存在，Octave包管理器将在这个目录中执行。/configure，如果这个目录中存在一个Makefile文件，那么Octave包管理器将调用make。Make install将不会被调用。当调用configure和make时，环境变量MKOCTFILE、OCTAVE\_CONFIG和OCTAVE将分别被设置为正确版本的MKOCTFILE、OCTAVE -config和OCTAVE程序的完整路径。如果一个名为FILES的文件存在，那么列出的所有文件都将被复制到inst目录，因此它们也将被安装。如果FILES文件不存在，src/\*。M和src/\*。Oct将被复制到inst目录。

package/doc

包含包文档的可选目录。此目录下的文件将直接安装在已安装包的子目录中，以备将来参考。

package/bin

一个可选目录，其中的文件将在加载包时添加到Octave EXEC\_PATH中。这可能包含外部脚本等，由包内的函数调用。

**37.4.1描述文件**

DESCRIPTION文件包含有关包的各种信息，例如包的名称、作者和版本。这个文件的格式非常简单

* 以“#”开头的行是注释。
* 以空白字符开头的行是前一行的延续。
* 其他的都是NameOfOption: ValueOfOption的形式。

下面是DESCRIPTION文件的一个简单示例

Name: The name of my package

Version: 1.0.0

Date: 2007-18-04

Author: The name (and possibly email) of the package author.

Maintainer: The name (and possibly email) of the current

package maintainer.

Title: The title of the package

Description: A short description of the package. If this

description gets too long for one line it can continue

on the next by adding a space to the beginning of the

following lines.

License: GPLv3+

包管理器当前识别以下关键字

Name

Name of the package.

Version

Version of the package. A package version is typically digits separated by dots but may also contain ‘+’, ‘-’, ‘~’, and alphanumeric characters (in the "C" locale). For example, "2.1.0+" could indicate a development version of a package. Versions are compared using compare\_versions.

Date

Date of last update.

Author

Original author of the package.

Maintainer

Maintainer of the package.

Title

A one line description of the package.

Description

A one paragraph description of the package.

Categories

Optional keyword describing the package (if no INDEX file is given this is mandatory).

Problems

Optional list of known problems.

Url

Optional list of homepages related to the package.

Depends

A list of other Octave packages that this package depends on. This can include dependencies on particular versions, with the following format:

Depends: package (>= 1.0.0)

Possible operators are <, <=, ==, >= or >. If the part of the dependency in () is missing, any version of the package is acceptable. Multiple dependencies can be defined as a comma separated list. This can be used to define a range of versions of a particular package:

Depends: package (>= 1.0.0), package (< 1.5.0)

It is also possible to depend on particular versions of Octave core:

Depends: octave (>= 3.8.0)

License

使用许可证的可选简短描述(例如，GPL版本3或更新版本)。这是可选的，因为文件复制是强制性的。

SystemRequirements

这些是包的外部安装依赖项，包管理器不会检查它们。这是对发行包程序的一个提示。它们遵循与Depends关键字相同的约定。

BuildRequires

这些是包的外部构建依赖项，包管理器不会检查它们。这是对发行包程序的一个提示。它们遵循与Depends关键字相同的约定。请注意，通常情况下，诸如rpm或deb之类的打包系统会自动从构建依赖中探测安装依赖，因此buildrerequires依赖通常会消除对SystemRequirements依赖的需要。

开发人员可以根据自己的目的自由地向DESCRIPTION文件添加其他参数。进一步帮助打包器的一个细节是SystemRequirements和buildrerequires关键字可以有一个依赖于发行版的部分，并且自动构建过程将使用它们。格式的一个例子是

BuildRequires: libtermcap-devel [Mandriva] libtermcap2-devel

其中第一个包名将被用作默认值，如果rpm是在Mandriva发行版上构建的，那么将使用第二个包名来代替。

**37.4.2 INDEX文件**

可选的INDEX文件提供包中函数的分类视图。这个文件的格式非常简单

* 以“#”开头的行是注释。
* 第一个非注释行应该是这样的

toolbox >> Toolbox name

* 以字母字符开头的行表示函数的新类别。
* 以空格字符开头的行表示该行上的函数名属于最后提到的类别。

该格式可以用以下示例进行总结:

# A comment

toolbox >> Toolbox name

Category Name 1

function1 function2 function3

function4

Category Name 2

function2 function5

如果您希望引用用户可能希望在您的包中找到但不存在的函数，提供解决方法或指出该函数在其他地方可用，您可以使用:

fn = workaround description

当使用pkg description列出包中的函数时，此解决方法描述将不会出现，但它们将在HTML文档中在线发布。变通描述可以使用任何HTML标记，但请记住，它将被括在粗体环境中。的特殊情况:

fn = use <code>alternate expression</code>

粗体斜体将被自动抑制。即使在引用中也需要使用<code>:

fn = use <a href="someothersite.html"><code>fn</code></a>

有时函数只是部分兼容，在这种情况下，您可以单独列出不兼容的情况。要引用包中的另一个函数，使用<f>fn</f>。例如:

eig (a, b) = use <f>qz</f>

由于站点可能有许多缺失的功能，您可以定义宏，而不是一遍又一遍地键入相同的链接。

$id = expansion

定义宏id。你可以在描述的任何地方使用$id，它将被扩展。例如:

$TSA = see <a href="link\_to\_spctools">SPC Tools</a>

arcov = $TSA <code>armcv</code>

Id是由字母、数字和\_组成的任意字符串。

**37.4.3 PKG\_ADD和PKG\_DEL指令**

如果包包含称为PKG\_ADD或PKG\_DEL的文件，则在将包添加或从用户路径中删除时将执行这些文件中的命令。在某些情况下，这样的文件维护起来有点麻烦，因此包管理器支持自动创建这样的文件。如果包中的源文件包含PKG\_ADD或PKG\_DEL指令，它们将被添加到PKG\_ADD或PKG\_DEL文件中。

在m-files中，PKG\_ADD指令看起来像这样

## PKG\_ADD: some\_octave\_command

这样的行应该添加在function关键字之前。在c++文件中，PKG\_ADD指令是这样的

// PKG\_ADD: some\_octave\_command

在这两种情况下，some\_octave\_command都应该替换为应该放在PKG\_ADD文件中的命令。PKG\_DEL指令以相同的方式工作，除了PKG\_ADD关键字被PKG\_DEL替换，并且命令被添加到PKG\_DEL文件中。

**37.4.4部件丢失**

如果一个包依赖于一个组件，比如另一个Octave包，而这个组件可能不存在，那么安装一个函数告诉用户当某个特定组件丢失时该怎么做可能是有用的。该函数必须由包维护者编写，并使用missing\_component\_hook在Octave中注册。

: val = missing\_component\_hook ()

: old\_val = missing\_component\_hook (new\_val)

: old\_val = missing\_component\_hook (new\_val, "local")

查询或设置内部变量，该变量指定在缺少Octave的组件时要调用的函数。

这对于可能将Octave安装拆分为多个子包的打包程序非常有用，例如，向用户提供如何安装缺失组件的提示。

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。

钩子函数的形式应该是

Octave将使用需要该组件的函数名和描述缺失组件的字符串调用fcn。钩子函数应该返回一条要显示的错误消息。

参见:missing\_function\_hook。

**附录A外部代码接口**

“人类智慧的总和不包含在任何一种语言中”

——埃兹拉·庞德

对于解决科学和工程中的许多问题，Octave是一种奇妙的语言。然而，它不是唯一的计算机语言，有时您可能想使用用其他语言编写的代码。这样做的好理由包括:1)不要重复发明轮子;现有的经过彻底测试和调试的函数库或大规模模拟代码库就是一个很好的例子，2)访问不同语言的独特功能;例如Perl中众所周知的正则表达式函数(但不要这样做，因为在Octave中已经存在regexp)。

性能通常不应该成为使用编译扩展的原因。虽然编译后的扩展可以运行得更快，特别是如果它们替换了Octave代码中的循环，但这几乎不是最好的选择。首先，有许多技术可以在保持语言不变的情况下加速Octave的性能。其次，Octave是一种高级语言，可以很容易地执行常见的数学任务。放弃这一点意味着将焦点从解决实际问题转移到解决计算机编程问题上。这意味着返回到低级结构，如指针、内存管理、数学溢出/下溢等。由于其底层特性，以及编译后的代码在Octave外部执行的事实，很有可能导致解释器崩溃并失去工作。

在继续之前，您应该首先确定是否真的需要在Octave之外编写代码。

* 我可以单独使用Octave脚本语言获得相同的功能吗?

即使函数已经存在于语言之外，在m文件中简单地重现该行为也比尝试与外部代码接口要好。

* 代码是否完全优化了Octave?

如果性能是一个问题，您应该始终从获得更好性能的语言内技术开始。其中最主要的是向量化(参见向量化和更快的代码执行)，它不仅使代码简洁易懂，而且提高了性能(10X-100X)。如果必须使用循环，请确保变量的空间分配是在循环之外进行的，使用对大小合适的矩阵或零的赋值。

* 代码是否尽可能多地使用现有的内置库例程?

这些例程经过了高度优化，许多例程不需要进行解释。

* 相对于留在Octave中，编写一个动态链接的函数是否代表了对时间的有效投资?

学习Octave的外部代码接口需要时间，并且不可避免地会出现编译器等工具的问题。

也就是说，Octave提供了一个通用的接口，可以将编译后的代码块作为动态链接的扩展包含在内。这些动态链接的函数可以从解释器中以与任何普通函数相同的方式调用。接口是双向的，外部代码可以调用Octave函数(如plot)，否则可能很难开发。

该接口主要支持c++、C和Fortran语言。Octave本身是用c++编写的，可以通过其本机oct-file接口调用外部c++ /C代码。为了与MATLAB兼容，还通过mex-file接口支持C语言。Fortran代码最容易通过oct-file接口访问。

由于许多其他语言提供C或c++ api，因此在Octave和其他语言之间构建桥梁相对简单。这也是一种连接硬件资源的方法，这些硬件资源通常有用C编写的设备驱动程序。

**A.1 Oct-Files**

**A.1.1开始使用oct文件**

oct文件是用Octave API编译成可动态加载对象的c++代码片段。它们的名称取自包含扩展名为.oct的对象的文件。

找到一个c++编译器，使用正确的开关，为头文件添加正确的include路径，等等都是一项艰巨的任务。Octave通过提供mkoctfile命令来自动执行此操作，该命令用于构建oct文件。该命令可从Octave中或在shell命令行中使用。

: mkoctfile [-options] file …

: [output, status] = mkoctfile (…)

mkoctfile函数编译用C、c++或Fortran编写的源代码。根据与mkoctfile一起使用的选项，编译后的代码可以在Octave中调用，也可以作为独立的应用程序使用。

mkoctfile可以从shell提示符或Octave提示符中调用。从Octave提示符调用它只是将调用委托给shell提示符。任何输出都存储在输出变量中，退出状态存储在状态变量中。如果调用时没有输出并且编译失败，那么Octave将发出一个错误。但是，如果程序员请求输出或状态，Octave只会发出警告，程序员有责任验证命令是否成功。

mkoctfile接受以下选项，除了要编译的代码的文件名外，所有选项都是可选的:

‘-I DIR’

Add the include directory DIR to compile commands.

‘-D DEF’

Add the definition DEF to the compiler call.

‘-l LIB’

Add the library LIB to the link command.

‘-L DIR’

Add the library directory DIR to the link command.

‘-M’

‘--depend’

Generate dependency files (.d) for C and C++ source files.

‘-R DIR’

Add the run-time path to the link command.

‘-Wl,…’

Pass options to the linker like "-Wl,-rpath=…". The quotes are needed since commas are interpreted as command separators.

‘-W…’

Pass options to the assembler like "-Wa,OPTION".

‘-c’

Compile but do not link.

‘-g’

Enable debugging options for compilers.

‘-o FILE’

‘--output FILE’

Output filename. Default extension is .oct (or .mex if ‘--mex’ is specified) unless linking a stand-alone executable.

‘-p VAR’

‘--print VAR’

Print configuration variable VAR. There are three categories of variables:

Octave configuration variables that users may override with environment variables. These are used in commands that mkoctfile executes.

ALL\_CFLAGS INCLUDEDIR

ALL\_CXXFLAGS LAPACK\_LIBS

ALL\_FFLAGS LDFLAGS

ALL\_LDFLAGS LD\_STATIC\_FLAG

BLAS\_LIBS LFLAGS

CC LIBDIR

CFLAGS LIBOCTAVE

CPICFLAG LIBOCTINTERP

CPPFLAGS OCTAVE\_LINK\_OPTS

CXX OCTINCLUDEDIR

CXXFLAGS OCTAVE\_LIBS

CXXLD OCTAVE\_LINK\_DEPS

CXXPICFLAG OCTLIBDIR

DL\_LDFLAGS OCT\_LINK\_DEPS

F77 OCT\_LINK\_OPTS

F77\_INTEGER8\_FLAG RDYNAMIC\_FLAG

FFLAGS SPECIAL\_MATH\_LIB

FPICFLAG XTRA\_CFLAGS

INCFLAGS XTRA\_CXXFLAGS

Octave配置变量如上所述，但目前未被mkoctfile使用。

AR

DEPEND\_EXTRA\_SED\_PATTERN

DEPEND\_FLAGS

FFTW3F\_LDFLAGS

FFTW3F\_LIBS

FFTW3\_LDFLAGS

FFTW3\_LIBS

FFTW\_LIBS

FLIBS

LIBS

RANLIB

READLINE\_LIBS

提供的八度配置变量仅供参考。除了' OCTAVE\_HOME '和' OCTAVE\_EXEC\_HOME '，用户不能覆盖这些变量。

如果在环境中设置了OCTAVE\_HOME或OCTAVE\_EXEC\_HOME，那么将相应地调整其他变量，将OCTAVE\_HOME或OCTAVE\_EXEC\_HOME替换为配置Octave时使用的——prefix或——exec-prefix选项指定的目录的原始值。

API\_VERSION LOCALFCNFILEDIR

ARCHLIBDIR LOCALOCTFILEDIR

BINDIR LOCALSTARTUPFILEDIR

CANONICAL\_HOST\_TYPE LOCALVERARCHLIBDIR

DATADIR LOCALVERFCNFILEDIR

DATAROOTDIR LOCALVEROCTFILEDIR

DEFAULT\_PAGER MAN1DIR

EXEC\_PREFIX MAN1EXT

EXEEXT MANDIR

FCNFILEDIR OCTAVE\_EXEC\_HOME

IMAGEDIR OCTAVE\_HOME

INFODIR OCTAVE\_VERSION

INFOFILE OCTDATADIR

LIBEXECDIR OCTDOCDIR

LOCALAPIARCHLIBDIR OCTFILEDIR

LOCALAPIFCNFILEDIR OCTFONTSDIR

LOCALAPIOCTFILEDIR STARTUPFILEDIR

LOCALARCHLIBDIR

‘--link-stand-alone’

Link a stand-alone executable file.

‘--mex’

Assume creation of a MEX file. Set the default output extension to .mex.

‘-s’

‘--strip’

Strip the output file.

‘-v’

‘--verbose’

Echo commands as they are executed.

‘file’

The file to compile or link. Recognized file types are:

.c C source

.cc C++ source

.cp C++ source

.cpp C++ source

.CPP C++ source

.cxx C++ source

.c++ C++ source

.C C++ source

.f Fortran source (fixed form)

.F Fortran source (fixed form)

.f90 Fortran source (free form)

.F90 Fortran source (free form)

.o object file

.a library file

考虑下面这个简短的例子，它介绍了编写一个可以链接到Octave的c++函数的基础知识。

#include <octave/oct.h>

DEFUN\_DLD (helloworld, args, nargout,

"Hello World Help String")

{

octave\_stdout << "Hello World has "

<< args.length () << " input arguments and "

<< nargout << " output arguments.\n";

// Return empty matrices for any outputs

octave\_value\_list retval (nargout);

for (int i = 0; i < nargout; i++)

retval(i) = octave\_value (Matrix ());

return retval;

}

第一个关键行是#include <octave/oct.h>，它提供了c++ oct文件所需的大多数定义。注意，octave/oct.h是c++头文件，不能直接在C源文件中#include '，也不能在任何其他语言中#include '。

oct.h包含了宏DEFUN\_DLD的定义，它创建了一个动态加载的函数。这个宏有四个参数:

1. 在Octave中看到的函数名，
2. octave\_value\_list类型函数的参数列表，
3. 输出参数的数量，如果不使用，可以(通常)省略
4. 用于函数的帮助文本的字符串。

用DEFUN\_DLD定义的函数的返回类型总是octave\_value\_list。

在选择函数名时，有几个重要的考虑因素。首先，它必须是一个有效的Octave函数名，因此必须是由字母、数字和下划线组成的序列，且不能以数字开头。其次，由于Octave使用函数名来定义它试图在其中查找函数的文件名，因此DEFUN\_DLD宏中的函数名必须与oct文件的文件名匹配。因此，上面的函数应该在文件helloworld中。Cc，并将使用该命令编译为oct文件

mkoctfile helloworld.cc

这将创建一个名为helloworld的文件。这是函数的编译版本。应该注意的是，在一个源文件中有多个DEFUN\_DLD函数是完全可以接受的。但是，对于源代码中使用DEFUN\_DLD宏定义的每个函数，必须有一个指向oct-文件的符号链接，或者应该使用自动加载(函数文件)函数。

函数的其余部分展示了如何查找输入参数的数量，如何通过Octave分页器打印，以及如何从函数返回。像上面那样编译这个函数之后，使用它的一个例子是

helloworld (1, 2, 3)

-| Hello World has 3 input arguments and 0 output arguments.

接下来的部分将展示如何使用Octave核心内部的特定类。像dMatrix这样的基类(双精度矩阵)可以在目录libocave /array中找到。关于如何使用特定类的权威参考是头文件本身。然而，为了能够使用一个类，通常只需学习手册中的示例就足够了。

**A.1.2 oct文件中的矩阵和数组**

Octave支持许多不同的数组和矩阵类，其中大多数是基于array类的。例外是下面单独讨论的稀疏矩阵类型。有三种基本的矩阵类型:

Matrix

在dMatrix.h中定义的双精度矩阵类

ComplexMatrix

在CMatrix.h中定义的一个复杂矩阵类

BoolMatrix

在boolMatrix.h中定义的布尔矩阵类

这些是Octave的基本二维矩阵类型。此外还有许多多维数组类型包括

NDArray

在dnarray .h中定义的双精度数组类

ComplexNDarray

在cnarray .h中定义的复杂数组类

boolNDArray

在boolnarray .h中定义的布尔数组类

int8NDArray

int16NDArray

int32NDArray

int64NDArray

在int8narray .h, int16narray .h等中定义的8,16,32和64位有符号数组类。

uint8NDArray

uint16NDArray

uint32NDArray

uint64NDArray

在uint8narray .h、uint16narray .h等中定义的8、16、32和64位无符号数组类。

构造矩阵或多维数组有几种基本方法。使用类矩阵作为一个例子可以

* 用空构造函数创建一个空矩阵或数组。例如:

Matrix a;

这可以用于所有矩阵和数组类型。

* 使用dim\_vector定义矩阵或数组的维度，该向量与从size返回的向量具有相同的特征。例如:

dim\_vector dv (2, 3); // 2 rows, 3 columns

Matrix a (dv);

这可以用于所有矩阵和数组类型。

* 定义矩阵中的行数和列数。例如:

Matrix a (2, 2)

此构造函数只能用于矩阵类型。

这些类型都共享一些基本方法和操作符。许多类似于存在于解释器中的函数。一些有用的方法包括

Method: T& operator () (octave\_idx\_type)

Method: T& elem (octave\_idx\_type)

()运算符或elem方法允许读取或设置矩阵或数组的值。这些方法接受一个octave\_idx\_type类型的参数，即矩阵或数组的索引。此外，矩阵类型允许()操作符和elem方法的两个参数版本，给出要获取或设置的值的行和列索引。

请注意，这些函数进行重要的错误检查，因此在某些情况下，用户可能更喜欢通过下面讨论的fortran\_vec方法直接访问数组或矩阵的数据。

Method: octave\_idx\_type numel () const

The total number of elements in the matrix or array.

Method: size\_t byte\_size () const

The number of bytes used to store the matrix or array.

Method: dim\_vector dims () const

The dimensions of the matrix or array in value of type dim\_vector.

Method: int ndims () const

The number of dimensions of the matrix or array. Matrices are always 2-D, but arrays can be N-dimensional.

Method: void resize (const dim\_vector&)

Method: void resize (nrows, ncols)

A method taking either an argument of type dim\_vector, or, in the case of a matrix, two arguments of type octave\_idx\_type defining the number of rows and columns in the matrix.

Method: T\* fortran\_vec ()

This method returns a pointer to the underlying data of the matrix or array so that it can be manipulated directly, either within Octave or by an external library.

像+、-或\*这样的操作符可以用于大多数矩阵和数组类型。此外，还有一些只对矩阵感兴趣的方法，如转置，厄米特，求解等。

从DEFUN\_DLD函数的输入参数中提取矩阵或数组的典型方法如下

#include <octave/oct.h>

DEFUN\_DLD (addtwomatrices, args, , "Add A to B")

{

if (args.length () != 2)

print\_usage ();

NDArray A = args(0).array\_value ();

NDArray B = args(1).array\_value ();

return octave\_value (A + B);

}

为了避免分割错误导致Octave中止，此函数在访问这些参数之前显式检查是否有足够的参数可用。然后，它获得两个NDArray类型的多维数组，并将它们相加。注意，调用array\_value方法时不使用is\_matrix\_type方法。如果在尝试提取该值时发生错误，Octave将打印一条消息并抛出异常。选择这种编码结构的原因是，参数可能不是NDArray的类型，但将它们转换为NDArray是有意义的。array\_value方法允许在可能的情况下透明地执行这种转换。如果需要捕获这样的错误，并执行某种清理或其他操作，可以捕获octave\_execution\_error异常。

A + B，对两个NDArray对象进行操作，返回一个NDArray，该NDArray在函数返回时被强制转换为octave\_value。使用此演示函数的一个示例是

addtwomatrices (ones (2, 2), eye (2, 2))

⇒ 2 1

1 2

一个基本矩阵和数组类型的列表,从octave\_value提取这些的方法,以及下面列出的关联头文件。

| **Type** | **Function** | **Source Code** |
| --- | --- | --- |
| *RowVector* | *row\_vector\_value* | *dRowVector.h* |
| *ComplexRowVector* | *complex\_row\_vector\_value* | *CRowVector.h* |
| *ColumnVector* | *column\_vector\_value* | *dColVector.h* |
| *ComplexColumnVector* | *complex\_column\_vector\_value* | *CColVector.h* |
| *Matrix* | *matrix\_value* | *dMatrix.h* |
| *ComplexMatrix* | *complex\_matrix\_value* | *CMatrix.h* |
| *boolMatrix* | *bool\_matrix\_value* | *boolMatrix.h* |
| *charMatrix* | *char\_matrix\_value* | *chMatrix.h* |
| *NDArray* | *array\_value* | *dNDArray.h* |
| *ComplexNDArray* | *complex\_array\_value* | *CNDArray.h* |
| *boolNDArray* | *bool\_array\_value* | *boolNDArray.h* |
| *charNDArray* | *char\_array\_value* | *charNDArray.h* |
| *int8NDArray* | *int8\_array\_value* | *int8NDArray.h* |
| *int16NDArray* | *int16\_array\_value* | *int16NDArray.h* |
| *int32NDArray* | *int32\_array\_value* | *int32NDArray.h* |
| *int64NDArray* | *int64\_array\_value* | *int64NDArray.h* |
| *uint8NDArray* | *uint8\_array\_value* | *uint8NDArray.h* |
| *uint16NDArray* | *uint16\_array\_value* | *uint16NDArray.h* |
| *uint32NDArray* | *uint32\_array\_value* | *uint32NDArray.h* |
| *uint64NDArray* | *uint64\_array\_value* | *uint64NDArray.h* |

**A.1.3 oct文件中的字符串**

Octave中的字符串只是一个特殊的Array类。考虑一下这个例子:

#include <octave/oct.h>

DEFUN\_DLD (stringdemo, args, , "String Demo")

{

if (args.length () != 1)

print\_usage ();

octave\_value\_list retval;

charMatrix ch = args(0).char\_matrix\_value ();

retval(1) = octave\_value (ch, '\''); // Single Quote String

octave\_idx\_type nr = ch.rows ();

for (octave\_idx\_type i = 0; i < nr / 2; i++)

{

std::string tmp = ch.row\_as\_string (i);

ch.insert (ch.row\_as\_string (nr-i-1).c\_str (), i, 0);

ch.insert (tmp.c\_str (), nr-i-1, 0);

}

retval(0) = octave\_value (ch, '"'); // Double Quote String

return retval;

}

使用此函数的一个示例是

s0 = ["First String"; "Second String"];

[s1,s2] = stringdemo (s0)

⇒ s1 = Second String

First String

⇒ s2 = First String

Second String

typeinfo (s2)

⇒ sq\_string

typeinfo (s1)

⇒ string

Octave中字符串的另一个复杂之处是单引号和双引号字符串之间的区别。要查找octave\_value是否包含单引号或双引号字符串，请使用下面所示的谓词测试之一。

if (args(0).is\_sq\_string ())

octave\_stdout << "First argument is a single quoted string\n";

else if (args(0).is\_dq\_string ())

octave\_stdout << "First argument is a double quoted string\n";

但是请注意，这两种类型的字符串都由charNDArray类型表示，因此在赋值给octave\_value时，应该指定字符串的类型。例如:

octave\_value\_list retval;

charNDArray ch;

…

// Create single quoted string

retval(1) = octave\_value (ch); // default constructor is sq\_string

OR

retval(1) = octave\_value (ch, '\''); // explicitly create sq\_string

// Create a double quoted string

retval(0) = octave\_value (ch, '"');

**A.1.4 oct文件中的单元格数组**

Octave的单元格类型也可以从oct文件中获得。单元格数组只是一个octave\_values的数组，因此单元格数组的每个元素都可以像任何其他octave\_value一样被处理。一个简单的例子是

#include <octave/oct.h>

#include <octave/Cell.h>

DEFUN\_DLD (celldemo, args, , "Cell Demo")

{

if (args.length () != 1)

print\_usage ();

Cell c = args(0).cell\_value ();

octave\_value\_list retval;

retval.resize (c.numel ()); // faster code by pre-declaring size

for (octave\_idx\_type i = 0; i < c.numel (); i++)

{

retval(i) = c(i); // using operator syntax

//retval(i) = c.elem (i); // using method syntax

}

return retval;

}

注意，单元格数组在标准oct-文件中使用的频率较低，因此必须显式包含cell .h头文件。示例的其余部分依次从单元格数组中提取octave\_values，并将它们作为单独的输出参数返回。例如:

[b1, b2, b3] = celldemo ({1, [1, 2], "test"})

⇒

b1 = 1

b2 =

1 2

b3 = test

**A.1.5 oct文件中的结构**

Octave中的结构是表示的许多字段及其值之间的映射。使用标准模板库映射类，其对由std::string和Octave Cell变量组成。

演示在oct-文件中使用结构的简单示例如下

#include <octave/oct.h>

#include <octave/ov-struct.h>

DEFUN\_DLD (structdemo, args, , "Struct Demo")

{

if (args.length () != 2)

print\_usage ();

if (! args(0).isstruct ())

error ("structdemo: ARG1 must be a struct");

octave\_scalar\_map arg0 = args(0).scalar\_map\_value ();

//octave\_map arg0 = args(0).map\_value ();

if (! args(1).is\_string ())

error ("structdemo: ARG2 must be a character string");

std::string arg1 = args(1).string\_value ();

octave\_value tmp = arg0.contents (arg1);

//octave\_value tmp = arg0.contents (arg1)(0);

if (! tmp.is\_defined ())

error ("structdemo: struct does not have a field named '%s'\n",

arg1.c\_str ());

octave\_scalar\_map st;

st.assign ("selected", tmp);

return octave\_value (st);

}

使用它的一个例子是

x.a = 1; x.b = "test"; x.c = [1, 2];

structdemo (x, "b")

⇒ selected = test

上面的例子特别使用了octave\_scalar\_map类，它用于表示单个结构。对于结构数组，则使用octave\_map类。注释的代码显示了如何修改演示以处理结构数组。在这种情况下，contents方法返回一个可能有多个元素的Cell。因此，要在单个结构体示例中获得底层的octave\_value，我们可以这样写

octave\_value tmp = arg0.contents (arg1)(0);

其中末尾的(0)是Cell对象上的()操作符。如果这是一个包含多个元素的真正结构数组，则可以使用()操作符遍历这些元素。

结构体是一种相对复杂的数据容器，在oct-map.h中有更多可用的函数，这使得使用结构体编码比仅依赖内容更容易。

**A.1.6 oct文件中的稀疏矩阵**

有三类用户感兴趣的稀疏对象。

SparseMatrix

双精度稀疏矩阵类

SparseComplexMatrix

一个复稀疏矩阵类

SparseBoolMatrix

布尔稀疏矩阵类

所有这些类都继承自Sparse<T>模板类，因此它们都具有类似的功能和用法。稀疏<T>类是基于Octave的Array<T>类的，熟悉Octave的Array类的用户将会对稀疏类的使用感到舒适。

由于稀疏类与现有Array类的相似性，本节将不完全描述稀疏类。然而，由于稀疏对象的性质，存在一些差异，这些将被描述。首先，虽然拥有n维稀疏对象从根本上是可能的，但Octave稀疏类目前不允许它们;稀疏类的所有实例都必须是二维的。这意味着SparseMatrix实际上更类似于Octave的Matrix类，而不是NDArray类。

**A.1.6.1数组和稀疏类的区别**

稀疏矩阵中元素的个数被认为是非零元素的个数，而不是维数的乘积。因此,

SparseMatrix sm;

…

int nnz = sm.nelem ();

返回非零元素的个数(类似于解释器函数nnz)。如果用户确实需要矩阵中元素的数量，包括非零元素，他们应该使用numel而不是nelem。请注意，对于非常大的矩阵，其中两个维度的乘积大于unsigned int的表示，numel可能会溢出。一个例子是speye (1e6)，它将创建一个包含一百万行和一百万列的矩阵，但只有一百万个非零元素。在这种情况下，行数乘以列数大于unsigned 32位int所能表示的最大值的200倍。因此，应该避免使用numel，除非知道它不会溢出。

当使用执行基本相同功能的elem方法或()运算符时，也需要非常小心。原因是，如果一个稀疏对象是非const的，那么Octave将假设对稀疏矩阵中的零元素的请求实际上是创建该元素的请求，以便可以填充该元素。因此，一段代码就像

SparseMatrix sm;

…

for (int j = 0; j < nc; j++)

for (int i = 0; i < nr; i++)

std::cerr << " (" << i << "," << j << "): " << sm(i,j) << "\n";

是将稀疏矩阵转化为密集矩阵的好方法，但速度很慢，因为它为矩阵中的每个零元素重新分配稀疏对象。

防止上述情况发生的一种简单方法是创建稀疏矩阵的临时常数版本。请注意，只复制稀疏矩阵的容器，而数据的实际表示将在两个版本的稀疏矩阵之间共享;这不是一个昂贵的操作。上面的例子，重新编写以防止稀疏到密集的转换，是

SparseMatrix sm;

…

const SparseMatrix tmp (sm);

for (int j = 0; j < nc; j++)

for (int i = 0; i < nr; i++)

std::cerr << " (" << i << "," << j << "): " << tmp(i,j) << "\n";

最后，因为稀疏类型不是由连续的内存块表示的，所以Array<T>的fortran\_vec方法不可用。但是，它被三个独立的方法ridx、cidx和data所取代，它们访问存储Octave稀疏矩阵的原始压缩列格式。这些方法可以以类似于element的方式使用，以允许访问或填充矩阵。然而，这取决于用户是否遵守稀疏矩阵压缩列格式，否则矩阵将被损坏。

**A.1.6.2在oct文件中创建稀疏矩阵**

创建稀疏矩阵有两种有用的策略。第一种方法是创建三个向量，表示行索引、列索引和数据值，并从这些向量创建矩阵。第二种选择是创建一个具有适当空间的稀疏矩阵，然后填充值。这两种技术各有优缺点。

下面是使用第一种技术创建一个小型稀疏矩阵的示例

int nz, nr, nc;

nz = 4, nr = 3, nc = 4;

ColumnVector ridx (nz);

ColumnVector cidx (nz);

ColumnVector data (nz);

ridx(0) = 1; cidx(0) = 1; data(0) = 1;

ridx(1) = 2; cidx(1) = 2; data(1) = 2;

ridx(2) = 2; cidx(2) = 4; data(2) = 3;

ridx(3) = 3; cidx(3) = 4; data(3) = 4;

SparseMatrix sm (data, ridx, cidx, nr, nc);

生成稀疏矩阵存储部分给出的矩阵。请注意，在创建矩阵本身时不使用压缩矩阵格式，而是在内部使用。

正如在稀疏矩阵一章中所讨论的，稀疏矩阵的值以递增的列主顺序存储。虽然用户传递的数据不需要遵守这一要求，但对数据进行预排序将大大加快稀疏矩阵的创建速度。

这种创建稀疏矩阵的技术的缺点是存在两个数据副本的时间很短。对于内存极度受限的问题，这可能不是创建稀疏矩阵的最佳技术。

另一种方法是首先创建一个包含所需数量的非零元素的稀疏矩阵，然后再填充这些元素。示例代码:

int nz, nr, nc;

nz = 4, nr = 3, nc = 4;

SparseMatrix sm (nr, nc, nz);

sm(0,0) = 1; sm(0,1) = 2; sm(1,3) = 3; sm(2,3) = 4;

这将创建与前面相同的矩阵。同样，虽然不是严格必要的，但如果创建稀疏矩阵并按列为主顺序添加元素，速度会快得多。这样做的原因是，当元素插入到当前已知元素列表的末尾时，矩阵中的任何元素都不需要移动以允许插入新元素;只有列索引需要更新。

关于这种创建稀疏矩阵的方法，还有几点需要注意。首先，可以创建一个元素少于实际插入元素的稀疏矩阵。因此,

int nr, nc;

nr = 3, nc = 4;

SparseMatrix sm (nr, nc, 0);

sm(0,0) = 1; sm(0,1) = 2; sm(1,3) = 3; sm(2,3) = 4;

完全有效。然而，这是一个非常糟糕的主意，因为当每个新元素被添加到稀疏矩阵时，矩阵需要请求更多的空间并重新分配内存。这是一个代价高昂的操作，将显著减缓这种创建稀疏矩阵的方式。可以创建具有多余存储空间的稀疏矩阵，因此在本例中nz大于4也是有效的。缺点是矩阵占用的内存比严格需要的要多。

当然，在填充矩阵之前不可能总是知道非零元素的数量。由于这个原因，可以在使用maybe\_compress函数创建稀疏矩阵之后删除它的额外未使用的存储空间。除了释放未使用的存储空间外，maybe\_compress还可以从矩阵中删除零元素。通过将maybe\_compress函数的参数设置为true来控制从矩阵中移除零元素。然而，移除零的代价很高，因为这意味着要对元素重新排序。如果可能的话，用户最好一开始就避免添加不必要的零。maybe\_compress的使用示例如下

int nz, nr, nc;

nz = 6, nr = 3, nc = 4;

SparseMatrix sm1 (nr, nc, nz);

sm1(0,0) = 1; sm1(0,1) = 2; sm1(1,3) = 3; sm1(2,3) = 4;

sm1.maybe\_compress (); // No zero elements were added

SparseMatrix sm2 (nr, nc, nz);

sm2(0,0) = 1; sm2(0,1) = 2; sm(0,2) = 0; sm(1,2) = 0;

sm1(1,3) = 3; sm1(2,3) = 4;

sm2.maybe\_compress (true); // Zero elements were added

应该尽可能避免使用maybe\_compress函数，因为它会减慢矩阵的创建速度。

创建稀疏矩阵的第三种方法是直接使用压缩行格式的数据。这种先进技术的一个例子可能是

octave\_value arg;

…

int nz, nr, nc;

nz = 6, nr = 3, nc = 4; // Assume we know the max # nz

SparseMatrix sm (nr, nc, nz);

Matrix m = arg.matrix\_value ();

int ii = 0;

sm.cidx (0) = 0;

for (int j = 1; j < nc; j++)

{

for (int i = 0; i < nr; i++)

{

double tmp = m(i,j);

if (tmp != 0.)

{

sm.data(ii) = tmp;

sm.ridx(ii) = i;

ii++;

}

}

sm.cidx(j+1) = ii;

}

sm.maybe\_compress (); // If don't know a priori the final # of nz.

这可能是创建稀疏矩阵最有效的方法。

最后，有时可能出现最初创建的存储量不足以完全存储稀疏矩阵的情况。因此，存在方法change\_capacity来重新分配稀疏内存。然后将上面的示例修改为

octave\_value arg;

…

int nz, nr, nc;

nz = 6, nr = 3, nc = 4; // Guess the number of nz elements

SparseMatrix sm (nr, nc, nz);

Matrix m = arg.matrix\_value ();

int ii = 0;

sm.cidx (0) = 0;

for (int j = 1; j < nc; j++)

{

for (int i = 0; i < nr; i++)

{

double tmp = m(i,j);

if (tmp != 0.)

{

if (ii == nz)

{

nz += 2; // Add 2 more elements

sm.change\_capacity (nz);

}

sm.data(ii) = tmp;

sm.ridx(ii) = i;

ii++;

}

}

sm.cidx(j+1) = ii;

}

sm.maybe\_compress (); // If don't know a priori the final # of nz.

请注意，增加和减少稀疏矩阵中非零元素的数量都是昂贵的，因为它涉及内存重新分配。另外，由于矩阵的一部分(虽然不是全部)同时作为新旧副本存在，因此需要额外的内存。因此，如果可能的话，避免改变容量。

**A.1.6.3在oct文件中使用稀疏矩阵**

在Octave解释器中可用的用于稀疏矩阵的大多数相同运算符和函数也可以在oct文件中使用。从octave\_value中提取稀疏矩阵并将其作为octave\_value返回的基本方法可以在下面的示例中看到。

octave\_value\_list retval;

SparseMatrix sm = args(0).sparse\_matrix\_value ();

SparseComplexMatrix scm = args(1).sparse\_complex\_matrix\_value ();

SparseBoolMatrix sbm = args(2).sparse\_bool\_matrix\_value ();

…

retval(2) = sbm;

retval(1) = scm;

retval(0) = sm;

到octave\_value的转换由稀疏的octave\_value构造函数处理，因此不需要特别注意。

**A.1.7访问oct文件中的全局变量**

全局变量允许访问全局作用域中的变量。全局变量可以在oct文件中通过使用支持函数global\_varval和global\_assign从当前解释器的符号表中访问。这两个函数的第一个参数都是一个字符串，表示要获取或分配的变量名。global\_assign的第二个参数是要分配的值。使用这两个函数的一个例子是

#include <octave/oct.h>

#include <octave/interpreter.h>

DEFMETHOD\_DLD (globaldemo, interp, args, , "Global Demo")

{

if (args.length () != 1)

print\_usage ();

octave\_value retval;

std::string s = args(0).string\_value ();

octave::symbol\_table& symtab = interp.get\_symbol\_table ();

octave\_value tmp = symtab.global\_varval (s);

if (tmp.is\_defined ())

retval = tmp;

else

retval = "Global variable not found";

symtab.global\_assign ("a", 42.0);

return retval;

}

使用它的一个例子是

global a b

b = 10;

globaldemo ("b")

⇒ 10

globaldemo ("c")

⇒ "Global variable not found"

num2str (a)

⇒ 42

**A.1.8从Oct-Files调用Octave函数**

通常需要能够从oct文件中调用另一个Octave函数，在Octave本身中有许多这样的例子。例如，quad函数是一个oct文件，它通过对用户提供的函数进行正交计算定积分。

也有许多方法可以将函数作为输入。它可能被认为是

1. 函数处理
2. 匿名函数句柄
3. 字符串

下面的代码演示了将函数传递给oct文件的所有四种方法。

#include <octave/oct.h>

#include <octave/parse.h>

DEFMETHOD\_DLD (funcdemo, interp, args, nargout, "Function Demo")

{

int nargin = args.length ();

if (nargin < 2)

print\_usage ();

octave\_value\_list newargs;

for (octave\_idx\_type i = nargin - 1; i > 0; i--)

newargs(i-1) = args(i);

octave\_value\_list retval;

if (args(0).is\_function\_handle () || args(0).is\_inline\_function ()

|| args(0).is\_string ())

retval = interp.feval (args(0), newargs, nargout);

else

error ("funcdemo: INPUT must be string, inline, or function handle");

return retval;

}

演示代码的第一个输入是用户提供的函数，其余参数都传递给该函数。

funcdemo (@sin, 1)

⇒ 0.84147

funcdemo (@(x) sin (x), 1)

⇒ 0.84147

funcdemo ("sin", 1)

⇒ 0.84147

funcdemo (@atan2, 1, 1)

⇒ 0.78540

当user函数作为字符串传递时，对该函数的处理是不同的。在某些情况下，有必要将用户提供的函数作为octave\_function对象。在这种情况下，string参数可以用来创建一个临时函数，如下所示。

std::octave fcn\_name = unique\_symbol\_name ("\_\_fcn\_\_");

std::string fcode = "function y = ";

fcode.append (fcn\_name);

fcode.append ("(x) y = ");

fcn = extract\_function (args(0), "funcdemo", fcn\_name,

fcode, "; endfunction");

…

if (fcn\_name.length ())

clear\_function (fcn\_name);

在这种情况下，有两件重要的事情需要知道。首先，user函数的输入参数数量是固定的，在上面的例子中是一个参数。其次，为了避免在Octave符号表中留下临时函数，应该在使用后将其清除。此外，按照惯例，所有内部函数名都以字符序列' \_\_ '开始和结束。

**A.1.9从oct文件调用外部代码**

将外部C代码链接到Octave相对简单，因为C函数可以很容易地直接从c++中调用。一个可能的问题是，外部C函数的声明可能需要显式地定义为编译器的C函数。如果外部C函数的声明在头文件foo.h中，那么确保c++编译器将这些声明视为C代码的策略是

#ifdef \_\_cplusplus

extern "C"

{

#endif

#include "foo.h"

#ifdef \_\_cplusplus

} /\* end extern "C" \*/

#endif

当调用用Fortran代码实现的函数时，必须考虑到一些特性。Fortran中的符号名是不区分大小写的，根据所使用的Fortran编译器，函数名导出时要么全小写，要么全大写。此外，一些编译器在导出的函数名后面不附加一个、一个或两个下划线“\_”。这被称为“名字混淆”。

Octave提供宏，允许编写代码自动处理许多不同Fortran编译器的名称混淆。这两个宏是F77\_FUNC和F77\_FUNC\_。前者应该用于名称中不包含下划线的Fortran函数。后者应该用于名称中带有下划线的Fortran函数。这两个宏都有两个参数:第一个是用小写字符表示的Fortran函数名。第二个是相同的Fortran函数名，但都是大写字符。

除了名称混淆之外，不同的编译器对某些类型使用不同的调用约定。Octave定义了以下预处理器宏，以允许编写可用于不同Fortran调用约定的代码。

注意，我们不尝试处理Fortran函数，我们总是为它们使用子例程包装器，并将返回值作为额外的参数传递。

使用以下宏将字符串从C传递到Fortran:

F77\_CHAR\_ARG(x)

F77\_CONST\_CHAR\_ARG(x)

F77\_CXX\_STRING\_ARG(x)

F77\_CHAR\_ARG\_LEN(l)

F77\_CHAR\_ARG\_DECL

F77\_CONST\_CHAR\_ARG\_DECL

F77\_CHAR\_ARG\_LEN\_DECL

使用以下宏来编写接受fortran风格字符串的c语言函数:

F77\_CHAR\_ARG\_DEF(s,len)

F77\_CONST\_CHAR\_ARG\_DEF(s,len)

F77\_CHAR\_ARG\_LEN\_DEF(len)

F77\_CHAR\_ARG\_USE(s)

F77\_CHAR\_ARG\_LEN\_USE(s,len)

在c++代码中使用以下宏来处理Fortran类型:

F77\_INT4

相当于Fortran的INTEGER\*4类型

F77\_DBLE

相当于Fortran的DOUBLE PRECISION类型

F77\_REAL

相当于Fortran的REAL类型

F77\_CMPLX

相当于Fortran的COMPLEX类型

F77\_DBLE\_CMPLX

相当于Fortran的DOUBLE COMPLEX类型

F77\_LOGICAL

相当于Fortran的LOGICAL类型

F77\_RET\_T

类似于Fortran子程序的c++函数的返回类型。

使用以下宏从c语言函数返回，这些函数的作用类似于Fortran子例程。F77\_NORETURN被用作带有“noreturn”属性标记的函数的最后一条语句。

F77\_RETURN(retval)

F77\_NORETURN(retval)

底层Fortran代码应该使用XSTOPX函数来替换Fortran STOP函数。XSTOPX使用Octave异常处理程序显式地处理Fortran代码中的失败情况。注意，Octave提供了自己的替代BLAS XERBLA函数，该函数使用XSTOPX。

下面的示例显示了在oct文件中包含Fortran函数，其中包含c++包装器

#include <octave/oct.h>

#include <octave/f77-fcn.h>

extern "C"

{

F77\_RET\_T

F77\_FUNC (fortransub, FORTRANSUB)

(const F77\_INT&, F77\_DBLE\*, F77\_CHAR\_ARG\_DECL F77\_CHAR\_ARG\_LEN\_DECL);

}

DEFUN\_DLD (fortrandemo, args, , "Fortran Demo")

{

if (args.length () != 1)

print\_usage ();

NDArray a = args(0).array\_value ();

double \*av = a.fortran\_vec ();

octave\_idx\_type na = a.numel ();

OCTAVE\_LOCAL\_BUFFER (char, ctmp, 128);

F77\_FUNC (fortransub, FORTRANSUB)

(na, av, ctmp F77\_CHAR\_ARG\_LEN (128));

return ovl (a, std::string (ctmp));

}

Fortran函数是

subroutinefortransub(n,a,s)

implicitnone

character\*(\*)s

real\*8a(\*)

integer\*4i,n,ioerr

doi=1,n

if(a(i).eq.0d0)then

callxstopx('fortransub:dividebyzero')

else

a(i)=1d0/a(i)

endif

enddo

write(unit=s,fmt='(a,i3,a,a)',iostat=ioerr)

$'Thereare',n,

$'valuesintheinputvector',char(0)

if(ioerr.ne.0)then

callxstopx('fortransub:errorwritingstring')

endif

return

end

这个例子演示了链接到外部Fortran函数所需的大部分特性，包括传递数组和字符串，以及异常处理。为了使示例工作，Fortran和c++文件都需要编译。

mkoctfile fortrandemo.cc fortransub.f

[b, s] = fortrandemo (1:3)

⇒

b = 1.00000 0.50000 0.33333

s = There are 3 values in the input vector

[b, s] = fortrandemo (0:3)

error: fortrandemo: fortransub: divide by zero

**A.1.10在oct文件中分配本地内存**

在oct文件中分配内存似乎很容易，因为可以使用c++的new/delete操作符。但是，在这种情况下，必须非常小心地避免内存泄漏。分配供本地使用的内存的首选方式是使用OCTAVE\_LOCAL\_BUFFER宏。使用它的一个例子是

OCTAVE\_LOCAL\_BUFFER (double, tmp, len)

返回一个长度为len的double \*类型指针TMP。

在这种情况下，Octave本身将担心引用计数和变量作用域，并在没有程序员干预的情况下正确释放内存。

**A.1.11 oct文件输入参数检查**

由于oct文件是编译函数，它们可能会由于粗心的函数调用或内存错误而导致Octave崩溃。非常重要的是，每个函数都有足够的参数检查水平，以确保Octave表现良好。

如前所述，最低要求是在使用输入参数之前检查它们的数量，以避免引用不存在的参数。然而，在某些情况下，这可能是不够的，因为底层代码施加了进一步的约束。例如，如果输入参数不是整数，或者其中一个参数为零，或者输入是复杂的并且期望是实数，则外部函数调用可能是未定义的。因此，oct文件通常需要额外的输入参数检查。

在Octave中有几个函数可以用于参数检查。这些方法包括octave\_value类的方法，如is\_real\_matrix、is\_numeric\_type等(参见v.h)。通常，通过了解Octave m文件语言，您可以猜测相应的c++例程将会是什么。此外，还有一些更专门的输入验证函数，下面将演示其中的一些函数。

#include <octave/oct.h>

DEFUN\_DLD (paramdemo, args, nargout, "Parameter Check Demo")

{

if (args.length () != 1)

print\_usage ();

NDArray m = args(0).array\_value ();

double min\_val = -10.0;

double max\_val = 10.0;

octave\_stdout << "Properties of input array:\n";

if (m.any\_element\_is\_negative ())

octave\_stdout << " includes negative values\n";

if (m.any\_element\_is\_inf\_or\_nan ())

octave\_stdout << " includes Inf or NaN values\n";

if (m.any\_element\_not\_one\_or\_zero ())

octave\_stdout << " includes other values than 1 and 0\n";

if (m.all\_elements\_are\_int\_or\_inf\_or\_nan ())

octave\_stdout << " includes only int, Inf or NaN values\n";

if (m.all\_integers (min\_val, max\_val))

octave\_stdout << " includes only integers in [-10,10]\n";

return octave\_value\_list ();

}

其用法的一个例子是:

paramdemo ([1, 2, NaN, Inf])

⇒ Properties of input array:

includes Inf or NaN values

includes other values than 1 and 0

includes only int, Inf or NaN values

**A.1.12 oct文件异常与错误处理**

Octave的另一个重要特性是它能够在扩展计算期间对用户键入Control-C作出反应。这种能力是基于c++异常处理程序的，当处理异常时，由c++ new/delete方法分配的内存会自动释放。当编写可能长时间运行的oct文件时，程序员必须定期使用宏OCTAVE\_QUIT，以便允许Octave检查并可能响应用户键入Control-C。例如:

for (octave\_idx\_type i = 0; i < a.nelem (); i++)

{

OCTAVE\_QUIT;

b.elem (i) = 2. \* a.elem (i);

}

内部循环中存在的OCTAVE\_QUIT宏允许Octave检测并确认Control-C键序列。如果没有这个宏，用户必须等待oct-file函数在中断被处理之前返回，或者用户必须按Control-C三次，这将迫使Octave完全退出。

OCTAVE\_QUIT宏确实会造成很小的性能损失;对于已知较小的循环，包含OCTAVE\_QUIT可能没有意义。

在创建使用外部库的oct文件时，该函数可能会在外部库中花费大量时间。在这种情况下，通常不可能使用OCTAVE\_QUIT宏。在这种情况下的替代代码是

BEGIN\_INTERRUPT\_IMMEDIATELY\_IN\_FOREIGN\_CODE;

… some code that calls a "foreign" function …

END\_INTERRUPT\_IMMEDIATELY\_IN\_FOREIGN\_CODE;

这样做的缺点是，如果外部代码在内部分配任何内存，那么这些内存可能会在中断期间丢失，而不会被重新分配。因此，理想情况下，Octave本身应该使用fortran\_vec方法或OCTAVE\_LOCAL\_BUFFER宏分配外部代码所需的任何内存。

Octave unwind\_protect机制(unwind\_protect语句)也可以用于oct文件。与Octave的异常处理相结合，它确保即使发生异常也始终运行某些恢复代码。这是一个使用这个机器的例子

#include <octave/oct.h>

#include <octave/unwind-prot.h>

void

my\_err\_handler (const char \*fmt, ...)

{

// Do nothing!!

}

void

my\_err\_with\_id\_handler (const char \*id, const char \*fmt, ...)

{

// Do nothing!!

}

DEFUN\_DLD (unwinddemo, args, nargout, "Unwind Demo")

{

if (args.length () < 2)

print\_usage ();

NDArray a = args(0).array\_value ();

NDArray b = args(1).array\_value ();

// Create unwind\_action objects. At the end of the enclosing scope,

// destructors for these objects will call the given functions with

// the specified arguments.

octave::unwind\_action restore\_warning\_handler

(set\_liboctave\_warning\_handler, current\_liboctave\_warning\_handler);

octave::unwind\_action restore\_warning\_with\_id\_handler

(set\_liboctave\_warning\_with\_id\_handler,

current\_liboctave\_warning\_with\_id\_handler);

set\_liboctave\_warning\_handler (my\_err\_handler);

set\_liboctave\_warning\_with\_id\_handler (my\_err\_with\_id\_handler);

return octave\_value (quotient (a, b));

}

从这个例子中可以看出:

unwinddemo (1, 0)

⇒ Inf

1 / 0

⇒ warning: division by zero

Inf

除零的警告(以及实际上所有的警告)在unwinddemo函数中被禁用。

**A.1.13 oct文件的记录和测试**

oct文件的文档包含在DEFUN\_DLD宏的第四个字符串参数中。这个字符串可以用与用户函数的帮助字符串相同的方式进行格式化，但是在oct文件中格式化帮助字符串有一些特殊的问题。

主要问题是帮助字符串通常比单行文本长，因此需要考虑长多行帮助字符串的格式。有几种可能的解决方案，但最常见的解决方案在

DEFUN\_DLD (do\_what\_i\_want, args, nargout,

"-\*- texinfo -\*-\n\

@deftypefn {} {} do\_what\_i\_say (@var{n})\n\

A function that does what the user actually wants rather\n\

than what they requested.\n\

@end deftypefn")

{

…

}

其中每行文本以\n\结束，\n\是一个嵌入在字符串中的换行符和一个c++字符串延续字符。注意，最后的\必须是该行的最后一个字符。

Octave还包括在代码本身内嵌入函数的测试和演示代码的能力(参见测试和演示函数)。这可以在oct文件(实际上是任何文件)中使用，但有某些附带条件。首先，测试和演示函数Octave查找%!作为一行的前两个字符，用于标识测试代码和演示代码。这也是oct文件的要求。此外，测试和演示代码必须包装在注释块中，以避免编译器对其进行解释。最后，Octave测试和演示代码必须能够访问oct文件的原始源代码，而不仅仅是编译后的代码，因为测试是从编译后的代码中剥离出来的。oct文件中的一个示例可能是

/\*

%!assert (sin ([1,2]), [sin(1),sin(2)])

%!error (sin ())

%!error (sin (1,1))

\*/

**A.2 Mex-Files**

Octave包含一个接口，允许编译和使用遗留的mex文件。该接口还可以用于在Octave和MATLAB用户之间共享编译代码。但是，由于mex-files暴露了MATLAB的内部API，并且Octave的内部结构是不同的，因此在Octave中，mex-file永远不可能具有与等效oct-file相同的性能。特别是，为了支持将变量传递给mex函数的方式，在调用或从mex文件函数返回时需要大量额外的内存块副本。出于这个原因，建议使用前面讨论的oct-file接口编写任何新代码。

**A.2.1入门x- files**

构建mex文件的基本命令是mkoctfile——mex或mex。第一个命令可以在Octave中使用，也可以在命令行中使用。为了避免MATLAB自己的mex命令的问题，命令mex的使用仅限于在Octave内。编译后的mex文件扩展名为.mex。

: mex [-options] file …

: status = mex (…)

将用C、c++或Fortran编写的源代码编译为一个MEX文件。

Status是mkoctfile函数的返回状态。

如果编译失败，并且未请求输出参数，则会引发错误。但是，如果程序员请求状态，Octave只会发出警告，程序员有责任验证命令是否成功。

这相当于mkoctfile——mex [-options] file。

参见:mkoctfile, next。

: ext = mexext ()

返回用于MEX文件的文件扩展名。

编程注意:Octave对所有mex文件使用扩展名mex，而不管操作系统(Linux、Windows、Apple)或硬件的位宽(32位或64位)。

参见:mex。

考虑下面这个简短的例子:

#include "mex.h"

void

mexFunction (int nlhs, mxArray \*plhs[],

int nrhs, const mxArray \*prhs[])

{

mexPrintf ("Hello, World!\n");

mexPrintf ("I have %d inputs and %d outputs\n", nrhs, nlhs);

/\* Return empty matrices for any outputs \*/

int i;

for (i = 0; i < nlhs; i++)

plhs[i] = mxCreateDoubleMatrix (0, 0, mxREAL);

}

第一行#include“ext .h”提供了一个ext文件所需的所有定义。Octave和MATLAB之间的一个重要区别是，头文件“matrix.h”是通过包含“ext .h”来隐式包含的。对于不区分大写和小写文件名的操作系统和编译器，这是避免与Octave文件“Matrix.h”冲突所必需的。

进入ext文件的入口点由mexFunction定义。该函数接受四个参数:

1. 返回参数的个数(左侧参数的个数)。
2. 返回参数的指针数组。
3. 输入参数的数量(右侧参数的数量)。
4. 指向输入参数的指针数组。

注意，函数名定义没有显式地包含在mexFunction中，因此每个文件只能有一个mexFunction入口点。相反，在Octave中看到的函数名由ext文件本身的名称减去扩展名决定。如果上述函数位于myhello.c文件中，则可以使用

mkoctfile --mex myhello.c

这将创建一个文件myhello.mex。然后可以从Octave中运行该函数

myhello (1,2,3)

⇒ Hello, World!

⇒ I have 3 inputs and 0 outputs

需要注意的是，ext文件不包含帮助字符串。要记录mex-files，在与mex-file本身相同的目录中应该存在一个m-file。以上面的示例为例，需要一个文件myhello。M，其中可能包含文本

%MYHELLO Simple test of the functionality of a mex-file.

在这种情况下，将在Octave中执行的函数将由mex-file给出，而帮助字符串将来自m-file。这对于允许在Octave语言本身中使用ext文件的样例实现以进行测试也很有用。

尽管在单个mex-file中不能有多个入口点，但是可以使用mexFunctionName函数来确定调用mex-file的名称。这可以用于根据函数名更改ext文件的行为。例如，如果

#include "mex.h"

void

mexFunction (int nlhs, mxArray \*plhs[],

int nrhs, const mxArray \*prhs[])

{

const char \*nm;

nm = mexFunctionName ();

mexPrintf ("You called function: %s\n", nm);

if (strcmp (nm, "myfunc") == 0)

mexPrintf ("This is the principal function\n", nm);

return;

}

在文件myfunction .c中，并使用

mkoctfile --mex myfunc.c

ln -s myfunc.mex myfunc2.mex

然后就可以看出了

myfunc ()

⇒ You called function: myfunc

This is the principal function

myfunc2 ()

⇒ You called function: myfunc2

可以根据函数的名称更改mex-file的行为。

虽然用户应该只在他们的代码中包含ext .h，但Octave声明了额外的函数、类型等，供用户在头文件mexprotoo .h和mxarray.h中编写ext文件。

**A.2.2在x文件中使用矩阵和数组**

所有变量的基本mex类型是mxArray。任何对象，如矩阵、单元数组或结构，都存储在这个基本类型中。mxArray的作用基本上与oct-files中的octave\_value类相同，因为它充当所有更专门化类型的容器。

mxArray结构至少包含它所表示的变量的名称、维度、类型以及变量是实变量还是复杂变量。它还可以包含许多附加字段，具体取决于mxArray的类型。有许多函数可以创建mxArray结构，包括mxCreateDoubleMatrix、mxCreateCellArray、mxCreateSparse和通用的mxCreateNumericArray。

访问数组中数据的基本函数是mxGetPr。因为mex接口假设一个复杂数组的实部和虚部是分开存储的，所以有一个等价的函数mxGetPi来获取虚部。这两个函数只适用于双精度矩阵。通用函数mxGetData和mxGetImagData对所有矩阵类型执行相同的操作。例如:

mxArray \*m;

mwSize \*dims;

UINT32\_T \*pr;

dims = (mwSize \*) mxMalloc (2 \* sizeof (mwSize));

dims[0] = 2; dims[1] = 2;

m = mxCreateNumericArray (2, dims, mxUINT32\_CLASS, mxREAL);

pr = (UINT32\_T \*) mxGetData (m);

还有函数mxSetPr等，它们执行逆操作，并设置数组的数据使用mxSetPr参数所指向的内存块。

请注意，上面使用的mwSize类型以及mwIndex都被定义为构建mex文件的平台上Octave中索引的本机精度。这使得32位和64位平台都可以支持mex-files。mwSize用于定义数组的尺寸和元素的最大数量，而mwIndex用于定义数组的索引。

下面的mypow2.c文件给出了一个示例，演示了如何处理任意实数或复杂的双精度数组。

#include "mex.h"

void

mexFunction (int nlhs, mxArray \*plhs[],

int nrhs, const mxArray \*prhs[])

{

mwSize n;

mwIndex i;

double \*vri, \*vro;

if (nrhs != 1 || ! mxIsDouble (prhs[0]))

mexErrMsgTxt ("ARG1 must be a double matrix");

n = mxGetNumberOfElements (prhs[0]);

plhs[0] = mxCreateNumericArray (mxGetNumberOfDimensions (prhs[0]),

mxGetDimensions (prhs[0]),

mxGetClassID (prhs[0]),

mxIsComplex (prhs[0]));

vri = mxGetPr (prhs[0]);

vro = mxGetPr (plhs[0]);

if (mxIsComplex (prhs[0]))

{

double \*vii, \*vio;

vii = mxGetPi (prhs[0]);

vio = mxGetPi (plhs[0]);

for (i = 0; i < n; i++)

{

vro[i] = vri[i] \* vri[i] - vii[i] \* vii[i];

vio[i] = 2 \* vri[i] \* vii[i];

}

}

else

{

for (i = 0; i < n; i++)

vro[i] = vri[i] \* vri[i];

}

}

使用它的一个例子是

b = randn (4,1) + 1i \* randn (4,1);

all (b.^2 == mypow2 (b))

⇒ 1

上面的例子使用了函数mxGetDimensions、mxGetNumberOfElements和mxGetNumberOfDimensions来处理多维数组的维度。函数mxGetM和mxGetN也可用于查找二维矩阵(MxN矩阵)中的行数和列数。

**A.2.3 x文件中的字符串**

由于mex-files不像Octave那样区分单引号和双引号字符串，因此使用字符串和字符矩阵可能不那么复杂。一个与stringdemo中的demo类似的使用示例。Cc在mystring.c文件中给出，如下所示。

#include <string.h>

#include "mex.h"

void

mexFunction (int nlhs, mxArray \*plhs[],

int nrhs, const mxArray \*prhs[])

{

mwSize m, n;

mwIndex i, j;

mxChar \*pi, \*po;

if (nrhs != 1 || ! mxIsChar (prhs[0])

|| mxGetNumberOfDimensions (prhs[0]) > 2)

mexErrMsgTxt ("ARG1 must be a char matrix");

m = mxGetM (prhs[0]);

n = mxGetN (prhs[0]);

pi = mxGetChars (prhs[0]);

plhs[0] = mxCreateNumericMatrix (m, n, mxCHAR\_CLASS, mxREAL);

po = mxGetChars (plhs[0]);

for (j = 0; j < n; j++)

for (i = 0; i < m; i++)

po[j\*m + m - 1 - i] = pi[j\*m + i];

}

其预期输出的一个示例是

mystring (["First String"; "Second String"])

⇒ Second String

First String

mex接口中用于处理字符串的其他函数有mxCreateString、mxArrayToString和mxCreateCharMatrixFromStrings。在ext文件中，字符串被认为是矢量而不是矩阵。这可能是一种任意的区分，因为矩阵的mxArray中的数据在任何情况下都是连续的。

**A.2.4带有x- files的Cell Arrays**

可以在ext -files中对Cell数组执行与oct-files中完全相同的操作。一个复制celldemo功能的示例。在ext -file中，由mycell.c给出Cc oct-file，如下所示。

#include "mex.h"

void

mexFunction (int nlhs, mxArray \*plhs[],

int nrhs, const mxArray \*prhs[])

{

mwSize n;

mwIndex i;

if (nrhs != 1 || ! mxIsCell (prhs[0]))

mexErrMsgTxt ("ARG1 must be a cell");

n = mxGetNumberOfElements (prhs[0]);

n = (n > nlhs ? nlhs : n);

for (i = 0; i < n; i++)

plhs[i] = mxDuplicateArray (mxGetCell (prhs[0], i));

}

输出也与oct-file版本相同。

[b1, b2, b3] = mycell ({1, [1, 2], "test"})

⇒

b1 = 1

b2 =

1 2

b3 = test

注意在示例中使用了mxDuplicateArray函数。这是必需的，因为mxGetCell返回的mxArray指针可能会被释放。用于设置Cell值的mxGetCell的逆函数是mxSetCell，定义为

void mxSetCell (mxArray \*ptr, int idx, mxArray \*val);

最后，要创建单元格数组或矩阵，适当的函数是

mxArray \*mxCreateCellArray (int ndims, const int \*dims);

mxArray \*mxCreateCellMatrix (int m, int n);

**A.2.5带有x文件的结构**

在ext文件中创建结构的基本函数是mxCreateStructMatrix，它创建一个带有二维矩阵的结构数组，或mxCreateStructArray。

mxArray \*mxCreateStructArray (int ndims, int \*dims,

int num\_keys,

const char \*\*keys);

mxArray \*mxCreateStructMatrix (int rows, int cols,

int num\_keys,

const char \*\*keys);

然后可以使用mxGetField和mxSetField或mxGetFieldByNumber和mxSetFieldByNumber函数来访问结构的字段。

mxArray \*mxGetField (const mxArray \*ptr, mwIndex index,

const char \*key);

mxArray \*mxGetFieldByNumber (const mxArray \*ptr,

mwIndex index, int key\_num);

void mxSetField (mxArray \*ptr, mwIndex index,

const char \*key, mxArray \*val);

void mxSetFieldByNumber (mxArray \*ptr, mwIndex index,

int key\_num, mxArray \*val);

oct-file结构接口和mex-file版本之间的区别在于，在mex-files中对结构进行操作的函数直接包括对每个字段的元素数组的元素的索引;而oct-file结构的每个字段包含一个Cell Array。

可以在下面的mystruct.c文件中找到一个演示在ext文件中使用结构的示例。

#include "mex.h"

void

mexFunction (int nlhs, mxArray \*plhs[],

int nrhs, const mxArray \*prhs[])

{

int i;

mwIndex j;

mxArray \*v;

const char \*keys[] = { "this", "that" };

if (nrhs != 1 || ! mxIsStruct (prhs[0]))

mexErrMsgTxt ("ARG1 must be a struct");

for (i = 0; i < mxGetNumberOfFields (prhs[0]); i++)

for (j = 0; j < mxGetNumberOfElements (prhs[0]); j++)

{

mexPrintf ("field %s(%d) = ", mxGetFieldNameByNumber (prhs[0], i), j);

v = mxGetFieldByNumber (prhs[0], j, i);

mexCallMATLAB (0, NULL, 1, &v, "disp");

}

v = mxCreateStructMatrix (2, 2, 2, keys);

mxSetFieldByNumber (v, 0, 0, mxCreateString ("this1"));

mxSetFieldByNumber (v, 0, 1, mxCreateString ("that1"));

mxSetFieldByNumber (v, 1, 0, mxCreateString ("this2"));

mxSetFieldByNumber (v, 1, 1, mxCreateString ("that2"));

mxSetFieldByNumber (v, 2, 0, mxCreateString ("this3"));

mxSetFieldByNumber (v, 2, 1, mxCreateString ("that3"));

mxSetFieldByNumber (v, 3, 0, mxCreateString ("this4"));

mxSetFieldByNumber (v, 3, 1, mxCreateString ("that4"));

if (nlhs)

plhs[0] = v;

}

这个函数在Octave中的行为示例如下

a(1).f1 = "f11"; a(1).f2 = "f12";

a(2).f1 = "f21"; a(2).f2 = "f22";

b = mystruct (a);

⇒ field f1(0) = f11

field f1(1) = f21

field f2(0) = f12

field f2(1) = f22

b

⇒ 2x2 struct array containing the fields:

this

that

b(3)

⇒ scalar structure containing the fields:

this = this3

that = that3

**A.2.6带有x文件的稀疏矩阵**

用于稀疏矩阵的Octave格式与mex格式相同，因为它是压缩的列稀疏格式。此外，在这两种实现中，稀疏矩阵都需要是二维的。对程序员来说，唯一重要的区别是矩阵的实部和虚部是分开存储的。

ext -file接口除了使用mxGetM、mxGetN、mxSetM、mxSetN、mxGetPr、mxGetPi、mxSetPr和mxSetPi之外，还提供了以下函数。

mwIndex \*mxGetIr (const mxArray \*ptr);

mwIndex \*mxGetJc (const mxArray \*ptr);

mwSize mxGetNzmax (const mxArray \*ptr);

void mxSetIr (mxArray \*ptr, mwIndex \*ir);

void mxSetJc (mxArray \*ptr, mwIndex \*jc);

void mxSetNzmax (mxArray \*ptr, mwSize nzmax);

mxGetNzmax获取可存储在稀疏矩阵中的最大元素数。这并不一定是稀疏矩阵中非零元素的个数。mxGetJc返回一个数组，其值比稀疏矩阵中的列数多一个。mxGetJc返回的数组连续值之间的差定义了稀疏矩阵每列中非零元素的数量。因此,

mwSize nz, n;

mwIndex \*Jc;

mxArray \*m;

…

n = mxGetN (m);

Jc = mxGetJc (m);

nz = Jc[n];

返回nz中存储在矩阵中的非零元素的实际数目。由于mxGetPr和mxGetPi返回的数组只包含矩阵的非零值，我们还需要一个指向非零元素行的指针，这是由mxGetIr给出的。下面的mysparse.c文件给出了在ext -files中使用稀疏矩阵的完整示例。

#include "mex.h"

void

mexFunction (int nlhs, mxArray \*plhs[],

int nrhs, const mxArray \*prhs[])

{

mwSize m, n, nz;

mxArray \*v;

mwIndex i;

double \*pr, \*pi;

double \*pr2, \*pi2;

mwIndex \*ir, \*jc;

mwIndex \*ir2, \*jc2;

if (nrhs != 1 || ! mxIsSparse (prhs[0]))

mexErrMsgTxt ("ARG1 must be a sparse matrix");

m = mxGetM (prhs[0]);

n = mxGetN (prhs[0]);

nz = mxGetNzmax (prhs[0]);

if (mxIsComplex (prhs[0]))

{

mexPrintf ("Matrix is %d-by-%d complex sparse matrix", m, n);

mexPrintf (" with %d elements\n", nz);

pr = mxGetPr (prhs[0]);

pi = mxGetPi (prhs[0]);

ir = mxGetIr (prhs[0]);

jc = mxGetJc (prhs[0]);

i = n;

while (jc[i] == jc[i-1] && i != 0) i--;

mexPrintf ("last nonzero element (%d, %d) = (%g, %g)\n",

ir[nz-1]+ 1, i, pr[nz-1], pi[nz-1]);

v = mxCreateSparse (m, n, nz, mxCOMPLEX);

pr2 = mxGetPr (v);

pi2 = mxGetPi (v);

ir2 = mxGetIr (v);

jc2 = mxGetJc (v);

for (i = 0; i < nz; i++)

{

pr2[i] = 2 \* pr[i];

pi2[i] = 2 \* pi[i];

ir2[i] = ir[i];

}

for (i = 0; i < n + 1; i++)

jc2[i] = jc[i];

if (nlhs > 0)

plhs[0] = v;

}

else if (mxIsLogical (prhs[0]))

{

mxLogical \*pbr, \*pbr2;

mexPrintf ("Matrix is %d-by-%d logical sparse matrix", m, n);

mexPrintf (" with %d elements\n", nz);

pbr = mxGetLogicals (prhs[0]);

ir = mxGetIr (prhs[0]);

jc = mxGetJc (prhs[0]);

i = n;

while (jc[i] == jc[i-1] && i != 0) i--;

mexPrintf ("last nonzero element (%d, %d) = %d\n",

ir[nz-1]+ 1, i, pbr[nz-1]);

v = mxCreateSparseLogicalMatrix (m, n, nz);

pbr2 = mxGetLogicals (v);

ir2 = mxGetIr (v);

jc2 = mxGetJc (v);

for (i = 0; i < nz; i++)

{

pbr2[i] = pbr[i];

ir2[i] = ir[i];

}

for (i = 0; i < n + 1; i++)

jc2[i] = jc[i];

if (nlhs > 0)

plhs[0] = v;

}

else

{

mexPrintf ("Matrix is %d-by-%d real sparse matrix", m, n);

mexPrintf (" with %d elements\n", nz);

pr = mxGetPr (prhs[0]);

ir = mxGetIr (prhs[0]);

jc = mxGetJc (prhs[0]);

i = n;

while (jc[i] == jc[i-1] && i != 0) i--;

mexPrintf ("last nonzero element (%d, %d) = %g\n",

ir[nz-1]+ 1, i, pr[nz-1]);

v = mxCreateSparse (m, n, nz, mxREAL);

pr2 = mxGetPr (v);

ir2 = mxGetIr (v);

jc2 = mxGetJc (v);

for (i = 0; i < nz; i++)

{

pr2[i] = 2 \* pr[i];

ir2[i] = ir[i];

}

for (i = 0; i < n + 1; i++)

jc2[i] = jc[i];

if (nlhs > 0)

plhs[0] = v;

}

}

mysparse的一个示例用法是

sm = sparse ([1, 0; 0, pi]);

mysparse (sm)

⇒

Matrix is 2-by-2 real sparse matrix with 2 elements

last nonzero element (2, 2) = 3.14159

**A.2.7调用x文件中的其他函数**

可以使用mexCallMATLAB从ext文件中调用其他Octave函数。在下面的示例中可以看到使用mexCallMATLAB的示例。

#include "mex.h"

void

mexFunction (int nlhs, mxArray \*plhs[],

int nrhs, const mxArray \*prhs[])

{

char \*str;

mexPrintf ("Starting file myfeval.mex\n");

mexPrintf ("I have %d inputs and %d outputs\n", nrhs, nlhs);

if (nrhs < 1 || ! mxIsChar (prhs[0]))

mexErrMsgTxt ("ARG1 must be a function name");

str = mxArrayToString (prhs[0]);

mexPrintf ("I'm going to call the function %s\n", str);

if (nlhs == 0)

nlhs = 1; // Octave's automatic 'ans' variable

/\* Cast prhs just to get rid of 'const' qualifier and stop compile warning \*/

mexCallMATLAB (nlhs, plhs, nrhs-1, (mxArray\*\*)prhs+1, str);

mxFree (str);

}

如果这段代码在文件myfeval.c中，并被编译为myfeval.c。Mex，那么使用它的一个例子是

a = myfeval ("sin", 1)

⇒ Starting file myfeval.mex

I have 2 inputs and 1 outputs

I'm going to call the interpreter function sin

a = 0.84147

注意，不能在ext文件中使用函数句柄。

**A.3独立程序**

Octave本身使用的库可以在独立应用程序中使用。然后，这些应用程序可以访问数组和矩阵类，以及所有的Octave算法。下面的c++程序，使用了libctave中的类Matrix。A或libotave，所以。

#include <iostream>

#include <octave/oct.h>

int

main (void)

{

std::cout << "Hello Octave world!\n";

int n = 2;

Matrix a\_matrix = Matrix (n, n);

for (octave\_idx\_type i = 0; i < n; i++)

for (octave\_idx\_type j = 0; j < n; j++)

a\_matrix(i,j) = (i + 1) \* 10 + (j + 1);

std::cout << a\_matrix;

return 0;

}

Mkoctfile可用于使用如下命令构建独立应用程序

$ mkoctfile --link-stand-alone standalone.cc -o standalone

$ ./standalone

Hello Octave world!

11 12

21 22

$

注意，独立应用程序将动态链接到Octave库和任何Octave支持库。上面的代码允许应用程序使用Octave数学库。但是，它不允许应用程序使用Octave的脚本文件、oct文件或内置函数。要做到这一点，需要首先初始化Octave解释器。然后，代码中可以看到如何这样做的示例

#include <iostream>

#include <octave/oct.h>

#include <octave/octave.h>

#include <octave/parse.h>

#include <octave/interpreter.h>

int

main (void)

{

// Create interpreter.

octave::interpreter interpreter;

try

{

// Inhibit reading history file by calling

//

// interpreter.initialize\_history (false);

// Set custom load path here if you wish by calling

//

// interpreter.initialize\_load\_path (false);

// Perform final initialization of interpreter, including

// executing commands from startup files by calling

//

// interpreter.initialize ();

//

// if (! interpreter.initialized ())

// {

// std::cerr << "Octave interpreter initialization failed!"

// << std::endl;

// exit (status);

// }

//

// You may skip this step if you don't need to do anything

// between reading the startup files and telling the interpreter

// that you are ready to execute commands.

// Tell the interpreter that we're ready to execute commands:

int status = interpreter.execute ();

if (status != 0)

{

std::cerr << "creating embedded Octave interpreter failed!"

<< std::endl;

return status;

}

octave\_idx\_type n = 2;

octave\_value\_list in;

for (octave\_idx\_type i = 0; i < n; i++)

in(i) = octave\_value (5 \* (i + 2));

octave\_value\_list out = octave::feval ("gcd", in, 1);

if (out.length () > 0)

std::cout << "GCD of ["

<< in(0).int\_value ()

<< ", "

<< in(1).int\_value ()

<< "] is " << out(0).int\_value ()

<< std::endl;

else

std::cout << "invalid\n";

}

catch (const octave::exit\_exception& ex)

{

std::cerr << "Octave interpreter exited with status = "

<< ex.exit\_status () << std::endl;

}

catch (const octave::execution\_exception&)

{

std::cerr << "error encountered in Octave evaluator!" << std::endl;

}

return 0;

}

与之前一样，哪个作为独立应用程序编译并运行

$ mkoctfile --link-stand-alone embedded.cc -o embedded

$ ./embedded

GCD of [10, 15] is 5

$

值得重申的是，如果只从c++独立程序调用内置函数，则不需要初始化解释器。一般规则是，对于解释器中名为function\_name的内置函数，将有一个名为Ffunction\_name的c++函数(注意前面的大写F)可以在c++ API中访问。所有内置函数的声明都收集在头文件builtin-defun-decls.h中。应该谨慎使用此功能，因为内置函数列表可能会更改。不能保证当前是内置函数的函数将来不会被实现为.m文件或动态链接函数。在代码中可以看到如何从c++中调用内置函数的示例

#include <iostream>

#include <octave/oct.h>

#include <octave/builtin-defun-decls.h>

int

main (void)

{

int n = 2;

Matrix a\_matrix = Matrix (n, n);

for (octave\_idx\_type i = 0; i < n; i++)

for (octave\_idx\_type j = 0; j < n; j++)

a\_matrix(i,j) = (i + 1) \* 10 + (j + 1);

std::cout << "This is a matrix:" << std::endl

<< a\_matrix << std::endl;

octave\_value\_list in;

in(0) = a\_matrix;

octave\_value\_list out = Fnorm (in, 1);

double norm\_of\_the\_matrix = out(0).double\_value ();

std::cout << "This is the norm of the matrix:" << std::endl

<< norm\_of\_the\_matrix << std::endl;

return 0;

}

哪个是作为独立应用程序编译和运行的

$ mkoctfile --link-stand-alone standalonebuiltin.cc -o standalonebuiltin

$ ./standalonebuiltin

This is a matrix:

11 12

21 22

This is the norm of the matrix:

34.4952

$

**A.4 Java接口**

Java接口是为从Octave内部调用Java函数而设计的。如果您想做相反的事情，并从Java内部调用Octave，请尝试使用joPas (http://jopas.sourceforge.net)这样的库。

**A.4.1使Java类可用**

Java通过搜索类路径来查找类，类路径是Java归档文件和/或包含类文件的目录的列表。在Octave中，类路径由两部分组成:

静态类路径在JVM启动时初始化一次，动态类路径可以在运行时修改。

Octave首先搜索静态类路径，然后搜索动态类路径。因此，在静态类路径和动态类路径中出现的类将在静态类路径中找到，并从这个位置加载。应该将经常使用或必须对所有用户可用的类添加到静态类路径中。静态类路径是在Java虚拟机启动时从名为javaclasspath.txt(历史上是classpath.txt)的纯文本文件的内容填充一次的。该文件包含一行，用于将每个单独的类路径添加到静态类路径。这些行可以标识包含类文件的目录，或者具有完整类文件层次结构的Java归档文件。以“#”或“%”字符开头的注释行将被忽略。

文件javaclasspath.txt(或classpath.txt)的搜索规则如下:

首先，Octave尝试在当前目录(Octave是从哪里启动的)中定位它。如果找到这样的文件，就读取它并定义初始静态类路径。因此，可以在“每个Octave调用”的基础上定义静态类路径。

接下来，Octave在用户的主目录中搜索。如果这里存在一个文件javaclasspath.txt，那么它的内容将被附加到静态类路径(如果有的话)。因此，可以在“每个用户”的基础上构建初始静态类路径。

最后，Octave在Octave Java函数所在的m-file目录中查找javaclasspath.txt。这就是函数javaclass路径所在的位置。通常是OCTAVE\_HOME/share/octave/OCTAVE\_VERSION/m/java/。可以通过执行命令找到该目录

which javaclasspath

如果这个文件存在，它的内容也被附加到静态类路径中。注意，在最后一步中定义的归档文件和类目录将影响所有用户。

仅由特定脚本使用的类应该放在动态类路径中。可以在运行时使用javaaddpath和jav腋窝函数修改类路径的这一部分。

例子:

octave> base\_path = "C:/Octave/java\_files";

octave> # add two JAR archives to the dynamic classpath

octave> javaaddpath ([base\_path, "/someclasses.jar"]);

octave> javaaddpath ([base\_path, "/moreclasses.jar"]);

octave> # check the dynamic classpath

octave> p = javaclasspath;

octave> disp (p{1});

C:/Octave/java\_files/someclasses.jar

octave> disp (p{2});

C:/Octave/java\_files/moreclasses.jar

octave> # remove the first element from the classpath

octave> javarmpath ([base\_path, "/someclasses.jar"]);

octave> p = javaclasspath;

octave> disp (p{1});

C:/Octave/java\_files/moreclasses.jar

octave> # provoke an error

octave> disp (p{2});

error: A(I): Index exceeds matrix dimension.

向动态类路径中添加文件的另一种方法是使用.octaverc文件，该文件存储在主目录中。每次启动Octave的新实例时，都会执行此文件中的所有Octave命令。下面的示例将目录octave添加到octave的搜索路径中，并将该目录中的存档文件myclasses.jar添加到Java搜索路径中。

# contents of .octaverc:

addpath ("~/octave");

javaaddpath ("~/octave/myclasses.jar");

A.4.2如何在Octave中使用Java

函数javaObject创建Java对象。实际上，它调用具有给定名称和给定参数的类的公共构造函数。

下面的示例展示了如何调用内置Java类Java .math.BigDecimal的构造函数BigDecimal(double)和BigDecimal(String)。

javaObject ("java.math.BigDecimal", 1.001 );

javaObject ("java.math.BigDecimal", "1.001");

注意，Octave类型double的参数隐式地转换为Java类型double，而Octave类型char(数组)被转换为Java类型String。由javaObject创建的Java对象永远不会自动转换为Octave类型，而仍然是Java对象。它可以分配给一个Octave变量。

a = 1.001;

b = javaObject ("java.math.BigDecimal", a);

使用isjava，可以检查变量是否为Java对象，也可以确定其类。除了前面的例子:

isjava (a)

⇒ ans = 0

class (a)

⇒ ans = double

isjava (b)

⇒ ans = 1

class (b)

⇒ ans = java.math.BigDecimal

上面的例子可以只使用Java对象来执行:

a = javaObject ("java.lang.Double", 1.001);

b = javaObject ("java.math.BigDecimal", a);

isjava (a)

⇒ ans = 1

class (a)

⇒ ans = java.lang.Double

isjava (b)

⇒ ans = 1

class (b)

⇒ ans = java.math.BigDecimal

可以看到，当javaObject创建java.lang.Double时，即使是java.lang.Double也不会转换为Octave double。但是，如果Java类Java .lang. double或double是方法(或构造函数)的参数，则可能出现歧义。在这种情况下，它们可以根据上下文相互转换。

通过javaObject可以创建除数组以外的所有类型的Java对象。后者是通过javaArray创建的。

在Java语法中调用Java对象的公共成员方法是可能的:

a.toString

⇒ ans = 1.001

b.toString

⇒ ans = 1.000999999999999889865...

第二个结果可能令人惊讶，但它只是来自这样一个事实，即由于四舍五入，1.001不能精确地表示为双精度。请注意，与Java不同，在Octave中，不带参数的方法可以带或不带括号()调用。

目前，在Octave中不可能用类似Java的语法调用静态方法。相反，必须使用函数javaMethod，如下面的例子所示:

java.math.BigDecimal.valueOf(1.001); # does not work

javaMethod ("valueOf", "java.math.BigDecimal", 1.001); # workaround

如前所述，如果合适的话，方法和构造函数参数会在Octave和Java类型之间自动转换。对于函数，对于返回值也是如此，而对于构造函数，则不是这样。

也可以使用Java语法从Octave中访问Java对象的公共字段，但有静态字段的限制:

java.math.BigDecimal.ONE; # does not work

java\_get ("java.math.BigDecimal", "ONE"); # workaround

因此，使用java\_set可以设置字段的值。注意，只有公共Java字段可以从Octave中访问。

下面的例子表明，在Octave中，空括号[]表示Java的空值，以及如何表示Java异常。

javaObject ("java.math.BigDecimal", []);

⇒ error: [java] java.lang.NullPointerException

不建议用空括号[]表示Java的null值，因为null没有类型，而[]有double类型。

在Octave中，可以通过列出Java对象的公共字段和方法(静态或非静态)来提供有限的Java反射。

fieldnames (<Java object>)

methods (<Java object>)

最后，展示了如何从Octave中访问堆栈跟踪的示例，其中使用函数debug\_java设置和获取当前调试状态。在调试模式下，显示Java错误和堆栈跟踪信息。

debug\_java (true) # use "false" to omit display of stack trace

debug\_java ()

⇒ ans = 1

javaObject ("java.math.BigDecimal", "1") ...

.divide (javaObject ("java.math.BigDecimal", "0"))

**A.4.3设置JVM**

为了执行Java代码，Octave创建了一个Java虚拟机(JVM)。默认情况下，使用在类unix系统上配置时检测到的JVM版本，或者从HKEY\_LOCAL\_MACHINE\SOFTWARE\JavaSoft\JRE或Windows上的HKEY\_LOCAL\_MACHINE\SOFTWARE\JavaSoft\Java Runtime Environment的注册表项指向的JVM版本。可以通过将环境变量JAVA\_HOME设置为安装JVM的路径来覆盖JVM的默认路径。例如，在Windows上可能是C:\Program Files\Java\ jre-10.0.2。确保选择一个包含JVM的目录，其位位数与Octave的位位数匹配。

JVM在每个Octave会话中只加载一次。因此，要更改所使用的JVM版本，可能必须重新启动Octave。要检查当前正在使用的JVM版本，请运行version -java。

JVM分配固定数量的初始内存，并可以将此池扩展到固定的最大内存限制。默认值取决于Java版本(参见javamem)。内存池由JVM中运行的所有Java对象共享。这种严格的内存限制主要是为了避免web浏览器或企业服务器中失控的应用程序消耗所有内存并导致系统崩溃。当达到最大内存限制时，Java代码将抛出异常，从而导致应用程序失败或行为异常。

您可以在名为java.opts的文件中指定创建JVM的选项。这是一个文本文件，您在其中输入包含-X和-D选项的行，然后在初始化期间将这些选项传递给JVM。

Java选项文件所在的目录由环境变量OCTAVE\_JAVA\_DIR指定。如果不设置javaclasspath. path所在的目录。使用m驻留代替(通常是OCTAVE\_HOME/share/octave/OCTAVE\_VERSION/m/java/)。可以通过执行命令找到该目录

which javaclasspath

x选项允许您增加JVM可用的最大内存量。下面的示例允许使用256兆字节，方法是将以下行添加到java中。选择文件:

-Xmx256m

最大可能的内存量取决于您的系统。在主存为2g的Windows系统上，您应该能够将此最大值设置为1gb左右。

如果您的应用程序从一开始就需要大量内存，您还可以指定分配给JVM的初始内存量。将以下行添加到java。opts文件用64兆的初始内存启动JVM:

-Xms64m

有关Java虚拟机可用-X选项的更多详细信息，请在操作系统命令提示符下发出命令' Java -X '，并查阅Java文档。

可以使用-D选项来定义系统属性，然后由Octave中的Java类使用这些属性。系统属性可以通过使用java.lang.System类的getProperty()方法来检索。下面的示例行定义了属性MyProperty并将字符串12.34赋值给它。

-DMyProperty=12.34

这个属性的值可以通过Java对象或Octave作为字符串检索:

octave> javaMethod ("getProperty", "java.lang.System", "MyProperty");

ans = 12.34

A.4.4 Java接口函数

以下功能是Java接口的核心。它们提供了一种方法来创建Java对象，获取和设置其数据字段，以及调用返回结果给Octave的Java方法。

: jobj = javaObject (classname)

: jobj = javaObject (classname, arg1, …)

通过使用参数arg1，…调用类构造函数，创建类classsname的Java对象。

下面的第一个示例创建了一个未初始化的对象，而第二个示例为构造函数提供了一个初始参数。

x = javaObject ("java.lang.StringBuffer")

x = javaObject ("java.lang.StringBuffer", "Initial string")

参见:javaMethod, javaArray。

: jary = javaArray (classname, sz)

: jary = javaArray (classname, m, n, …)

创建一个大小为sz的Java数组，其元素为classname类。

classname可以是表示类的Java对象，也可以是包含完全限定类名的字符串。对象的大小也可以用单个整数参数m、n等来指定。

生成的数组未初始化。如果classname是引用类型，则所有元素都设置为null，如果classname是基本类型，则设置为默认值(通常为0)。

示例代码:

jary = javaArray ("java.lang.String", 2, 2);

jary(1,1) = "Hello";

参见:javaObject。

Octave中有许多不同的变量类型，但只有通过javaObject创建的变量类型才能使用Java函数。在对未知对象使用Java之前，可以使用isjava检查类型。

: tf = isjava (x)

如果x是Java对象则返回true。

参见:class, typeinfo, isa, javaObject。

一旦创建了对象，就很自然地找出对象有哪些字段，并读取(获取)和写入(设置)它们。

在Octave中，结构体的fieldnames函数被重载以返回Java对象的字段。例如:

dobj = javaObject ("java.lang.Double", pi);

fieldnames (dobj)

⇒

{

[1,1] = public static final double java.lang.Double.POSITIVE\_INFINITY

[1,2] = public static final double java.lang.Double.NEGATIVE\_INFINITY

[1,3] = public static final double java.lang.Double.NaN

[1,4] = public static final double java.lang.Double.MAX\_VALUE

[1,5] = public static final double java.lang.Double.MIN\_NORMAL

[1,6] = public static final double java.lang.Double.MIN\_VALUE

[1,7] = public static final int java.lang.Double.MAX\_EXPONENT

[1,8] = public static final int java.lang.Double.MIN\_EXPONENT

[1,9] = public static final int java.lang.Double.SIZE

[1,10] = public static final java.lang.Class java.lang.Double.TYPE

}

对象与结构的类比也适用于对象字段的读写。要读取字段，对象将使用'。'结构中的操作符。这是读取字段的首选方法，但是Octave还提供了一个使用java\_get读取字段的函数接口。下面是两种风格的示例。

dobj = javaObject ("java.lang.Double", pi);

dobj.MAX\_VALUE

⇒ 1.7977e+308

java\_get ("java.lang.Float", "MAX\_VALUE")

⇒ 3.4028e+38

: val = java\_get (obj, name)

获取Java对象obj的字段名的值。

对于静态字段，obj可以是表示相应类的完全限定名的字符串。

当obj是一个普通的Java对象时，类结构索引可以用作一种快捷语法。例如，下面两个语句是等价的

java\_get (x, "field1")

x.field1

参见:java\_set, javaMethod, javaObject。

: obj = java\_set (obj, name, val)

将Java对象obj的字段名的值设置为val。

对于静态字段，obj可以是表示相应Java类的完全限定名称的字符串。

当obj是一个普通的Java对象时，类结构索引可以用作一种快捷语法。例如，下面两个语句是等价的

java\_set (x, "field1", val)

x.field1 = val

参见:java\_get, javaMethod, javaObject。

要查看可以用对象调用哪些函数，请使用方法。例如，使用前面创建的dobj:

methods (dobj)

⇒

Methods for class java.lang.Double:

boolean equals(java.lang.Object)

java.lang.String toString(double)

java.lang.String toString()

…

调用具有相同结构索引操作符的对象的方法。’是常用的。Octave还提供了一个函数接口，可以通过javaMethod调用对象的方法。下面是展示这两种风格的示例。

dobj = javaObject ("java.lang.Double", pi);

dobj.equals (3)

⇒ 0

javaMethod ("equals", dobj, pi)

⇒ 1

: ret = javaMethod (methodname, obj)

: ret = javaMethod (methodname, obj, arg1, …)

用参数arg1， ....在Java对象obj上调用方法methodname

对于静态方法，obj可以是表示相应类的完全限定名的字符串。

当obj是一个普通的Java对象时，类结构索引可以用作一种快捷语法。例如，下面两个语句是等价的

ret = javaMethod ("method1", x, 1.0, "a string")

ret = x.method1 (1.0, "a string")

javaMethod返回方法调用的结果。

参见:methods, javaObject。

下面三个函数用于显示和修改Java虚拟机使用的类路径。这与Octave的PATH变量完全分离，JVM使用它来查找要执行的正确代码。

: javaclasspath ()

: dpath = javaclasspath ()

: [dpath, spath] = javaclasspath ()

: clspath = javaclasspath (what)

以字符串单元数组的形式返回Java虚拟机的类路径。

如果调用时没有输入:

如果不要求输出，则将动态和静态类路径打印到标准输出。

如果请求一个输出值dpath，则结果是动态类路径。

如果请求两个输出值dpath和spath，第一个变量将包含动态类路径，第二个变量将包含静态类路径。

如果用一个输入参数调用:

"-dynamic"

Return the dynamic classpath.

"-static"

Return the static classpath.

"-all"

Return both the static and dynamic classpath in a single cellstr.

参见:javaaddpath, jav腋窝。

: javaaddpath (clspath)

: javaaddpath (clspath1, …)

: javaaddpath ({clspath1, …})

: javaaddpath (…, "-end")

将clspath添加到Java虚拟机的动态类路径的开头。

clspath可以是.class文件所在的目录，也可以是包含Java类的.jar文件。可以通过指定附加参数或使用字符串的单元数组来一次添加多个路径。

如果最后一个参数是"-end"，则将新元素附加到当前类路径的末尾。

参见:jav腋窝，javaclass路径。

: javarmpath (clspath)

: javarmpath (clspath1, …)

: javarmpath ({clspath1, …})

从Java虚拟机的动态类路径中删除clspath。

clspath可以是.class文件所在的目录，也可以是包含Java类的.jar文件。可以通过指定附加参数或使用字符串的单元数组来一次删除多个路径。

参见:javaaddpath, javaclasspath。

下面的函数为Octave和Java虚拟机之间的接口提供信息和控制。

: msg = javachk (feature)

: msg = javachk (feature, caller)

检查当前会话中是否存在Java特性。如果功能不可用、未启用或无法识别，则返回错误结构。

可能识别的特征有:

"awt"

Abstract Window Toolkit for GUIs.

"desktop"

Interactive desktop is running.

"jvm"

Java Virtual Machine.

"swing"

Swing components for lightweight GUIs.

如果不支持feature，则返回带有字段“message”和“identifier”的标量结构体。字段“message”包含一个提到特性的错误消息以及可选的用户指定的调用者。这种结构适合传递给error函数。

如果feature被支持并且可用，则返回一个空的struct数组，包含字段"message"和"identifier"。

javachk确定特定的Java特性在Octave会话中是否可用。这个函数是为脚本提供的，脚本可以根据Java的可用性或特定的Java运行时特性来改变它们的行为。

兼容性注意:由于Octave没有基于java的桌面，所以“桌面”功能永远不可用。

参见:usejava, error。

: tf = usejava (feature)

如果Java元素特性可用，则返回true。

可能的特征有:

"awt"

Abstract Window Toolkit for GUIs.

"desktop"

Interactive desktop is running.

"jvm"

Java Virtual Machine.

"swing"

Swing components for lightweight GUIs.

usejava确定特定的Java特性是否在Octave会话中可用。这个函数是为脚本提供的，这些脚本可能会根据Java的可用性改变它们的行为。特性“desktop”总是返回false，因为Octave没有基于java的桌面。如果使用Java接口编译Octave并安装了Java，则可以使用其他功能。

参见:javachk。

: javamem ()

: jmem = javamem ()

显示Java虚拟机(JVM)的当前内存使用情况并运行垃圾收集器。

当没有返回参数时，信息被打印到屏幕上。否则，输出单元数组jmem包含Maximum、Total和Free内存(以字节为单位)。

所有基于java的例程都在JVM的共享内存池中运行，这是JVM从计算机的总内存(包括物理RAM和硬盘上的虚拟内存/交换空间)中声明的专用且独立的内存部分。

可以使用文件java.opts配置允许的最大内存使用量。该文件所在的目录由环境变量OCTAVE\_JAVA\_DIR决定。如果未设置，则javaaddpath. path所在的目录。使用m驻留代替(通常是OCTAVE\_HOME/share/octave/OCTAVE\_VERSION/m/java/)。

java。Opts是一个纯文本文件，每行有一个选项。默认的初始内存大小和默认的最大内存大小(都依赖于系统)可以像这样重写:

-Xms64m

-Xmx512m

(本例中以兆字节为单位)。如果系统的可用物理内存有限，或者出现Java内存错误，可以根据自己的需求调整这些值。

“总内存”是操作系统当前分配给JVM的内存，取决于实际的和活动的内存使用情况。“空闲内存”是不言自明的。在基于Java的Octave函数的操作期间，Total和Free内存的数量会有所不同，这是由于Java自己的清理和操作系统的内存管理。

: val = java\_matrix\_autoconversion ()

: old\_val = java\_matrix\_autoconversion (new\_val)

: old\_val = java\_matrix\_autoconversion (new\_val, "local")

查询或设置控制Java数组是否自动转换为Octave矩阵的内部变量。

默认值为false。

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:java\_unsigned\_autoconversion, debug\_java。

: val = java\_unsigned\_autoconversion ()

: old\_val = java\_unsigned\_autoconversion (new\_val)

: old\_val = java\_unsigned\_autoconversion (new\_val, "local")

查询或设置内部变量，该变量控制在启用java\_matrix\_autoconversion时如何转换整数类。

启用后，Byte或Integer类的Java数组将分别转换为uint8或uint32类的矩阵。默认值为true。

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:java\_matrix\_autoconversion, debug\_java。

: val = debug\_java ()

: old\_val = debug\_java (new\_val)

: old\_val = debug\_java (new\_val, "local")

查询或设置内部变量，该变量决定是否打印有关JVM初始化和任何Java异常的额外调试信息。

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:java\_matrix\_autoconversion, java\_unsigned\_autoconversion。

**附录B测试和演示功能**

Octave包含许多函数，允许在函数本身的源代码中集成测试和演示代码。

**B.1测试功能**

: test name

: test name quiet|normal|verbose

: test ("name", "quiet|normal|verbose", fid)

: test ("name", "quiet|normal|verbose", fname)

: success = test (…)

: [n, nmax, nxfail, nbug, nskip, nrtskip, nregression] = test (…)

: [code, idx] = test ("name", "grabdemo")

: test ([], "explain", fid)

: test ([], "explain", fname)

从loadpath匹配名称中的第一个文件执行内置自我测试。

Test可以以命令或函数的形式调用。测试的确切操作是由模式(交互式或批处理)、报告级别(“安静”、“正常”、“详细”)以及是否使用日志文件或摘要输出变量的组合决定的。

从命令行调用test时的默认模式是交互式的。在此模式下，测试将一直运行，直到遇到第一个错误或所有测试成功完成。在批处理模式下，无论出现任何故障，都将运行所有测试，并收集结果以供报告。需要用户交互的测试，即演示块，永远不会以批处理模式运行。

批处理模式可以通过以下两种方式启用:1)使用第三个参数fname或fid指定日志文件，或2)请求输出参数，如success、n等。

可选的第二个参数确定要生成的输出量以及要运行的测试类型。默认值为normal。请求输出参数将禁止打印最终摘要消息和任何中间警告，除非启用了详细报告。

"quiet"

当所有测试都通过时打印摘要消息，或者在发生失败时打印带有第一个错误测试结果的错误。不要运行需要用户交互的测试。

"normal"

在测试执行期间显示有关跳过测试或xtest失败的警告消息。当所有测试都通过时打印摘要消息，或者在发生失败时打印带有第一个错误测试结果的错误。不要运行需要用户交互的测试。

"verbose"

执行前显示测试。打印所有警告信息。在交互模式下，运行所有测试，包括那些需要用户交互的测试。

可选的第三个输入参数指定一个日志文件，应该在其中写入测试结果。日志文件可以是字符串(fname)或打开的文件描述符ID (fid)。要启用批处理，但仍将结果打印到屏幕上，请使用stdout for fid。

当只使用一个输出参数success调用时，如果所有测试都成功，则test返回true。如果调用时带有多个输出参数，则返回成功测试的次数(n)、文件中的测试总数(nmax)、xtest失败的次数(nxfail)、由于已知错误而失败的测试次数(nbug)、由于缺少特性而跳过的测试次数(nskip)、由于运行时条件而跳过的测试次数(nrtskip)和回归的次数(nregression)。

示例

test sind

⇒

PASSES 5 out of 5 tests

[n, nmax] = test ("sind")

⇒

n = 5

nmax = 5

附加的调用语法

如果第二个参数是字符串"grabdemo"，则提取任何内置demo块的内容，但不执行。所有代码块的文本被连接起来并作为代码返回，其中idx是每个演示块末端位置的向量。有关从文件中提取演示块的更简单方法，请参见example。

如果第二个参数是“explain”，则忽略name，并将测试输出报告中使用的行标记的解释写入由fname或fid指定的文件。

参见:assert, fail, demo, example, error。

Test扫描命名的脚本文件，查找以标识符“%!”开头的行。前缀被去掉，行其余部分通过Octave解释器处理。如果代码产生一个错误，那么测试就会失败。

由于eval()将在遇到第一个错误时停止，因此必须将测试划分为块，单独块中的任何内容分别求值。块由紧接在' %! '后面的有效关键字(如test、function或assert)引入。和Python一样，块是通过缩进定义的。以“%!”<空格> '是前一个块的一部分。

例如:

%!test error ("this test fails!")

%!test "test doesn't fail. it doesn't generate an error"

当测试失败时，您将看到如下内容:

\*\*\*\*\* test error ("this test fails!")

!!!!! test failed

this test fails!

通常，要测试某些东西是否有效，您需要断言它产生了正确的值。真正的测试可能是这样的

%!test

%! a = [1, 2, 3; 4, 5, 6]; B = [1; 2];

%! expect = [ a ; 2\*a ];

%! get = kron (b, a);

%! if (any (size (expect) != size (get)))

%! error ("wrong size: expected %d,%d but got %d,%d",

%! size (expect), size (get));

%! elseif (any (any (expect != get)))

%! error ("didn't get what was expected.");

%! endif

为了简化这个过程，可以使用assert函数。例如，使用assert，前面的测试被简化为:

%!test

%! a = [1, 2, 3; 4, 5, 6]; b = [1; 2];

%! assert (kron (b, a), [ a; 2\*a ]);

assert可以接受一个公差，以便您可以绝对地或相对地比较结果。例如，以下都成功:

%!test assert (1+eps, 1, 2\*eps) # absolute error

%!test assert (100+100\*eps, 100, -2\*eps) # relative error

你也可以自己做比较，但仍然需要assert生成错误:

%!test assert (isempty ([]))

%!test assert ([1, 2; 3, 4] > 0)

因为assert经常单独在测试块中使用，所以有一个简写形式:

%!assert (…)

这相当于:

%!test assert (…)

有时候，一个测试块将依赖于Octave中的可选功能。在测试这些块之前，必须检查所需功能的可用性。% !testf HAVE\_XXX块只会在Octave使用' HAVE\_XXX '功能编译时运行。例如，稀疏单值分解svds()依赖于ARPACK库。所有svd的测试都以

%!testif HAVE\_ARPACK

查看config.h或\_\_octave\_config\_info\_\_ ("build\_features")以查看一些可能需要检查的值。

有时在开发过程中，会有一个应该可以工作的测试，但已知会失败。您仍然希望保留测试，因为当最终代码准备好时，测试应该通过，但是您可能无法立即修复它。为了避免不必要的错误报告这些已知的失败，用xtest而不是test标记块:

%!xtest assert (1==0)

%!xtest fail ("success=1", "error")

在这种情况下，测试将运行，任何失败都将被报告。但是，测试不会中止，后续测试块将正常处理。xtest的另一个用途是用于统计测试，这些测试在大多数情况下应该可以通过，但偶尔会失败。

每个块在其自己的函数环境中求值，这意味着在一个块中定义的变量不会自动与其他块共享。如果您确实想共享变量，那么必须在使用它们之前将它们声明为共享。例如，下面声明变量a，给它一个初始值(默认为空)，然后在随后的几个测试中使用它。

%!shared a

%! a = [1, 2, 3; 4, 5, 6];

%!assert (kron ([1; 2], a), [ a; 2\*a ])

%!assert (kron ([1, 2], a), [ a, 2\*a ])

%!assert (kron ([1,2; 3,4], a), [ a,2\*a; 3\*a,4\*a ])

您可以同时共享多个变量:

%!shared a, b

只有当测试成功时，对共享变量的修改才会从一个测试持续到下一个测试。因此，如果一个测试修改了共享变量，以后的测试将无法知道期望共享变量的哪个值，因为早期测试的通过/失败状态是未知的。因此，不建议在测试中修改共享变量。

你也可以共享试函数:

%!function a = fn (b)

%! a = 2\*b;

%!endfunction

%!assert (fn(2), 4)

注意，当声明一个新的共享块时，所有以前的变量和值都会丢失。

记住%!函数开始一个新的块，并且%!Endfunction结束这个块。请注意，在开始一个新块之前，以' %!<space> '将作为注释被丢弃。下面的代码几乎与上面的示例相同，但什么也没做。

%!function a = fn (b)

%! a = 2\*b;

%!endfunction

%! assert (fn(2), 4)

因为“%!”后面有一个空格。， assert语句不开始一个新的块，这一行被当作注释处理。

错误和警告块类似于测试块，但它们只有在代码产生错误时才会成功。可以使用可选的正则表达式<pattern>检查错误文本是否正确。例如:

%!error <passes!> error ("this test passes!")

如果代码没有生成错误，则测试失败。例如:

%!error "this is an error because it succeeds."

产生

\*\*\*\*\* error "this is an error because it succeeds."

!!!!! test failed: no error

尽可能自动化测试是很重要的，但是有些测试需要用户交互。可以将它们隔离到演示块中，如果您处于批处理模式，则仅在使用demo或verbose选项进行测试时调用演示块。代码在执行前显示。例如,

%!demo

%! t = [0:0.01:2\*pi]; x = sin (t);

%! plot (t, x);

%! # you should now see a sine wave in your figure window

产生

funcname example 1:

t = [0:0.01:2\*pi]; x = sin (t);

plot (t, x);

# you should now see a sine wave in your figure window

Press <enter> to continue:

注意，演示块不能使用任何共享变量。这样它们就可以自己执行，而忽略所有其他测试。

如果您想暂时禁用测试块，请将#放在块类型的位置。这会创建一个注释块，该注释块在日志文件中回显，但不会执行。例如:

%!#demo

%! t = [0:0.01:2\*pi]; x = sin (t);

%! plot (t, x);

%! # you should now see a sine wave in your figure window

下面的代码片段提供了使用fail、assert、error和xtest的示例:

function output = must\_be\_zero (input)

if (input != 0)

error ("Nonzero input!")

endif

output = input;

endfunction

%!fail ("must\_be\_zero (1)")

%!assert (must\_be\_zero (0), 0)

%!error <Nonzero> must\_be\_zero (1)

%!xtest error ("This code generates an error")

当将其放入文件中时，必须\_be\_zero。M，运行测试，我们看到

test must\_be\_zero verbose

⇒

>>>>> /path/to/must\_be\_zero.m

\*\*\*\*\* fail ("must\_be\_zero (1)")

\*\*\*\*\* assert (must\_be\_zero (0), 0)

\*\*\*\*\* error <Nonzero> must\_be\_zero (1)

\*\*\*\*\* xtest error ("This code generates an error")

!!!!! known failure

This code generates an error

PASSES 3 out of 4 tests (1 expected failure)

块类型总结:

%!test

%!test <MESSAGE>

检查整个块是否正确。如果<MESSAGE>存在，测试块被解释为xtest。

%!testif HAVE\_XXX

%!testif HAVE\_XXX, HAVE\_YYY, …

%!testif HAVE\_XXX, HAVE\_YYY …; RUNTIME\_COND

%!testif … <MESSAGE>

只有当Octave用HAVE\_XXX特性编译时才会检查阻塞。RUNTIME\_COND是一个可选表达式，用于在执行测试时检查是否满足某些条件。如果RUNTIME\_COND为false，则跳过测试。如果<MESSAGE>存在，测试块被解释为xtest。

%!xtest

%!xtest <MESSAGE>

检查阻塞，报告测试失败，但不要中止测试。如果<MESSAGE>存在，则如果测试失败，则显示消息的文本，如下所示:

!!!!! known bug: MESSAGE

如果消息是整数，则将其解释为Octave错误跟踪器的错误ID，并报告为

!!!!! known bug: https://octave.org/testfailure/?BUG-ID

其中bug - id为整数错误号。这样做的目的是为了更清晰地记录已知问题。

如果MESSAGE是前面有一个星号的整数(例如，\*12345)，它被解释为已经关闭的bug报告的id。这通常意味着这个测试中探测到的问题已经解决了。如果此类测试失败，则测试函数将其报告为回归:

!!!!! regression: https://octave.org/testfailure/?BUG-ID

%!error

%!error <MESSAGE>

%!warning

%!warning <MESSAGE>

检查是否有正确的错误或警告消息。如果提供了<MESSAGE>，则将其解释为期望与错误或警告消息匹配的正则表达式模式。

%!demo

Demo只在交互模式下执行。

%!#

发表评论。忽略块中的所有内容

%!shared x,y,z

声明用于多个测试的变量。

%!function

定义一个用于多个测试的函数。

%!endfunction

关闭函数定义。

%!assert (x, y, tol)

%!assert <MESSAGE> (x, y, tol)

%!fail (CODE, PATTERN)

%!fail <MESSAGE> (CODE, PATTERN)

%!的简写测试assert (x, y, tol)或%!测试失败(CODE, PATTERN)。如果<MESSAGE>存在，测试块被解释为xtest。

在编码测试时，Octave约定以块类型开始的行在末尾没有分号。但是，块内的任何代码都是正常的Octave代码，并且通常后面有一个分号。例如,

## bare block instantiation

%!assert (sin (0), 0)

但是

## test block with normal Octave code

%!test

%! assert (sin (0), 0);

您还可以为内置函数和您自己的c++函数创建测试脚本。为此，在加载路径的目录中放置一个带有裸函数名(没有.m扩展名)的文件，测试函数将发现它。或者，你可以直接在c++代码中嵌入测试:

/\*

%!test disp ("this is a test")

\*/

或者

#if 0

%!test disp ("this is a test")

#endif

然而，在这种情况下，原始源代码需要在加载路径上，用户必须记住进行类型测试(“funcname.cc”)。

: assert (cond)

: assert (cond, errmsg)

: assert (cond, errmsg, …)

: assert (cond, msg\_id, errmsg, …)

: assert (observed, expected)

: assert (observed, expected, tol)

如果不满足指定的条件，则产生错误。

assert可以通过三种不同的方式调用。

assert (cond)

assert (cond, errmsg)

assert (cond, errmsg, …)

assert (cond, msg\_id, errmsg, …)

使用单个参数cond调用时，如果cond为false(数字零)，assert将产生错误。

任何附加参数都传递给error函数进行处理。

assert (observed, expected)

如果观察到的与预期的不相同，则产生错误。

注意，观察到的和期望的可以是标量、向量、矩阵、字符串、单元数组或结构。

assert (observed, expected, tol)

如果观察到的结果与预期的不一样，则产生错误，但是对于数值数据的相等性比较使用容差工具。

如果tol为正，则绝对公差，如果abs(观察-预期)> abs (tol)，则会产生误差。

如果tol是负的，那么它是一个相对公差，如果abs(观察到的-期望的)> abs (tol \*期望的)将产生一个错误。

如果期望为零，总将被解释为绝对容忍。

如果工具不是标量，则其维度必须与观察到的和预期的维度一致，并且测试是在逐个元素的基础上执行的。

参见:fail, test, error, isequal。

: status = fail (code)

: status = fail (code, pattern)

: status = fail (code, "warning")

: status = fail (code, "warning", pattern)

如果代码失败并产生匹配模式的错误消息，则返回true，否则产生错误。

代码必须是通过evalin函数传递给Octave解释器的字符串形式，即(带引号的)字符串常量或字符串变量。

注意，如果代码成功运行，而不是失败，打印的错误是:

expected error <.> but got none

如果带两个参数调用，只有当代码失败并出现包含模式(区分大小写)的错误消息时，返回值才为真。如果代码失败并产生与pattern中指定的错误不同的错误，则产生的消息是:

expected <pattern>

but got <text of actual error>

尖括号不是输出的一部分。

当使用"warning"选项调用fail时，如果执行代码没有产生警告，则会产生错误。

参见:assert, error。

**B.2演示功能**

: demo name

: demo name n

: demo ("name")

: demo ("name", n)

运行与函数名相关的示例代码块n。

如果不指定n，则运行所有示例。

示例代码块的首选位置嵌入在脚本m文件中，紧跟着它执行的代码。或者，示例可以存储在具有相同名称但没有扩展名的文件中，该文件位于Octave的加载路径上。为了将示例与常规脚本代码分开，所有行都以%!作为前缀。每个示例还必须由关键字“demo”引入，与前缀左对齐，中间没有空格。示例的其余部分可以包含任意的Octave代码。例如:

%!demo

%! t = 0:0.01:2\*pi;

%! x = sin (t);

%! plot (t, x);

%! title ("one cycle of a sine wave");

%! #-------------------------------------------------

%! # the figure window shows one cycle of a sine wave

请注意，代码在执行之前会显示，因此在代码末尾的简单注释足以标记所显示的内容。对于图表，标注也可以用标题或文本来完成。在演示中通常不需要使用disp或printf。

演示在独立的函数环境中运行，不能访问外部变量。这意味着每个演示都必须有单独的初始化代码。或者，所有的演示都可以与代码组合成一个大的演示

%! input ("Press <enter> to continue: ", "s");

在节之间，但不鼓励这样使用。避免多个初始化块的其他技术包括在每个图之间使用多个图，或者使用subplot在同一窗口中放置多个图。

最后，由于demo在函数上下文中求值，因此不可能在代码中定义新函数。在大多数情况下，匿名函数是一个很好的替代品。如果必须使用函数块，则代码eval (example ("function"， n))将允许Octave查看它们。然而，这也有它自己的问题，因为eval一次只计算一行或一条语句。在这种情况下，函数声明必须用“if 1 <demo stuff> endif”包装，其中“if”与“demo”在同一行。例如:

%!demo if 1

%! function y = f(x)

%! y = x;

%! endfunction

%! f(3)

%! endif

参见:rundemos, example, test。

: example name

: example name n

: example ("name")

: example ("name", n)

: [codestr, codeidx] = example (…)

显示与函数名相关联的代码(例如n)，但不运行它。

如果不指定n，则显示所有示例。

当使用输出参数调用时，示例将以字符串编码器的形式返回，codeidx表示各种示例的结束位置。

完整的解释请参见demo。

参见:demo, test。

: oruntests ()

: oruntests (directory)

对指定目录中的所有m文件执行内置测试。

任何c++源文件(\*.cc)中的测试块也将被执行，以用于动态链接的oct-file函数。

如果没有指定目录，在Octave的函数搜索路径中对所有目录进行操作。

参见:rundemos, test, path。

: rundemos ()

: rundemos (directory)

为指定目录中的所有m文件执行内置演示。

任何c++源文件(\*.cc)中的演示块也将被执行，以便与动态链接的oct-file函数一起使用。

如果没有指定目录，在Octave的函数搜索路径中对所有目录进行操作。

参见:demo, oruntests, path。

: speed (f, init, max\_n, f2, tol)

: [order, n, T\_f, T\_f2] = speed (…)

确定不同输入值(n)的表达式(f)的执行时间。

n是对数间隔的，从1到max\_n。对于每个n，计算一个初始化表达式(init)来创建测试所需的任何数据。如果给出第二个表达式(f2)，则比较两个表达式的执行时间。当不带输出参数调用时，结果将打印到标准输出并以图形方式显示。

f

要计算的代码表达式。

max\_n

要运行的最大测试长度。缺省值是100。或者，使用[min\_n, max\_n]或用[n1, n2，…，nk]精确指定n。

init

函数参数值的初始化表达式。用k表示测试数，用n表示测试的大小。这将计算f使用的所有变量的值。注意，init将首先在k = 0时进行计算，因此在整个测试系列中不变的东西可以计算一次。默认值是x = randn (n, 1)。

f2

一个可供选择的表达式求值，以便可以直接比较两个表达式的速度。默认为[]。

tol

公差用于比较表达f和表达f2的结果。如果tol是正的，则公差是绝对的。如果tol为负，则公差是相对的。默认值是eps。如果tol为Inf，则不会进行比较。

order

表达式O(a\*n^p)的时间复杂度。这是一个有场a和p的结构。

n

计算表达式的值n且执行时间大于零。

T\_f

以秒为单位记录的表达式f的非零执行时间。

T\_f2

表达式f2记录的非零执行时间(以秒为单位)。如果需要，平均时间比率只需mean (T\_f ./ T\_f2)。

执行时间图的斜率表示渐近运行时间O(n^p)的近似幂。这个幂是为它所逼近的区域(图的后半部分)绘制的。估计的功率不是很准确，但应该足以确定算法的一般顺序。例如，它应该指出，如果实现意外地是O(n^2)而不是O(n)，因为它每次通过循环扩展一个向量，而不是预先分配存储。在当前版本的Octave中，以下不是预期的O(n)。

speed ("for i = 1:n, y{i} = x(i); endfor", "", [1000, 10000])

但如果你预先分配单元格数组y:

speed ("for i = 1:n, y{i} = x(i); endfor", ...

"x = rand (n, 1); y = cell (size (x));", [1000, 10000])

人们试图估算单个操作的成本，但这是非常不准确的。你可以通过对每个n做更多的功来提高稳定性。例如:

speed ("airy(x)", "x = rand (n, 10)", [10000, 100000])

当比较两个不同的表达式(f, f2)时，如果新表达式更快，则加速比图上的直线斜率应大于1。更好的算法具有较浅的斜率。通常，对算法进行矢量化不会改变执行时间图的斜率，但会使其相对于原始时间图发生偏移。例如:

speed ("sum (x)", "", [10000, 100000], ...

"v = 0; for i = 1:length (x), v += x(i); endfor")

下面是一个更复杂的例子。如果xcorr的原始版本使用for循环，而第二个版本使用FFT，那么可以比较各种滞后的运行速度，或者使用不同向量长度的固定滞后，如下所示:

speed ("xcorr (x, n)", "x = rand (128, 1);", 100,

"xcorr\_orig (x, n)", -100\*eps)

speed ("xcorr (x, 15)", "x = rand (20+n, 1);", 100,

"xcorr\_orig (x, n)", -100\*eps)

假设两个版本中的一个在xcorr\_origin中，这将比较它们的速度和输出值。请注意，FFT版本并不精确，因此必须在比较100\*eps时指定可接受的公差。在这种情况下，比较应该相对计算，如abs ((x - y). / y)，而不是绝对计算abs (x - y)。

输入example(“speed”)可以看到一些真实的例子，或者输入demo(“speed”)来运行它们。

**附录C废弃函数**

在两个主要版本中被标记为已弃用后，以下函数已从Octave中删除。该表的第三列显示了该函数被删除的Octave版本。在删除之前，列表中的每个函数至少在两个主要版本中被标记为已弃用。所有弃用的函数都会发出警告，解释它们将在Octave的未来版本中被删除，以及应该使用哪个函数。

替换函数并不总是接受与废弃函数完全相同的参数，但应该提供相同的功能。

| **Obsolete Function** | **Replacement** | **Version** |
| --- | --- | --- |
| *beta\_cdf* | *betacdf* in Octave Forge statistics pkg | 3.4.0 |
| *beta\_inv* | *betainv* in Octave Forge statistics pkg | 3.4.0 |
| *beta\_pdf* | *betapdf* in Octave Forge statistics pkg | 3.4.0 |
| *beta\_rnd* | *betarnd* in Octave Forge statistics pkg | 3.4.0 |
| *binomial\_cdf* | *binocdf* in Octave Forge statistics pkg | 3.4.0 |
| *binomial\_inv* | *binoinv* in Octave Forge statistics pkg | 3.4.0 |
| *binomial\_pdf* | *binopdf* in Octave Forge statistics pkg | 3.4.0 |
| *binomial\_rnd* | *binornd* in Octave Forge statistics pkg | 3.4.0 |
| *chisquare\_cdf* | *chi2cdf* in Octave Forge statistics pkg | 3.4.0 |
| *chisquare\_inv* | *chi2inv* in Octave Forge statistics pkg | 3.4.0 |
| *chisquare\_pdf* | *chi2pdf* in Octave Forge statistics pkg | 3.4.0 |
| *chisquare\_rnd* | *chi2rnd* in Octave Forge statistics pkg | 3.4.0 |
| *clearplot* | *clf* | 3.4.0 |
| *com2str* | *num2str* | 3.4.0 |
| *exponential\_cdf* | *expcdf* in Octave Forge statistics pkg | 3.4.0 |
| *exponential\_inv* | *expinv* in Octave Forge statistics pkg | 3.4.0 |
| *exponential\_pdf* | *exppdf* in Octave Forge statistics pkg | 3.4.0 |
| *exponential\_rnd* | *exprnd* in Octave Forge statistics pkg | 3.4.0 |
| *f\_cdf* | *fcdf* in Octave Forge statistics pkg | 3.4.0 |
| *f\_inv* | *finv* in Octave Forge statistics pkg | 3.4.0 |
| *f\_pdf* | *fpdf* in Octave Forge statistics pkg | 3.4.0 |
| *f\_rnd* | *frnd* in Octave Forge statistics pkg | 3.4.0 |
| *gamma\_cdf* | *gamcdf* in Octave Forge statistics pkg | 3.4.0 |
| *gamma\_inv* | *gaminv* in Octave Forge statistics pkg | 3.4.0 |
| *gamma\_pdf* | *gampdf* in Octave Forge statistics pkg | 3.4.0 |
| *gamma\_rnd* | *gamrnd* in Octave Forge statistics pkg | 3.4.0 |
| *geometric\_cdf* | *geocdf* in Octave Forge statistics pkg | 3.4.0 |
| *geometric\_inv* | *geoinv* in Octave Forge statistics pkg | 3.4.0 |
| *geometric\_pdf* | *geopdf* in Octave Forge statistics pkg | 3.4.0 |
| *geometric\_rnd* | *geornd* in Octave Forge statistics pkg | 3.4.0 |
| *hypergeometric\_cdf* | *hygecdf* in Octave Forge statistics pkg | 3.4.0 |
| *hypergeometric\_inv* | *hygeinv* in Octave Forge statistics pkg | 3.4.0 |
| *hypergeometric\_pdf* | *hygepdf* in Octave Forge statistics pkg | 3.4.0 |
| *hypergeometric\_rnd* | *hygernd* in Octave Forge statistics pkg | 3.4.0 |
| *intersection* | *intersect* | 3.4.0 |
| *is\_bool* | *isbool* | 3.4.0 |
| *is\_complex* | *iscomplex* | 3.4.0 |
| *is\_list* | None | 3.4.0 |
| *is\_matrix* | *ismatrix* | 3.4.0 |
| *is\_scalar* | *isscalar* | 3.4.0 |
| *is\_square* | *issquare* | 3.4.0 |
| *is\_stream* | None | 3.4.0 |
| *is\_struct* | *isstruct* | 3.4.0 |
| *is\_symmetric* | *issymmetric* | 3.4.0 |
| *is\_vector* | *isvector* | 3.4.0 |
| *lognormal\_cdf* | *logncdf* in Octave Forge statistics pkg | 3.4.0 |
| *lognormal\_inv* | *logninv* in Octave Forge statistics pkg | 3.4.0 |
| *lognormal\_pdf* | *lognpdf* in Octave Forge statistics pkg | 3.4.0 |
| *lognormal\_rnd* | *lognrnd* in Octave Forge statistics pkg | 3.4.0 |
| *meshdom* | *meshgrid* | 3.4.0 |
| *normal\_cdf* | *normcdf* in Octave Forge statistics pkg | 3.4.0 |
| *normal\_inv* | *norminv* in Octave Forge statistics pkg | 3.4.0 |
| *normal\_pdf* | *normpdf* in Octave Forge statistics pkg | 3.4.0 |
| *normal\_rnd* | *normrnd* in Octave Forge statistics pkg | 3.4.0 |
| *pascal\_cdf* | *nbincdf* in Octave Forge statistics pkg | 3.4.0 |
| *pascal\_inv* | *nbininv* in Octave Forge statistics pkg | 3.4.0 |
| *pascal\_pdf* | *nbinpdf* in Octave Forge statistics pkg | 3.4.0 |
| *pascal\_rnd* | *nbinrnd* in Octave Forge statistics pkg | 3.4.0 |
| *poisson\_cdf* | *poisscdf* in Octave Forge statistics pkg | 3.4.0 |
| *poisson\_inv* | *poissinv* in Octave Forge statistics pkg | 3.4.0 |
| *poisson\_pdf* | *poisspdf* in Octave Forge statistics pkg | 3.4.0 |
| *poisson\_rnd* | *poissrnd* in Octave Forge statistics pkg | 3.4.0 |
| *polyinteg* | *polyint* | 3.4.0 |
| *struct\_contains* | *isfield* | 3.4.0 |
| *struct\_elements* | *fieldnames* | 3.4.0 |
| *t\_cdf* | *tcdf* in Octave Forge statistics pkg | 3.4.0 |
| *t\_inv* | *tinv* in Octave Forge statistics pkg | 3.4.0 |
| *t\_pdf* | *tpdf* in Octave Forge statistics pkg | 3.4.0 |
| *t\_rnd* | *trnd* in Octave Forge statistics pkg | 3.4.0 |
| *uniform\_cdf* | *unifcdf* in Octave Forge statistics pkg | 3.4.0 |
| *uniform\_inv* | *unifinv* in Octave Forge statistics pkg | 3.4.0 |
| *uniform\_pdf* | *unifpdf* in Octave Forge statistics pkg | 3.4.0 |
| *uniform\_rnd* | *unifrnd* in Octave Forge statistics pkg | 3.4.0 |
| *weibull\_cdf* | *wblcdf* in Octave Forge statistics pkg | 3.4.0 |
| *weibull\_inv* | *wblinv* in Octave Forge statistics pkg | 3.4.0 |
| *weibull\_pdf* | *wblpdf* in Octave Forge statistics pkg | 3.4.0 |
| *weibull\_rnd* | *wblrnd* in Octave Forge statistics pkg | 3.4.0 |
| *wiener\_rnd* | *wienrnd* in Octave Forge statistics pkg | 3.4.0 |
| *create\_set* | *unique* | 3.6.0 |
| *dmult* | *diag (A) \* B* | 3.6.0 |
| *iscommand* | None | 3.6.0 |
| *israwcommand* | None | 3.6.0 |
| *lchol* | *chol (…, "lower")* | 3.6.0 |
| *loadimage* | *load* or *imread* | 3.6.0 |
| *mark\_as\_command* | None | 3.6.0 |
| *mark\_as\_rawcommand* | None | 3.6.0 |
| *spatan2* | *atan2* | 3.6.0 |
| *spchol* | *chol* | 3.6.0 |
| *spchol2inv* | *chol2inv* | 3.6.0 |
| *spcholinv* | *cholinv* | 3.6.0 |
| *spcumprod* | *cumprod* | 3.6.0 |
| *spcumsum* | *cumsum* | 3.6.0 |
| *spdet* | *det* | 3.6.0 |
| *spdiag* | *sparse (diag (…))* | 3.6.0 |
| *spfind* | *find* | 3.6.0 |
| *sphcat* | *horzcat* | 3.6.0 |
| *spinv* | *inv* | 3.6.0 |
| *spkron* | *kron* | 3.6.0 |
| *splchol* | *chol (…, "lower")* | 3.6.0 |
| *split* | *char (strsplit (s, t))* | 3.6.0 |
| *splu* | *lu* | 3.6.0 |
| *spmax* | *max* | 3.6.0 |
| *spmin* | *min* | 3.6.0 |
| *spprod* | *prod* | 3.6.0 |
| *spqr* | *qr* | 3.6.0 |
| *spsum* | *sum* | 3.6.0 |
| *spsumsq* | *sumsq* | 3.6.0 |
| *spvcat* | *vertcat* | 3.6.0 |
| *str2mat* | *char* | 3.6.0 |
| *unmark\_command* | None | 3.6.0 |
| *unmark\_rawcommand* | None | 3.6.0 |
| *autocor* | *xcorr* in Octave Forge signal pkg | 3.8.0 |
| *autocov* | *xcov* in Octave Forge signal pkg | 3.8.0 |
| *betai* | *betainc* | 3.8.0 |
| *cellidx* | *ismember* | 3.8.0 |
| *cquad* | *quadcc* | 3.8.0 |
| *dispatch* | None | 3.8.0 |
| *fstat* | *stat* | 3.8.0 |
| *gammai* | *gammainc* | 3.8.0 |
| *glpkmex* | *glpk* | 3.8.0 |
| *is\_duplicate\_entry* | *unique* | 3.8.0 |
| *is\_global* | *isglobal* | 3.8.0 |
| *krylovb* | *[Uret, ~, Ucols] = krylov (…)* | 3.8.0 |
| *perror* | None | 3.8.0 |
| *replot* | *refresh* | 3.8.0 |
| *saveimage* | *imwrite* | 3.8.0 |
| *setstr* | *char* | 3.8.0 |
| *strerror* | None | 3.8.0 |
| *values* | *unique* | 3.8.0 |
| *cut* | *histc* | 4.0.0 |
| *cor* | *corr* | 4.0.0 |
| *corrcoef* | *corr* | 4.0.0 |
| *\_\_error\_text\_\_* | *lasterr* | 4.0.0 |
| *error\_text* | *lasterr* | 4.0.0 |
| *polyderiv* | *polyder* | 4.0.0 |
| *shell\_cmd* | *system* | 4.0.0 |
| *studentize* | *zscore* | 4.0.0 |
| *sylvester\_matrix* | *hadamard (2^k)* | 4.0.0 |
| *default\_save\_options* | *save\_default\_options* | 4.2.0 |
| *gen\_doc\_cache* | *doc\_cache\_create* | 4.2.0 |
| *interp1q* | *interp1* | 4.2.0 |
| *isequalwithequalnans* | *isequaln* | 4.2.0 |
| *java\_convert\_matrix* | *java\_matrix\_autoconversion* | 4.2.0 |
| *java\_debug* | *debug\_java* | 4.2.0 |
| *java\_invoke* | *javaMethod* | 4.2.0 |
| *java\_new* | *javaObject* | 4.2.0 |
| *java\_unsigned\_conversion* | *java\_unsigned\_autoconversion* | 4.2.0 |
| *javafields* | *fieldnames* | 4.2.0 |
| *javamethods* | *methods* | 4.2.0 |
| *re\_read\_readline\_init\_file* | *readline\_re\_read\_init\_file* | 4.2.0 |
| *read\_readline\_init\_file* | *readline\_read\_init\_file* | 4.2.0 |
| *saving\_history* | *history\_save* | 4.2.0 |
| *allow\_noninteger\_range\_as\_index* | None | 4.4.0 |
| *bicubic* | *interp2* | 4.4.0 |
| *delaunay3* | *delaunay* | 4.4.0 |
| *do\_braindead\_shortcircuit\_evaluation* | None | 4.4.0 |
| *dump\_prefs* | None 4.4.0 |  |
| *find\_dir\_in\_path* | *dir\_in\_loadpath* | 4.4.0 |
| *finite* | *isfinite* | 4.4.0 |
| *fmod* | *rem* | 4.4.0 |
| *fnmatch* | *glob* or *regexp* | 4.4.0 |
| *gmap40* | None | 4.4.0 |
| *loadaudio* | *audioread* | 4.4.0 |
| *luinc* | *ichol* or *ilu* | 4.4.0 |
| *mouse\_wheel\_zoom* | *mousehweelzoom* property | 4.4.0 |
| *nfields* | *numfields* | 4.4.0 |
| *octave\_tmp\_file\_name* | *tempname* | 4.4.0 |
| *playaudio* | *audioplayer* | 4.4.0 |
| *saveaudio* | *audiowrite* | 4.4.0 |
| *setaudio* | None | 4.4.0 |
| *syl* | *sylvester* | 4.4.0 |
| *usage* | *print\_usage* | 4.4.0 |
| *bitmax* | *flintmax* | 5.1.0 |
| *mahalanobis* | *mahal* in Octave Forge statistics pkg | 5.1.0 |
| *md5sum* | *hash* | 5.1.0 |
| *octave\_config\_info* | *\_\_octave\_config\_info\_\_* | 5.1.0 |
| *onenormest* | *normest1* | 5.1.0 |
| *sleep* | *pause* | 5.1.0 |
| *usleep* | *pause* | 5.1.0 |
| *wavread* | *audioread* | 5.1.0 |
| *wavwrite* | *audiowrite* | 5.1.0 |

**附录D已知的故障原因**

本节描述影响Octave用户的已知问题。其中大多数不是Octave本身的bug——如果是的话，我们会修复它们。但对用户来说，结果可能就像一个bug的结果。

其中一些问题是由于其他软件中的错误，一些是由于添加太多工作而缺少的功能，还有一些是人们对什么是最好的意见不同的地方。

**D.1我们尚未修复的实际bug**

* 直接来自Fortran函数的输出不会通过寻呼机发送，并且可能与通过寻呼机发送的其他输出顺序不一致。避免这种情况的一种方法是，在调用将从Fortran函数内部产生输出的函数之前，强制刷新挂起的输出。要做到这一点，请使用命令

fflush (stdout)

另一个可能的解决方法是使用命令

page\_screen\_output (false);

关掉传呼机。

Octave发布了一个关于未来增强的想法列表。请参阅源代码发行版的顶级目录中的PROJECTS文件。

**D.2 bug报告**

bug报告在保证Octave的可靠性方面起着至关重要的作用。

当遇到问题时，首先要做的是看看是否已经知道了。参见已知的故障原因。如果不知道，那么应该报告这个问题。

报告错误可能会帮助您找到问题的解决方案，也可能不会。无论如何，bug报告的主要功能是通过使Octave的下一个版本更好地工作来帮助整个社区。Bug报告是您对Octave维护的贡献。

为了使错误报告发挥其作用，您必须包含使修复错误成为可能的信息。

**D.2.1你发现Bug了吗?**

如果你不确定你是否发现了一个bug，这里有一些指导方针:

* 如果Octave得到一个致命信号，不管输入是什么，那就是一个bug。可靠的解释器从不崩溃。
* 如果Octave对任何输入产生不正确的结果，那就是一个bug。
* 有些输出可能看起来不正确，但实际上是由于一个程序的行为未定义，这是偶然发生的，在另一个系统上给出了期望的结果。例如，由于不同系统处理浮点运算的方式不同，范围运算符可能产生不同的结果。
* 如果Octave为有效输入产生错误消息，那就是一个bug。
* 如果Octave没有为无效输入产生错误消息，那就是一个bug。然而，您应该注意到，您的“无效输入”的想法可能是我的“扩展”或“支持传统实践”的想法。
* 如果您是像Octave这样的程序的经验丰富的用户，您的改进建议无论如何都是受欢迎的。

**D.2.2在哪里报告bug**

要报告Octave中的错误，请向Octave错误跟踪器https://bugs.octave.org提交错误报告。

不要将bug报告发送到help-octave。Octave的大多数用户不希望收到错误报告。

**D.2.3如何报告bug**

向Octave bug跟踪器https://bugs.octave.org提交Octave的bug报告。

有效报告bug的基本原则是:报告所有事实。如果你不确定是陈述一个事实还是省略它，那就陈述吧!

人们经常忽略事实，因为他们认为他们知道问题的原因，他们认为一些细节无关紧要。因此，您可以假设在示例中使用的变量的名称无关紧要。嗯，可能不是，但谁也不能确定。也许这个bug是一个偶然的内存引用，它恰好从存储在内存中的名称的位置获取;也许，如果名称不同，该位置的内容会欺骗解释器，使其在存在错误的情况下执行正确的操作。谨慎起见，给出一个具体的、完整的例子。

请记住，错误报告的目的是使某人能够在不知道的情况下修复错误。在编写bug报告时，总是假设这个bug是未知的。

有时人们会给出一些粗略的事实，然后问:“这让你想起什么了吗?”这不能帮助我们修复错误。最好先发送一份完整的bug报告。

尽量使您的bug报告独立。如果我们需要您提供更多信息，您最好在您的回复中包括所有之前的信息，以及遗漏的信息。

为了让别人能够调查这个bug，你应该包括所有这些东西:

* 八度的版本。您可以通过注意Octave启动时打印的版本号或使用' -v '选项运行它来获得此信息。
* 将重现错误的完整输入文件。

单个语句可能不足以作为示例—错误可能依赖于最终发生错误的单个语句中缺少的其他细节。

* 您给Octave的命令参数，用于执行该示例并观察错误。为了保证你不会遗漏一些重要的东西，列出所有的选项。

如果我们试图猜测参数，我们可能会猜错，然后我们就不会遇到这个错误。

* 您正在使用的机器类型、操作系统名称和版本号。
* 安装解释器时给configure命令的命令行参数。
* 您对解释器源代码所做的任何修改的完整列表。

准确地描述这些变化——为它们显示上下文差异。

* 安装Octave标准程序的任何其他偏差的详细信息。
* 描述你观察到的你认为不正确的行为。例如，“解释器得到一个致命信号”，或者“第208行产生的输出不正确”。

当然，如果错误是解释器得到一个致命的信号，那么人们就不会错过它。但是，如果错误是不正确的输出，我们可能不会注意到，除非它是明显错误的。

即使你遇到的问题是一个致命的信号，你仍然应该明确地说出来。假设发生了一些奇怪的事情，例如，您的解释器副本不同步，或者您在系统上的C库中遇到了错误。你的副本可能会崩溃，而这里的副本不会。如果您说预期会发生崩溃，那么当这里的解释器没有崩溃时，我们就会知道没有发生错误。如果你不说预期崩溃，那么我们就不知道这个bug是否发生了。我们无法从我们的观察中得出任何结论。

通常观察到的症状是运行程序时的错误输出。不幸的是，除非程序简短，否则这些信息是不够的。如果您可以包括预期输出的解释，以及为什么实际输出是不正确的，这将非常有帮助。

* 如果您希望建议对Octave源进行更改，请将它们作为上下文差异发送。如果您甚至在Octave源中讨论某些内容，请根据上下文引用它，而不是通过行号，因为开发源中的行号可能与您的源中的行号不匹配。

以下是一些不必要的事情:

* 对bug的信封的描述。

遇到bug的人通常会花费大量时间调查对输入文件的哪些更改会使bug消失，哪些更改不会影响它。这些信息对于我们修复Octave中的错误通常是不必要的，但是如果你能找到一个更简单的例子来报告，而不是原来的那个，那就方便了。输出中的错误将更容易发现，在调试器下运行将花费更少的时间，等等。大多数Octave错误只涉及一个函数，因此简化示例的最直接方法是删除除发生错误的函数定义外的所有函数定义。

然而，简化并不重要;如果您不想这样做，那么无论如何都要报告bug并发送您使用的整个测试用例。

* bug的补丁。补丁可能很有帮助，但是如果您发现了错误，您应该报告它，即使您不能发送问题的修复程序。

**D.2.4为Octave发送补丁**

如果您想为Octave编写bug修复或改进，这将非常有帮助。当您发送更改时，请遵循这些指导方针，以避免给我们研究补丁造成额外的工作。

如果您不遵循这些指导方针，您的信息可能仍然有用，但使用它将需要额外的工作。在最好的情况下，维持八度是很多工作，除非你尽你最大的努力帮助我们，否则我们无法跟上。

* 发送一个解释，说明他们解决了什么问题，或者他们带来了什么改进。对于bug修复，只需包含bug报告的副本，并解释为什么更改修复了bug。
* 对于您认为已经修复的问题，始终包含适当的错误报告。在安装之前，我们需要说服自己这个更改是正确的。即使它是正确的，如果我们没有重现问题的方法，我们也可能很难判断它。
* 包括所有适当的注释，以帮助将来阅读源代码的人理解为什么需要进行此更改。
* 不要把出于不同原因所做的改变混在一起。单独发送。

如果您出于不同的原因进行了两个更改，那么我们可能不希望同时安装它们。我们可能只需要安装一个。

* 使用' diff -c '来设置差异。没有上下文的差异对我们来说很难可靠地安装。更重要的是，它们使我们很难研究差异，以决定是否要安装它们。统一的diff格式比无上下文的diffs格式更好，但不像' -c '格式那样容易阅读。

如果您有GNU diff，请使用' diff -cp '，它显示每个更改发生的函数的名称。

* 为您的更改编写更改日志条目。

阅读ChangeLog文件，看看要放入哪些类型的信息，并了解我们使用的样式。变更日志的目的是向人们展示在哪里可以找到更改的内容。所以你需要具体说明你改变了哪些功能;在大型函数中，指出在函数中的何处进行了更改通常很有帮助。

另一方面，一旦你告诉人们在哪里可以找到改变，你就不需要解释它的目的。因此，如果您添加了一个新函数，您只需说明它是新的即可。如果您觉得需要解释目的，那么很可能是需要的——但是如果将解释放在代码的注释中，解释将更加有用。

如果您希望您的名字出现在更改人的标题行中，请将标题行发送给我们。

**D.3如何获得八度的帮助**

邮件列表help@octave.org用于讨论与使用和安装Octave相关的问题。如果你想加入讨论，请发送一个简短的通知到help-request@octave.org。

请不要向列表本身发送添加或删除邮件列表或其他管理琐事的请求。

但是，如果您认为在Octave中或在安装过程中发现了错误，则应该向Octave错误跟踪器https://bugs.octave.org提交完整的错误报告。但在提交bug报告之前，请阅读https://www.octave.org/bugs.html了解如何提交有用的bug报告。

**D.4如何区分Octave和Matlab**

Octave和MATLAB非常相似，但处理Java略有不同。因此，可能有必要检测环境并使用适当的功能。可以使用以下函数来检测环境。由于是持久变量，可以重复调用它而不会对性能造成严重影响。

例子:

%%

%% Return: true if the environment is Octave.

%%

function retval = isOctave

persistent cacheval; % speeds up repeated calls

if isempty (cacheval)

cacheval = (exist ("OCTAVE\_VERSION", "builtin") > 0);

end

retval = cacheval;

end

**附录E安装Octave**

下面将描述在类unix系统上从源代码安装Octave的过程。在其他平台上进行构建将遵循类似的步骤。注意，此描述适用于Octave版本。从Mercurial存档构建开发源文件需要开发源文件本身中描述的其他步骤。

**E.1构建依赖**

Octave是一个相当大的程序，有许多构建依赖项。您可能能够找到作为系统一部分分发的依赖项的预打包版本，或者您可能必须自己构建部分或全部依赖项。

**E.1.1自动获取依赖项**

在某些系统上，您可以自动获得许多Octave的构建依赖项。执行此操作的命令因系统而异。类似地，预编译包的名称因系统而异，并不总是与构建工具和外部包中列出的名称完全匹配。

您通常需要外部依赖项的开发版本，以便获得用于构建软件的库和头文件，而不仅仅是用于运行已编译的程序。这些包的名称通常以-dev或-devel后缀结尾。

在使用apt-get的系统上(Debian, Ubuntu等)，您可以使用类似的命令安装大多数工具和外部包

apt-get build-dep octave

具体的包名可以是octave3.2或octave3.4。所需的工具集和外部依赖项不会经常更改，因此版本是否完全匹配并不重要，但您应该使用可用的最新版本。

在使用yum (Fedora、Red Hat等)的系统上，您可以使用类似的命令安装大多数工具和外部包

yum-builddep octave

yum- builddeep实用程序是yum-utils包的一部分。

对于任何一种类型的系统，包名都可以包含一个版本号。所需的工具集和外部依赖项不会经常更改，因此版本是否与您正在安装的版本完全匹配并不重要，但您应该使用可用的最新版本。

**E.1.2构建工具**

需要准备的工具如下:

c++、C和Fortran编译器

Octave源代码主要是用c++编写的，但也有一些部分是用C和Fortran编写的。Octave源是可移植的。最新版本的GNU编译器集合(GCC)应该可以工作(https://gcc.gnu.org)。如果您使用GCC，则应该避免混合版本。例如，请确保您没有使用过时的g77 Fortran编译器和最新版本的gcc和g++。

GNU Make

用于构建软件的工具(https://www.gnu.org/software/make)。Octave的构建系统需要GNU Make。其他版本的Make将无法工作。幸运的是，GNU Make具有高度的可移植性和易于安装。

AWK、sed和其他Unix实用程序

构建Octave需要基本的Unix系统实用程序。所有这些都可以在任何现代Unix系统上使用，也可以在Windows上使用Cygwin或MinGW和MSYS。

此外，可能还需要以下工具:

Bison

解析器生成器(https://www.gnu.org/software/bison)。如果要修改oct-parse，就需要Bison。Yy源文件，或者删除从中生成的文件。

Flex

词法分析器(https://www.gnu.org/software/flex)。如果要修改lex，就需要Flex。Ll源文件，或者删除从中生成的文件。

Autoconf

软件配置包(https://www.gnu.org/software/autoconf)。如果您修改Octave的配置，则需要Autoconf。Ac文件或所需的其他文件。

Automake

生成Makefile的包(https://www.gnu.org/software/automake)。如果您修改Octave的Makefile，则需要Automake。文件或它们所依赖的其他文件。

Libtool

用于构建软件库的软件包(https://www.gnu.org/software/libtool)。Automake需要Libtool。

gperf

完美的哈希函数生成器(https://www.gnu.org/software/gperf)。如果你修改八度，你将需要gperf。Gperf文件，或者删除由它生成的文件。

Texinfo

用于生成在线和打印文档的软件包(https://www.gnu.org/software/texinfo)。您将需要Texinfo来构建Octave的文档，或者如果您修改文档源文件或任何Octave函数的文档字符串。

**E.1.3外部包装**

需要以下外部包装:

BLAS

基本线性代数子程序库。为了获得最佳性能，建议使用加速的BLAS库，如OpenBLAS (https://www.openblas.net/)或ATLAS (http://math-atlas.sourceforge.net)。参考实现(http://www.netlib.org/blas)很慢，并且在极端情况输入中存在某些错误。

LAPACK

线性代数包(http://www.netlib.org/lapack)。

PCRE

Perl兼容正则表达式库(https://www.pcre.org)。

以下外部包是可选的，但强烈推荐:

GNU Readline

命令行编辑库(https://www.gnu.org/s/readline)。

如果您希望在没有安装GNU readline的情况下构建Octave，则必须在运行configure脚本时使用——disable-readline选项。

以下外部软件包为可选软件包。Octave可以在没有它们的情况下构建，但可能会缺少某些功能:

ARPACK

解决大规模特征值问题的库(https://forge.scilab.org/index.php/p/arpack-ng)。需要ARPACK来提供eigs和svds函数。

cURL

用URL语法传输数据的库(https://curl.haxx.se)。cURL需要提供urlread和urlwrite函数以及ftp类。

FFTW3

用于计算离散傅里叶变换的库(http://www.fftw.org)。FFTW3用于为计算离散傅里叶变换(fft, ifft, fft2等)的函数提供更好的性能。

FLTK

可移植GUI工具包(http://www.fltk.org)。FLTK可用于为Octave的基于opengl的图形功能提供窗口。

fontconfig

用于配置和定制字体访问的库(https://www.freedesktop.org/wiki/Software/fontconfig)。Fontconfig用于管理Octave基于opengl的图形功能的字体。

FreeType的

便携式字体引擎(https://www.freetype.org)。FreeType用于为Octave的基于opengl的图形函数执行字体渲染。

GLPK

GNU线性编程工具包(https://www.gnu.org/software/glpk)。函数glpk需要GPLK。

gl2ps

OpenGL到PostScript打印库(https://www.geuz.org/gl2ps/)。当使用基于opengl的图形工具包(目前是FLTK或Qt)时，打印需要gl2ps。

gnuplot

交互式图形程序(http://www.gnuplot.info)。gnuplot可以用作Octave的图形渲染器;在Octave 4.0之前，gnuplot是默认的图形渲染器。

GraphicsMagick + +

图像处理库(http://www.graphicsmagick.org)。graphicsmagick++用于提供imread和imwrite函数。

HDF5

用于操作可移植数据文件的库(https://www.hdfgroup.org/HDF5)。HDF5需要用于Octave的load和save命令来读取和写入HDF数据文件。

Java开发工具包

Java编程语言编译器和库。建议使用OpenJDK免费软件实现(http://openjdk.java.net/)，但其他JDK实现也可以使用。Java需要能够从Octave内部调用Java函数。

OpenGL

用于便携式2d和3d图形的API (https://www.opengl.org)。OpenGL实现可以用来为Octave的图形功能提供渲染器。Octave基于opengl的图形函数通常优于基于gnuplot的图形函数，因为绘图数据可以直接呈现，而不是将数据和命令发送到gnuplot进行解释和呈现。从Octave 4.0开始，默认的图形渲染器(“qt”)是基于opengl的。

PortAudio

PortAudio (http://www.portaudio.com/)提供了一个非常简单的API，用于使用简单的回调函数或阻塞读/写接口来录制和/或播放声音。音频处理功能audioplayer、audiorecorder和audiodevinfo都需要它。

Qhull

计算几何库(http://www.qhull.org)。Qhull需要提供convhull、convhulln、delaunay、delaunayn、voronoi和voronoin等功能。

QRUPDATE

QR分解更新库(https://sourceforge.net/projects/qrupdate)。QRUPDATE用于为qrdelete、qrinsert、qrshift和QRUPDATE函数提供改进的性能。

QScintilla

源代码高亮笔和操作器;Scintilla的Qt移植(http://www.riverbankcomputing.co.uk/software/qscintilla)。QScintilla用于GUI中的语法高亮显示和代码补全。

Qt

GUI和实用程序库(https://www.qt.io)。Qt是构建GUI所必需的。它是一个大型框架，但所需的组件只有GUI、核心和网络模块。从Octave 4.0开始，默认的图形渲染器(“qt”)是基于qt的，它是基于opengl的。

RapidJSON

一个用于c++的快速JSON解析器/生成器，具有SAX/DOM风格的API (https://rapidjson.org/)。使用jsondecode和jsonencode函数读取或写入JSON文件需要RapidJSON。

SuiteSparse

稀疏矩阵分解库(http://faculty.cse.tamu.edu/davis/suitesparse.html)。为了提供稀疏系统的稀疏矩阵分解和线性方程的解，需要使用SuiteSparse。

SUNDIALS

常微分方程(ODE)求解器ode15i和ode15s需要非线性和微分/代数方程求解器套件(https://computation.llnl.gov/projects/sundials)。

zlib

数据压缩库(https://zlib.net)。zlib库是Octave的load和save命令处理压缩数据(包括MATLAB v5 MAT文件)所必需的。

**E.2运行Configure和Make命令**

* 运行shell脚本configure。这将确定您的系统具有(或不具有)的特性，并从每个名为Makefile.in的文件中创建一个名为Makefile的文件。

要获取配置选项的完整列表，请运行configure——help。以下是构建Octave时最常用的配置选项的摘要:

--help

打印configure脚本识别的选项摘要。

--prefix=prefix

在prefix下面的子目录中安装Octave。prefix的默认值为“/usr/local”。

--srcdir=dir

在目录dir中查找Octave源文件。

--disable-64

禁用使用64位整数索引数组，改为使用32位整数。在32位指针的系统上，这个选项总是被禁用的。如果configure脚本确定您的BLAS库使用32位整数，那么使用以下库的操作仅限于尺寸小于2^{31}元素的数组:

* BLAS
* LAPACK
* QRUPDATE
* SuiteSparse
* ARPACK

此外，以下库在内部使用int，因此最大问题大小总是有限的:

* GLPK
* Qhull

请参阅编译带有64位索引的Octave，了解有关构建具有更完整的大型数组支持的Octave的更多详细信息。

--enable-address-sanitizer-flags

启用编译器选项-fsanitize=address和-fomit-frame-pointer进行内存访问检查。这个选项主要用于调试Octave。使用此选项构建Octave会对性能产生负面影响，不建议一般使用。它还可能干扰GUI的正常运行。

--disable-docs

禁止构建所有形式的文档(Info, PDF, HTML)。默认是构建文档，但是您的系统需要正常运行的Texinfo和TeX安装才能成功。

--enable-float-truncate

此选项允许截断计算中的中间浮点结果。它只对某些平台是必需的。

--enable-readline

使用readline库在终端环境中提供命令行编辑功能。该选项在默认情况下是打开的。

--enable-shared

创建共享库(这是默认设置)。如果您计划使用动态加载特性，则可能需要使用此选项。它将使你的。oct文件更小，在某些系统上，为了使用动态链接的函数，可能需要构建共享库。

如果您的系统还没有libstdc++的共享版本，您可能还想构建一个。

--with-blas=<lib>

默认情况下，configure会查找系统上最好的BLAS矩阵库，包括优化的实现，如免费的ATLAS 3.0，以及供应商调优的库。(使用优化的BLAS通常会使矩阵运算速度提高几倍。)使用此选项指定Octave应该使用的特定BLAS库。

--with-lapack=<lib>

默认情况下，configure会查找系统上最好的LAPACK矩阵库，包括优化的实现，比如免费的ATLAS 3.0，以及供应商调优的库。(使用经过优化的LAPACK通常会使矩阵运算速度提高几倍。)使用此选项指定Octave应该使用的特定LAPACK库。

--with-magick=<lib>

选择用于映像I/O的magick++库。对于许多发行版，可能的值是" graphicsmagick++ "(默认)或" imagemagick++ "。

--with-sepchar=<char>

使用<char>作为路径分隔字符。当在非unix系统上运行Octave时，此选项可以提供帮助。

--without-amd

不要使用AMD，禁用一些稀疏矩阵功能。

--without-camd

不要使用CAMD，禁用一些稀疏矩阵功能。

--without-colamd

不要使用COLAMD，禁用一些稀疏矩阵功能。

--without-ccolamd

不要使用CCOLAMD，禁用一些稀疏矩阵功能。

--without-cholmod

不要使用CHOLMOD，禁用一些稀疏矩阵功能。

--without-curl

不要使用cURL库，禁用ftp对象、urlread和urlwrite函数。

--without-cxsparse

不要使用CXSPARSE，禁用一些稀疏矩阵功能。

--without-fftw3

使用包含的FFTPACK库来计算快速傅里叶变换，而不是FFTW3库。

--without-fftw3f

使用包含的FFTPACK库来计算快速傅里叶变换，而不是在单精度(浮点)值上操作时使用FFTW3库。

--without-glpk

不要使用GLPK库进行线性规划。

--without-hdf5

不要使用HDF5库，禁用HDF5文件的读写。

--without-opengl

不要使用OpenGL，禁用本机图形工具包进行绘图。您需要安装gnuplot才能制作图表。

--without-qhull\_r

不要使用(可重入的)Qhull，禁用delaunay、convhull和相关功能。

--without-qrupdate

不要使用QRUPDATE，禁用QR和Cholesky更新功能。

--without-umfpack

不要使用UMFPACK，禁用一些稀疏矩阵功能。

--without-z

不要使用zlib库，禁用数据文件压缩和对最新MAT文件格式的支持。

--without-framework-carbon

即使configure测试成功，也不要使用框架的Carbon头、库或特定的源代码(默认情况下是使用可用的Carbon框架)。这是Mac系统的特定平台配置选项。

--without-framework-opengl

即使configure测试成功，也不要使用框架OpenGL头文件、库或特定的源代码。如果给出此选项，则测试标准系统位置中的OpenGL头文件和库(默认值是——with-frame - OpenGL)。这是Mac系统的特定平台配置选项。

有关configure使用的命令行选项的更多一般信息，请参阅文件INSTALL。该文件还包含在源代码所在目录以外的目录中进行编译的指令。

* 运行make。

您将需要一个最新版本的GNU Make，因为Octave依赖于某些在所有Make版本中通常不可用的特性。修改Octave的make文件来与其他make程序一起工作可能不值得你花时间;相反，我们只是建议安装GNU Make。

目前在Octave中有三种绘图选项:外部程序gnuplot，内部图形引擎使用OpenGL加上FLTK或Qt小部件。Gnuplot是一个命令驱动的交互式函数绘图程序。

要编译Octave，您将需要最新版本的g++或其他ANSI c++编译器。此外，您还需要一个Fortran 77编译器或f2c。如果你使用f2c，你将需要一个像for77这样的脚本，通过将f2c和C编译器结合在一个脚本中，它就像一个普通的Fortran编译器一样工作。

如果计划修改解析器，还需要GNU bison和flex。如果您修改文档，您将需要GNU Texinfo。

GNU Make, gcc(和libstdc++)， gnuplot, bison, flex和Texinfo都可以从许多匿名ftp存档中获得。主站点是ftp://ftp.gnu.org，但它经常非常繁忙。在ftp://ftp.gnu.org/pub/gnu/GNUinfo/FTP上可以通过匿名ftp获得与ftp://ftp.gnu.org上的软件镜像的网站列表。

Octave需要大约1.4 GB的磁盘存储空间来解包并从源代码编译(如果不使用调试符号编译，则需要显著减少400 MB)。要在没有调试符号的情况下进行编译，请尝试执行命令

make CFLAGS=-O CXXFLAGS=-O LDFLAGS=

而不只是make

* 如果在编译Octave时遇到错误，请首先参阅安装问题以获得已知问题的列表，以及是否有解决问题的变通方法或解决方案。如果没有，请参阅已知的问题原因，了解如何报告错误。
* 成功编译Octave后，运行make install。

这将在目标目录中安装Octave、它的库和它的文档的副本。作为分布式版本，Octave安装在以下目录中。在下面的表中，prefix默认为/usr/local, version表示解释器的当前版本号，arch是安装Octave的计算机类型(例如，' i586-unknown-gnu ')。

prefix/bin

人们想要直接运行的Octave和其他二进制文件。

prefix/lib/octave-version

像liboctave这样的库。A和libotinterp。

prefix/include/octave-version/octave

包括与Octave一起分发的文件。

prefix/share

与体系结构无关的数据文件。

prefix/share/man/man1

描述Octave的unix风格手册页。

prefix/share/info

描述八度的信息文件。

prefix/share/octave/version/m

随Octave分发的函数文件。这包括Octave版本，因此可以同时安装多个版本的Octave。

prefix/libexec/octave/version/exec/arch

可执行文件由Octave而不是用户运行。

prefix/lib/octave/version/oct/arch

将被动态加载的对象文件。

prefix/share/octave/version/imagelib

与Octave一起分发的图像文件。

**E.3编译带有64位索引的Octave**

注意:以下内容只适用于64位指针的系统。用——enable-64配置Octave不能神奇地使32位系统拥有64位地址空间。

在64位系统上，Octave默认使用64位整数来索引数组。如果configure脚本确定您的BLAS库使用32位整数，那么使用以下库的操作仅限于尺寸小于2^{31}元素的数组:

* BLAS
* LAPACK
* QRUPDATE
* SuiteSparse
* SUNDIALS IDA
* ARPACK

此外，以下库在内部使用int，因此最大问题大小总是有限的:

* GLPK
* Qhull

除了GLPK和Qhull，这些库也可以配置为使用64位整数，但是大多数系统不提供以这种方式构建的包。如果您希望尝试使用大型数组，下面的信息可能会有所帮助。

要确定Octave使用的BLAS库的整数大小，可以执行以下代码:

clear all;

N = 2^31;

## The following line requires about 8 GB of RAM!

a = b = ones (N, 1, "single");

c = a' \* b

如果BLAS库使用32位整数，则会抛出错误:

error: integer dimension or index out of range for Fortran

INTEGER type

否则，如果BLAS库使用64位整数，结果是:

c = 2^31 = 2147483648

注意，如果a和b不是由a = b = ....分配的话，上面的测试用例通常需要两倍的内存进一步注意，数据类型single的精度约为23二进制位。在这个特定的示例中，没有发生舍入错误。

通常，最好让所有这些库都在支持32位索引的版本中，或者所有这些库都必须支持64位索引。将64位索引库与32位索引库混合使用可能会导致不可预测的行为，包括程序崩溃和可能的数据丢失。

以下说明是在x86\_64 Debian系统上使用Octave的开发版本和GCC 4.3.4进行测试的，现在可能已经过时了。请在Octave错误跟踪器上报告任何问题或更正。

下面列出的版本是用于测试的版本。如果这些软件包的新版本可用，您应该尝试使用它们，尽管可能存在一些差异。

所有库和头文件都将安装在$prefix64的子目录中(您必须选择该目录的位置)。

* BLAS和LAPACK (http://www.netlib.org/lapack)

这两个库的参考版本都包含在http://www.netlib.org/的参考LAPACK 3.2.1发行版中。

* 复制文件make.inc.example并将其命名为make.inc。选项- default-integer-8和-fPIC(64位CPU)必须添加到变量OPTS和NOOPT中。
* 一旦你编译了这个库，确保你使用它来编译Suite Sparse和Octave。下面我们假设您将LAPACK库安装为$prefix64/lib/liblapack.a。
* QRUPDATE (https://sourceforge.net/projects/qrupdate)

在Makeconf文件中:

* 给fflag增加- default-integer-8。
* 如果64位感知的BLAS和LAPACK库位于非标准位置，则根据需要调整BLAS和LAPACK变量。
* 将PREFIX设置为安装树的顶级目录。
* 执行make solib命令创建共享库。
* 运行make install安装库。
* SuiteSparse (http://faculty.cse.tamu.edu/davis/suitesparse.html)

通过以下选项来为BLAS库调用启用64位整数。在64位Windows系统上，使用-DLONGBLAS="long long"代替。

CFLAGS='-DLONGBLAS=long'

CXXFLAGS='-DLONGBLAS=long'

SuiteSparse生成文件不会生成共享库。在某些系统上，您可以通过执行以下简单操作来生成它们

top=$(pwd)

for f in \*.a; do

mkdir tmp

cd tmp

ar vx ../$f

gcc -shared -o ../${f%%.a}.so \*.o

cd $top

rm -rf tmp

done

其他系统可能需要不同的解决方案。

* SUNDIALS IDA (https://computing.llnl.gov/projects/sundials/ida)

当使用cmake配置时，添加标志-DSUNDIALS\_INDEX\_SIZE=64。

* ARPACK (https://forge.scilab.org/index.php/p/arpack-ng/)
* 运行configure时，为fflag添加- default-integer-8。
* 运行make来构建库。
* 运行make install安装库。
* ATLAS代替参考BLAS和LAPACK

欢迎就如何编制地图集提出建议。

* GLPK
* Qhull (http://www.qhull.org)

GLPK和Qhull都在内部使用int，因此最大问题大小可能受到限制。

* Octave

Octave的64位索引支持通过配置选项--enable-64激活。

./configure \

LD\_LIBRARY\_PATH="$prefix64/lib" \

CPPFLAGS="-I$prefix64/include" LDFLAGS="-L$prefix64/lib" \

--enable-64

你必须确保除了liboctave/external/ranlib目录下的所有Fortran源代码都被编译成整数是8字节宽的。如果您使用的是gfortran，那么配置脚本应该自动将Makefile变量F77\_INTEGER\_8\_FLAG设置为- default-integer-8。如果使用其他编译器，则必须自己设置该变量。你不应该在fflag中设置这个标志，否则liboctave/external/ranlib中的文件会被错误编译。

* 其他依赖项

对于以下依赖项可能不需要做什么特别的事情。如果你发现某些事情确实需要做，请提交一个bug报告。

-pcre

-zlib

-hdf5

-fftw3

-cURL

-GraphicsMagick++

-OpenGL

-freetype

-fontconfig

-fltk

**E.4安装问题**

本节包含在安装Octave期间可能出现的问题列表(以及一些并不真正意味着有问题的明显问题)。

* 在一些SCO系统上，如果在config.h中定义了HAVE\_TERMIOS\_H, info将无法编译。简单地从info/config.h中删除定义应该允许它编译。
* 如果configure找到dlopen、dlsym、dlclose和dlerror，但没有找到头文件dlfcn.h，则需要找到头文件的源文件并将其安装在目录usr/include中。这是Slackware 3.1的一个问题。对于Linux/GNU系统，dlfcn.h的源代码在ldso包中。
* 构建.oct文件不起作用。

您可能应该有一个libstdc++的共享版本。在HP-PA架构上构建libstdc++ 2.7.2版本的共享版本需要一个补丁。您可以在ftp://ftp.cygnus.com/pub/g++/libg++-2.7.2-hppa-gcc-fix上找到该补丁。

* 在一些DEC alpha系统上，libdxml库可能存在问题，导致Octave调用的线性代数例程出现浮点错误和/或分割错误。如果遇到这样的问题，那么应该修改配置脚本，使SPECIAL\_MATH\_LIB不被设置为-ldxml。
* 在FreeBSD系统上，Octave在初始化一些内部常量时可能会挂起。解决方案似乎是使用

options GPL\_MATH\_EMULATE

而不是

options MATH\_EMULATE

在内核配置文件中(通常在目录/sys/i386/conf中找到)。进行此更改后，需要重新构建内核、安装内核并重新启动。

* 如果遇到类似的错误

passing `void (\*)()' as argument 2 of

`octave\_set\_signal\_handler(int, void (\*)(int))'

或者

warning: ANSI C++ prohibits conversion from `(int)'

to `(…)'

编译sighandlers时。Cc，您可能需要编辑GCC include子目录中的一些文件，以便为那里的函数添加适当的原型。例如，Ultrix 4.2需要对signal.h文件中的signal函数和SIG\_IGN宏进行适当的声明。

在某些系统上，SIG\_IGN宏被定义为如下所示:

#define SIG\_IGN (void (\*)())1

其实应该是这样的:

#define SIG\_IGN (void (\*)(int))1

来匹配信号函数的原型声明。也应该对SIG\_DFL和SIG\_ERR符号进行此更改。可能还需要更改sys/signal.h中的定义。

当gcc安装其修改后的头文件集时，gcc fixincludes和fixproto脚本可能会修复这些问题，但我认为这还没有完成。

您不应该更改/usr/include中的文件。您可以通过运行命令找到gcc include目录树

gcc -print-libgcc-file-name

gcc包含文件的目录通常从包含libgcc.a文件的目录开始。

* 一些Fortran子程序可能无法用旧版本的Sun Fortran编译器进行编译。如果你得到这样的错误

zgemm.f:

zgemm:

warning: unexpected parent of complex expression subtree

zgemm.f, line 245: warning: unexpected parent of complex

expression subtree

warning: unexpected parent of complex expression subtree

zgemm.f, line 304: warning: unexpected parent of complex

expression subtree

warning: unexpected parent of complex expression subtree

zgemm.f, line 327: warning: unexpected parent of complex

expression subtree

pcc\_binval: missing IR\_CONV in complex op

make[2]: \*\*\* [zgemm.o] Error 1

当编译libocave /external子目录中的Fortran子程序时，您应该升级编译器或尝试在关闭优化的情况下进行编译。

* 在NeXT系统上，如果你得到这样的错误:

/usr/tmp/cc007458.s:unknown:Undefined local

symbol LBB7656

/usr/tmp/cc007458.s:unknown:Undefined local

symbol LBE7656

编译Array时。cc和矩阵。Cc，尝试不使用-g重新编译这些文件。

* 有些人报告说，对system()和寻呼机的调用在SunOS系统上不起作用。这显然是由于在编译lib++时将G\_HAVE\_SYS\_WAIT定义为0而不是1。
* 在使用参考BLAS库的系统上，下面的矩阵乘向量乘法错误地处理NaN \* 0形式的NaN值。

[NaN, 1; 0, 0] \* [0; 1]

⇒

[ 1

0 ]

correct result ⇒

[ NaN

0 ]

安装不同的BLAS库(如OpenBLAS或ATLAS)来纠正此问题。

* 在NeXT系统上，链接到libsys\_s。A可能无法解析以下功能

\_tcgetattr

\_tcsetattr

\_tcflow

它们是libposix.a的一部分。不幸的是，将Octave与-posix连接会导致以下未定义的符号。

.destructors\_used

.constructors\_used

\_objc\_msgSend

\_NXGetDefaultValue

\_NXRegisterDefaults

.objc\_class\_name\_NXStringTable

.objc\_class\_name\_NXBundle

解决这个问题的一个方法是提取术语。O来自libposix。a，把它放在Octave的src目录中，并把它添加到makefile中链接到一起的文件列表中。欢迎提出更好的方法来解决这个问题!

* 如果Octave因浮点异常而立即崩溃，则很可能是它无法初始化infinity和NaN的IEEE浮点值。

如果您的系统确实支持IEEE算法，您应该能够通过修改文件lo-ieee中的octave\_ieee\_init函数来解决此问题。cc来正确初始化Octave的内部infinity和NaN变量。

如果您的系统不支持IEEE算法，但Octave的配置脚本错误地确定它支持，您可以通过编辑文件config.h来解决问题，不定义HAVE\_ISINF, HAVE\_FINITE和HAVE\_ISNAN。

在任何情况下，请将此报告为错误，因为可能会修改Octave的配置脚本以自动确定要做的正确事情。

* 如果Octave无法找到头文件，因为它安装在编译器通常不搜索的位置，您可以通过指定(例如)CPPFLAGS=-I/some/nonstandard/directory作为配置参数，将该目录添加到包含搜索路径中。可以以这种方式指定的其他变量有CFLAGS、CXXFLAGS、fflag和LDFLAGS。将它们作为选项传递给configure脚本也会在配置中记录它们。状态文件。默认情况下，CPPFLAGS和LDFLAGS为空，CFLAGS和CXXFLAGS设置为-g -O2, fflag设置为-O。

**附录F语法和解析器**

本附录最终将包含对Octave语言的半正式描述。

类别:

Utility Functions | Variable Declaration | Function Definition | Control Statements | Iterating Structures | Classdef Structures | Execution Environment

按字母顺序排列的关键字列表:

\_\_FILE\_\_ | \_\_LINE\_\_ | break | case | catch | classdef | continue | do | else | elseif | end | end\_try\_catch | end\_unwind\_protect | endclassdef | endenumeration | endevents | endfor | endfunction | endif | endmethods | endparfor | endproperties | endswitch | endwhile | endenumeration | events | for | function | global | if | methods | otherwise | parfor | persistent | properties | return | switch | try | until | unwind\_protect | unwind\_protect\_cleanup | while

效用函数:

函数iskeyword可用于快速检查某个标识符是否被Octave保留。

: iskeyword ()

: iskeyword (name)

如果name是一个Octave关键字则返回true。

如果省略name，则返回关键字列表。

参见:isvarname, exist。

变量声明：

: global var

声明具有全局作用域的变量。

global x;

if (isempty (x))

x = 1;

endif

参见:persistent。

: persistent var

将变量声明为持久变量。

在函数中声明为持久的变量将在对同一函数的后续调用之间将其内容保留在内存中。持久变量和全局变量的区别在于，持久变量在特定函数的作用域中是局部的，在其他地方是不可见的。

参见:global。

函数定义:

: function outputs = function\_name (input, …)

: function function\_name (input, …)

: function outputs = function\_name

以function\_name名称开始函数体，以输出作为结果，以输入作为参数。

稍后可以使用该语法在Octave中调用该函数

[output1, output2，…]= function\_name (input1, input2，…)

参见:return。

: endfunction

标记函数的结束。

参见:function。

: end

数组的最后一个元素或任何for、parfor、if、do、while、function、switch、try或unwind\_protect块的末尾。

作为数组的索引，魔术索引“end”指的是索引操作中的最后一个有效条目。

例子:

x = [ 1 2 3; 4 5 6 ];

x(1,end)

⇒ 3

x(end,1)

⇒ 4

x(end,end)

⇒ 6

编程提示:

end关键字不能在subsref、subsassign或substruct中用于手动索引操作。

对于自定义类，要启用在索引表达式中使用end，必须用函数定义重载它，例如:

function last\_index = end (obj, end\_dim, ndim\_obj)

if (end\_dim == ndim\_obj)

last\_index = prod (size (obj)(end\_dim:ndim\_obj));

else

last\_index = size (obj, end\_dim);

endif

endfunction

有关更多信息，请参阅面向对象编程。

参见:for, parfor, if, do, while, function, switch, try, unwind\_protect。

: return

立即将执行控制权从函数或脚本返回给调用代码。

Return用于立即停止执行代码并退出m文件，而不是继续执行直到到达函数或脚本的末尾。

如果函数或脚本是直接调用的，而不是调用m文件中的代码，那么Octave将返回到命令行。

参见:function。

控制叙述：

: if (cond) … endif

: if (cond) … else … endif

: if (cond) … elseif (cond) … endif

: if (cond) … elseif (cond) … else … endif

开始if块。

如果条件cond不为空且所有值都非零，则该条件cond为真。

x = 1;

if (x == 1)

disp ("one");

elseif (x == 2)

disp ("two");

else

disp ("not one or two");

endif

参见: switch。

: else

if块的替代操作。

看一个例子。

参见:if。

: elseif (cond)

if块的备用条件测试。

如果条件cond不为空且所有值都非零，则该条件cond为真。

看一个例子。

参见:if。

: endif

标记if块的结束。

看一个例子。

参见:if。

: switch statement

开始一个开关块。

yesno = "yes";

switch (yesno)

case {"Yes" "yes" "YES" "y" "Y"}

value = 1;

case {"No" "no" "NO" "n" "N"}

value = 0;

otherwise

error ("invalid value");

endswitch

参见:if, case, else。

: case value

: case {value, …}

switch块中的case语句。

八度用例是排他性的，不会像c语言用例那样失败。一个switch语句必须至少有一个case。请参见switch。

参见: switch。

: otherwise

switch块中的默认语句，当没有其他case语句与输入匹配时执行。

参见:switch, case。

: endswitch

标记开关块的末端。

请参见switch。

参见: switch。

: try

开始一个试接块。

如果在try块中发生错误，则会运行catch代码，并在catch块之后继续执行(尽管通常建议在清理完成后使用lasterr函数重新抛出错误)。

参见:catch, unwind\_protect。

: catch

: catch value

开始try-catch块的清理部分。

参见:try。

: end\_try\_catch

标记一个试接块的结束。

参见:try, catch。

: unwind\_protect

开始一个unwind\_protect块。

如果错误发生在unwind\_protect块的第一部分，则在抛出错误之前执行unwind\_protect\_cleanup块中的命令。如果没有抛出错误，则仍然执行unwind\_protect\_cleanup块。换句话说，无论unwind\_protect块的成功或失败，unwind\_protect\_cleanup代码都保证执行。

参见:unwind\_protect\_cleanup, try。

: unwind\_protect\_cleanup

开始unwind\_protect块的清理部分。

参见:unwind\_protect。

: end\_unwind\_protect

标记unwind\_protect块的结束。

参见:unwind\_protect。

迭代结构:

: for i = range

开始for循环。

for i = 1:10

i

endfor

参见:parfor, do, while。

: endfor

标记for循环的结束。

有关示例，请参见

参见:for。

: while (cond)

开始一个while循环。

如果条件cond不为空且所有值都非零，则该条件cond为真。

i = 0;

while (i < 10)

i++

endwhile

参见:do, endwhile, for, until。

: endwhile

标记while循环的结束。

参见while的例子。

参见:do, while。

: do

开始do-until循环。

这与while循环的不同之处在于循环体至少执行一次。

i = 0;

do

i++

until (i == 10)

参见:for, until, while。

: until (cond)

结束do-until循环。

如果条件cond不为空且所有值都非零，则该条件cond为真。

有关示例，请参见do。

参见:do。

: parfor i = range

: parfor (i = range, maxproc)

开始一个可以并行执行的for循环。

parfor循环的语法与for循环相同。如果您的Octave会话启用了并行处理池，则parfor循环的迭代将跨池的工作器并行执行。否则，parfor的行为将与for完全相同。

在并行模式下操作时，不能保证parfor循环的迭代顺序发生，并且对于可以在循环体内执行的数据访问操作还有其他限制。

警告:并行处理池目前未在Octave中实现;Parfor当前的行为与普通的for循环完全相同。

parfor i = 1:10

i

endparfor

参见:for, do, while。

: endparfor

标记parfor循环的结束。

参见parfor的示例。

参见:parfor。

: break

退出最内层的do、while或for循环。

参见:do, while, for, parfor, continue。

: continue

跳到最内层的do、while或for循环的末尾。

参见:break, do, while, for, parfor。

Classdef结构:

: classdef

开始一个classdef块。

参见: properties, methods, events, enumeration。

: endclassdef

标记类定义的结束。

参见:classdef。

: properties

在类定义中标记属性块的开头。注意，函数“properties”是一个列出classdef类或对象的属性的函数。

参见:endproperties。

: endproperties

在类定义中标记属性块的结束。

参见: properties。

: methods

在类定义中标记方法块的开头。注意，函数"methods"是一个列出类或对象的方法的函数。

参见:endmethods。

: endmethods

在类定义中标记方法块的结束。

参见: methods。

: events

在类定义中开始一个事件块。

: endevents

在类定义中标记事件块的结束。

参见: events。

: enumeration

在类定义中开始枚举块。

: endenumeration

在类定义中标记枚举块的结束。

参见: enumeration。

执行环境:

: \_\_FILE\_\_

当词法分析器识别"\_\_FILE\_\_"关键字时，它返回一个字符数组，其中包含正在执行的文件的全名和路径。如果从命令行调用"\_\_FILE\_\_"将返回stdin。

参见:\_\_LINE\_\_。

: \_\_LINE\_\_

当词法分析器识别出"\_\_LINE\_\_"关键字时，它返回一个数字值，其中包含正在执行的函数或文件的当前输入行号。如果从命令行调用"\_\_LINE\_\_"将返回1。

参见:\_\_FILE\_\_。

**F.2解析器**

解析器有许多影响其内部操作的变量。这些变量通常与它们影响的代码一起记录在手册中。

此外，还有三个非特定的解析器自定义函数。Add\_input\_event\_hook可用于调度用户函数进行定期评估。Remove\_input\_event\_hook将阻止用户函数定期求值。

: id = add\_input\_event\_hook (fcn)

: id = add\_input\_event\_hook (fcn, data)

将命名函数或函数句柄fcn添加到函数列表中，以便在Octave等待输入时定期调用。

函数应该有这样的形式

fcn (data)

如果省略data, Octave调用该函数时不带任何参数。

返回的标识符可用于从输入钩子函数列表中删除函数句柄。

参见:remove\_input\_event\_hook。

: remove\_input\_event\_hook (name)

: remove\_input\_event\_hook (fcn\_id)

当Octave等待输入时，从要定期调用的函数列表中删除带有给定标识符的命名函数或函数句柄。

参见:add\_input\_event\_hook。

最后，当解析器无法识别输入标记时，它调用一个特定的函数来处理这个问题。默认情况下，这是内部函数"\_\_unimplemented\_\_"，它对MATLAB函数可能的Octave替代品提出建议。

: val = missing\_function\_hook ()

: old\_val = missing\_function\_hook (new\_val)

: old\_val = missing\_function\_hook (new\_val, "local")

查询或设置指定要调用的函数的内部变量，以便在引用未知标识符时提供额外信息。

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:missing\_component\_hook。

**Appendix G GNU GENERAL PUBLIC LICENSE**

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <https://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this

license document, but changing it is not allowed.

**Preamble**

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers’ and authors’ protection, the GPL clearly explains that there is no warranty for this free software. For both users’ and authors’ sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users’ freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

**TERMS AND CONDITIONS**

1. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

1. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

1. Protecting Users’ Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work’s users, your or third parties’ legal rights to forbid circumvention of technological measures.

1. Conveying Verbatim Copies.

You may convey verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

1. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

* 1. The work must carry prominent notices stating that you modified it, and giving a relevant date.
  2. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.
  3. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
  4. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

1. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

* 1. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
  2. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
  3. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
  4. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
  5. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

1. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

* 1. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
  2. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
  3. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
  4. Limiting the use for publicity purposes of names of licensors or authors of the material; or
  5. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
  6. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

1. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

1. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

1. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

1. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

1. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

1. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

1. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

1. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

1. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

**END OF TERMS AND CONDITIONS**

**How to Apply These Terms to Your New Programs**

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

*one line to give the program's name and a brief idea of what it does.*

Copyright (C) *year* *name of author*

This program is free software: you can redistribute it and/or modify

it under the terms of the GNU General Public License as published by

the Free Software Foundation, either version 3 of the License, or (at

your option) any later version.

This program is distributed in the hope that it will be useful, but

WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

General Public License for more details.

You should have received a copy of the GNU General Public License

along with this program. If not, see <https://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

*program* Copyright (C) *year* *name of author*

This program comes with ABSOLUTELY NO WARRANTY; for details type ‘*show w*’.

This is free software, and you are welcome to redistribute it

under certain conditions; type ‘*show c*’ for details.

The hypothetical commands ‘*show w*’ and ‘*show c*’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <https://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <https://www.gnu.org/licenses/why-not-lgpl.html>.