**GNU Octave**

一种用于数值计算的高级交互语言

Octave版本9.1.0

2024年3月

2024年4月15日译



John W. Eaton

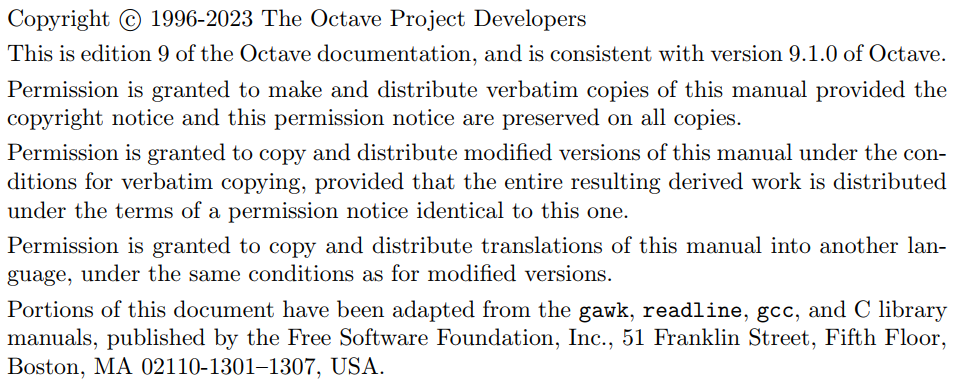
David Bateman

Søren Hauberg

Rik Wehbring

三山剑客 译

法律声明条款：



Copyright © 1996-2023 Octave项目开发者

这是Octave文档的第9版，与Octave的9.1.0版本一致。只要在所有副本上保留版权声明和本许可声明，就允许制作和分发本手册的逐字副本。

在逐字复制的条件下，允许复制和分发本手册的修改版本，只要整个结果派生作品在与本手册相同的许可声明条款下分发。

在与修改版本相同的条件下，允许复制和分发本手册的另一种语言翻译。

本文档的部分内容改编自gawk、readline、gcc和C库手册，这些手册由自由软件基金会公司(Franklin Street, 51 Fifth Floor)出版。

美国马萨诸塞州波士顿02110-1301-1307。

目录

**前言** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

致谢 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

引用Octave的出版物 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

如何同Octave社区做贡献 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

分发 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

**1导论** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

* 1. 运行Octave
  2. 简单示例
     1. 基本的计算
     2. 创建矩阵
     3. 矩阵计算
     4. 求解线性方程组
     5. 微分方程
     6. 生成图形输出
     7. 帮助文档
     8. 编辑键入内容
  3. 约定
     1. 字体
     2. 评价符号
     3. 印刷符号
     4. 错误信息
     5. 说明格式
        1. 功能描述样例
        2. 命令说明示例

**前言**

Octave最初打算作为一本本科水平的化学反应器设计教科书的配套软件，这本教科书是由威斯康星大学麦迪逊分校的James B. Rawlings和德克萨斯大学的John G. Ekerdt编写的。

显然，Octave现在不仅仅是一个“课件”包。虽然我们最初的目标有些模糊，但我们知道我们想要创造一些东西，使学生能够解决现实问题，并且他们可以用于化学反应器设计问题以外的许多事情。我们发现大多数学生很快就掌握了Octave的基础知识，并在短短几个小时内就能熟练地使用它。

虽然它最初是打算用来教授反应器设计的，但它已经被用在德克萨斯大学化学工程系的其他几个本科和研究生课程中，德克萨斯大学的数学系也一直在用它来教授微分方程和线性代数。最近，Octave被用作教授斯坦福在线机器学习课程(http://ml-class.org)的主要计算工具，该课程由Andrew Ng教授。成千上万的学生参加了这门课程。

如果你觉得Octave有用，请告诉我们。以便于我们了解Octave的其他用途。

几乎所有人都认为Octave（音阶）这个名字与音乐有关，但它实际上是约翰·w·伊顿(John W. Eaton)的一位前教授的名字，他写了一本著名的化学反应工程教科书，他也以快速“心算”计算的能力而闻名。我们希望这个软件能让很多人轻松地进行更有抱负的计算。

我们鼓励每个人在GNU通用公共许可证的条款下与他人共享本软件(参见GNU通用公共许可证)。我们还鼓励您为Octave编写和贡献额外的函数，并报告您可能遇到的任何问题，从而使Octave更有用。

**鸣谢**

许多人为Octave的发展做出了贡献。以下人员帮助编写了Octave的部分代码或以各种其他方式提供帮助(按字母顺序列出)。

|  |  |  |
| --- | --- | --- |
| Ben Abbott | Drew Abbot | NVS Abhilash |
| Andy Adler | Adam H. Aitkenhead | Fernando Alvarruiz |
| Joakim Andén | Giles Anderson | Joel Andersson |
| Lachlan Andrew | Pedro Angelo | Damjan Angelovski |
| Muthiah Annamalai | Markus Appel | Leonardo Araujo |
| Branden Archer | Willem Atsma | Marco Atzeri |
| Ander Aurrekoetxea | Shai Ayal | Sahil Badyal |
| Jeff Bai | Roger Banks | Ben Barrowes |
| Alexander Barth | David Bateman | Heinz Bauschke |
| Miguel Bazdresch | Julien Bect | Stefan Beller |
| Roman Belov | Markus Bergholz | Karl Berry |
| Andreas Bertsatos | Atri Bhattacharya | Ethan Biery |
| David Billinghurst | Don Bindner | Jakub Bogusz |
| Gaël Bonithon | Moritz Borgmann | Paul Boven |
| Richard Bovey | John Bradshaw | Marcus Brinkmann |
| Max Brister | Remy Bruno | Stefan Brüns |
| Clemens Buchacher | Ansgar Burchard | Antonius Burgers |
| Marco Caliari | Daniel Calvelo | John C. Campbell |
| Juan Pablo Carbajal | Jean-Francois Cardoso | Joao Cardoso |
| Larrie Carr | David Castelow | Vincent Cautaerts |
| Marco Cecchetti | Corbin Champion | Clinton Chee |
| Albert Chin-A-Young | Sunghyun Cho | Carsten Clark |
| Catalin Codreanu | J. D. Cole | Jacopo Corno |
| Andre da Costa Barros | Martin Costabel | Michael Creel |
| Richard Crozier | Jeff Cunningham | Martin Dalecki |
| Jacob Dawid | Jorge Barros de Abreu | Carlo de Falco |
| Thomas D. Dean | Philippe Defert | Bill Denney |
| Fabian Deutsch | Christos Dimitrakakis | Pantxo Diribarne |
| Vivek Dogra | John Donoghue | David M. Doolin |
| Carnë Draug | Sergey Dudoladov | Pascal A. Dupuis |
| John W. Eaton | Dirk Eddelbuettel | Pieter Eendebak |
| Paul Eggert | Stephen Eglen | Peter Ekberg |
| Abdallah K. Elshamy | Garrett Euler | Edmund Grimley Evans |
| Rolf Fabian | Francesco Faccio | Gunnar Farnebäck |
| Massimiliano Fasi | Stephen Fegan | Ramon Garcia Fernandez |
| Kasper H. Filtenborg | Torsten Finke | David Finkel |
| Guillaume Flandin | Colin Foster | Jose Daniel Munoz Frias |
| Brad Froehle | Castor Fu | Eduardo Gallestey |
| Walter Gautschi | Klaus Gebhardt | Driss Ghaddab |
| Eugenio Gianniti | Hartmut Gimpel | Michele Ginesi |
| Nicolo Giorgetti | Arun Giridhar | Michael D. Godfrey |
| Dave Goel | Michael Goffioul | Glenn Golden |
| Tomislav Goles | Keith Goodman | Brian Gough |
| Alexander Graf | Michael C. Grant | Steffen Groot |
| Etienne Grossmann | David Grundberg | Kyle Guinn |
| Vaibhav Gupta | Peter Gustafson | Kai Habel |
| Patrick Häcker | William P. Y. Hadisoeseno | Jaroslav Hajek |
| Benjamin Hall | Alexander Hansen | Kim Hansen |
| Gene Harvey | Søren Hauberg | Dave Hawthorne |
| Oliver Heimlich | Daniel Heiserer | Piotr Held |
| Martin Helm | Stefan Hepp | Martin Hepperle |
| Jordi Gutiérrez Hermoso | Israel Herraiz | Yozo Hida |
| Christian Himpe | Ryan Hinton | Roman Hodek |
| A. Scottedward Hodel | Júlio Hoffimann | Richard Allan Holcombe |
| Tom Holroyd | David Hoover | Kurt Hornik |
| Craig Hudson | Christopher Hulbert | Cyril Humbert |
| John Hunt | Stefan Husmann | Teemu Ikonen |
| Alan W. Irwin | Allan Jacobs | Marcel Jacobse |
| Geoff Jacobsen | Vytautas Jančauskas | Andrew Janke |
| Nicholas R. Jankowski | Mats Jansson | Robert Jenssen |
| Cai Jianming | Steven G. Johnson | Heikki Junes |
| Matthias Jüschke | Atsushi Kajita | Jarkko Kaleva |
| Avinoam Kalma | Mohamed Kamoun | Lute Kamstra |
| Fotios Kasolis | Thomas Kasper | Christof Kaufmann |
| Joel Keay | Mumit Khan | Paul Kienzle |
| Lars Kindermann | Aaron A. King | Erik Kjellson |
| Arno J. Klaassen | Alexander Klein | Lasse Kliemann |
| Geoffrey Knauth | Hendrik Koerner | Martin Köhler |
| Heine Kolltveit | Peter Konowski | Ken Kouno |
| Kacper Kowalik | Endre Kozma | Daniel Kraft |
| Nir Krakauer | Aravindh Krishnamoorthy | Oyvind Kristiansen |
| Artem Krosheninnikov | Piotr Krzyzanowski | Volker Kuhlmann |
| Ilya Kurdyukov | Tetsuro Kurita | Ben Kurtz |
| Philipp Kutin | Miroslaw Kwasniak | Rafael Laboissiere |
| Kai Labusch | Claude Lacoursiere | Walter Landry |
| Bill Lash | Dirk Laurie | Maurice LeBrun |
| Friedrich Leisch | Michael Leitner | Johannes Leuschner |
| Thorsten Liebig | Torsten Lilge | Jyh-miin Lin |
| Timo Lindfors | Benjamin Lindner | Ross Lippert |
| Yu Liu | David Livings | Barbara Locsi |
| Sebastien Loisel | Andrej Lojdl | Erik de Castro Lopo |
| Massimo Lorenzin | Emil Lucretiu | Yi-Hong Lyu |
| Hoxide Ma | Colin Macdonald | James Macnicol |
| Jens-Uwe Mager | Stefan Mahr | Rob Mahurin |
| Alexander Mamonov | Ricardo Marranita | Orestes Mas |
| Axel Mathéi | Makoto Matsumoto | Tatsuro Matsuoka |
| Christoph Mayer | Laurent Mazet | G. D. McBain |
| Ronald van der Meer | Markus Meisinger | Júlio Hoffimann Mendes |
| Ed Meyer | Thorsten Meyer | Stefan Miereis |
| Petr Mikulik | Linton Miller | Mike Miller |
| Serviscope Minor | Stefan Monnier | Rafael Monteiro |
| Stephen Montgomery-Smith | Anthony Morast | Antoine Moreau |
| Kai P. Mueller | Amod Mulay | Armin Müller |
| Hannes Müller | Victor Munoz | PrasannaKumar Muralidharan |
| Iain Murray | Nicholas Musolino | Markus Mützel |
| Carmen Navarrete | Todd Neal | Philip Nienhuis |
| Al Niessner | Felipe G. Nievinski | Rick Niles |
| Takuji Nishimura | Akira Noda | Kai Noda |
| Patrick Noffke | Victor Norton | Eric Norum |
| Krzesimir Nowak | Michael O’Brien | Cillian O’Driscoll |
| Peter O’Gorman | Thorsten Ohl | Kai T. Ohlhus |
| Serkan Önder | Arno Onken | Valentin Ortega-Clavero |
| Luis F. Ortiz | Carl Osterwisch | Janne Olavi Paanajärvi |
| Óvári | Scott Pakin | José Luis García Pallero |
| Jason Alan Palmer | Gabriele Pannocchia | Sylvain Pelissier |
| Rolando Pereira | Per Persson | Primozz Peterlin |
| Jim Peterson | Johannes Pfeifer | Danilo Piazzalunga |
| Nicholas Piper | Elias Pipping | Robert Platt |
| Hans Ekkehard Plesser | Sergey Plotnikov | Tom Poage |
| Nathan Podlich | Orion Poplawski | Ondrej Popp |
| Jef Poskanzer | Francesco Potortì | Konstantinos Poulios |
| Tejaswi D. Prakash | Jarno Rajahalme | Eduardo Ramos |
| Pooja Rao | James B. Rawlings | Eric S. Raymond |
| Balint Reczey | Joshua Redstone | Andy Register |
| Lukas Reichlin | Michael Reifenberger | Ernst Reissner |
| Reinhard Resch | Jens Restemeier | Anthony Richardson |
| Jason Riedy | E. Joshua Rigler | Sander van Rijn |
| Petter Risholm | Matthew W. Roberts | Melvin Robinson |
| Dmitry Roshchin | Peter Rosin | Andrew Ross |
| Fabio Rossi | Mark van Rossum | Joe Rothweiler |
| David Rörich | Kevin Ruland | Kristian Rumberg |
| Ryan Rusaw | Olli Saarela | Toni Saarela |
| Juhani Saastamoinen | Radek Salac | Mike Sander |
| Ben Sapp | Aleksej Saushev | Alois Schlögl |
| Michel D. Schmid | Julian Schnidder | Sebastian Schöps |
| Nicol N. Schraudolph | Sebastian Schubert | Lasse Schuirmann |
| Ludwig Schwardt | Thomas L. Scofield | Daniel J. Sebald |
| Dmitri A. Sergatskov | Vanya Sergeev | Marko Seric |
| Ahsan Ali Shahid | Baylis Shanks | Andriy Shinkarchuck |
| Robert T. Short | Paulo Silva | Joseph P. Skudlarek |
| John Smith | Julius Smith | Shan G. Smith |
| Peter L. Sondergaard | Rüdiger Sonderfeld | Joerg Specht |
| Quentin H. Spencer | Christoph Spiel | David Spies |
| Imad-Eddine Srairi | Andreas Stahel | Richard Stallman |
| Russell Standish | Ryan Starret | Brett Stewart |
| Doug Stewart | Jen Stewart | Jonathan Stickel |
| Judd Storrs | Thomas Stuart | Bernardo Sulzbach |
| Ivan Sutoris | John Swensen | Daisuke Takago |
| Ariel Tankus | Falk Tannhäuser | Duncan Temple Lang |
| Matthew Tennyhg | Remi Thebault | Kris Thielemans |
| Georg Thimm | Paul Thomas | Corey Thomasson |
| Andrew Thornton | Olaf Till | Petter Tomner |
| Christophe Tournery | Thomas Treichl | Abhinav Tripathi |
| Karsten Trulsen | David Turner | Frederick Umminger |
| Utkarsh Upadhyay | José Vallet | Stefan van der Walt |
| Peter Van Wieren | James R. Van Zandt | Risto Vanhanen |
| Gregory Vanuxem | Mihas Varantsou | Ivana Varekova |
| Sébastien Villemot | Marco Vitetta | Daniel Wagenaar |
| Steven Waldrip | Thomas Walter | Jun Wang |
| Andreas Weber | Olaf Weber | Thomas Weber |
| Rik Wehbring | Bob Weigel | Andreas Weingessel |
| Martin Weiser | Michael Weitzel | David Wells |
| Joachim Wiesemann | Alexander Wilms | Joe Winegarden |
| Georg Wiora | Eddy Xiao | Sahil Yadav |
| Fook Fah Yap | Sean Young | Michele Zaffalon |
| Serhiy Zahoriya | Johannes Zarl | Michael Zeising |
| Federico Zenith | Claudius Zingerli | Alex Zvoleff |

特别感谢以下支持Octave开发的个人和组织：

* 美国能源部，通过拨款号DE-FG02-04ER25635。
* 俄亥俄超级计算机中心的Ashok Krishnamurthy, David Hudak, Juan Carlos Chaves和Stanley C. Ahalt。
* 国家科学基金，资助号:CTS-0105360、CTS-9708497、CTS-9311420、CTS-8957123和CNS-0540147。
* 德州-威斯康辛建模与控制联盟(TWMCC)的工业成员。
* 威斯康星大学麦迪逊分校化学工程教授Paul A. Elfers。
* 数字设备公司，作为其外部研究计划的一部分。
* Sun Microsystems, Inc.申请学术设备补助金。
* 国际商业机器公司(International Business Machines, Inc.)，因其向得克萨斯大学工程学院提供设备而获奖。
* 德士古化学公司，为继续开发该软件提供资金。
* 德克萨斯大学工程学院，为卓越研究提供挑战补充，并提供学术发展基金赠款。
* 德克萨斯州通过德克萨斯州先进技术计划提供资金，批准号为003658-078。
* Noel Bell，高级工程师，德州奥斯汀德士古化学公司。
* Los Alamos国家实验室Continuum Dynamics (CCS-2)组长John A. Turner，感谢他注册了https://octave.org域名。
* James B. Rawlings，威斯康星大学麦迪逊分校化学与生物工程系教授。
* Richard Stallman，感谢您编写GNU。

如果没有用于Octave的GNU软件，这个项目是不可能实现的。

**引用Octave的出版物**

鉴于许多开发人员多年来做出的许多贡献，在研究或准备数据的过程中使用Octave时，在出版物中引用Octave是一种礼貌地行为。引用功能可以自动为Octave或其任何软件包生成推荐的引用文本。关于如何使用引文，请参阅下面的帮助文本。

: **citation**

: **citation** *package*

显示在出版物中引用GNU Octave或其软件包的说明。

当不带参数调用时，显示有关如何引用核心GNU Octave系统的信息。

当给定包名称package时，显示引用特定命名包的信息。请注意，有些包可能还没有关于如何引用它们的说明。

GNU Octave开发人员及其活跃的软件包作者社区已经投入了大量的时间和精力来创建今天的GNU Octave。当你使用GNU Octave及其软件包时，请注明出处。

**如何同Octave社区做贡献**

有许多方法可以帮助您使Octave成为一个更好的系统。也许贡献的最重要方式是为解决新问题编写高质量的代码，并使您的代码免费供其他人使用。详细信息请参见<https://www.octave.org/get-involved.html>。

如果您发现Octave有用，请考虑提供额外的资金以继续其开发。即使是少量的额外资金，也可以在用于开发和支持的时间上产生重大影响。

支持Octave发展的捐款可以在https://www.octave.org/donate网站上进行。

如果您不能提供资金或贡献代码，您仍然可以通过报告您发现的任何错误并提供改进Octave的方法建议来帮助Octave变得更好、更可靠。有关如何编写有用的bug报告的提示，请参阅已知的问题原因。

**分发**

Octave是免费软件。这意味着每个人都可以自由地使用它，并在一定条件下自由地重新发布它。然而，Octave并不属于公有领域。它是受版权保护的，并且对其发行有限制，但是这些限制是为了确保其他人将拥有与您相同的使用和重新发行Octave的自由。精确的条件可以在Octave附带的GNU通用公共许可证中找到，它也出现在GNU通用公共许可证中。

要下载Octave，请访问<https://www.octave.org/download.html>。

**1导论**

GNU Octave是一种高级语言，主要用于数值计算。它通常用于求解线性和非线性方程、数值线性代数、统计分析和执行其他数值实验等问题。它还可以用作面向批处理的语言，用于自动数据处理。

当前版本的Octave在图形用户界面(GUI)中执行。GUI托管一个集成开发环境(IDE)，其中包括带有语法高亮显示的代码编辑器、内置调试器、文档浏览器以及语言本身的解释器。当然也可以使用命令行接口。

GNU Octave是一个可自由分发的软件。您可以根据自由软件基金会发布的GNU通用公共许可证条款重新发布和/或修改它。GPL包含在本手册中，请参阅GNU通用公共许可证。

本手册提供了关于如何安装、运行、使用和扩展GNU Octave的全面文档。其他章节描述了如何报告bug和帮助贡献代码。

本文档对应于Octave版本9.1.0。

**1.1运行Octave**

在大多数系统上，Octave是用shell命令 **‘octave’**启动的。这将启动图形用户界面。GUI中的中心窗口是Octave命令行界面。在此窗口中，Octave显示一条初始消息，然后显示一个提示，表明它已准备好接受输入。如果您选择了传统的命令行界面，那么在运行shell的同一窗口中只会出现命令提示符。在任何一种情况下，您都可以立即开始键入Octave命令。

如果遇到麻烦，通常可以通过输入Control-C（简称C-c）来中断Octave。C-c的名字来源于你按住CTRL然后按下c来输入它。这样做通常会返回到Octave的提示符。

要退出Octave，请在Octave提示符处键入quit或exit。

在支持作业控制的系统上，可以通过向Octave发送SIGTSTP信号（通常是输入C-z ）来挂起Octave。

**1.2简单示例**

下面的章节详细描述了Octave的所有特性，但在此之前，给出它的一些功能示例可能会有所帮助。

如果您是Octave的新手，我们建议您尝试使用这些示例来开始学习Octave。标记为' octave:13> '的行是您键入的行，每个行以回车结束。Octave会给出一个答案，或者显示一个图表。

**1.2.1基础计算**

Octave可以很容易地用于基本的数值计算。Octave知道算术运算(+，-，\*，/)，幂运算(^)，自然对数/指数(log, exp)，以及三角函数(sin, cos，…)。此外，Octave计算适用于实数或虚数(i,j)。一些数学常数，如自然对数的底(e)和圆的周长与直径之比(pi)都是预先定义的。

例如，为了验证欧拉恒等式，

键入下面的语句，它将在计算的容差范围内求值为-1。

octave:1> exp (i\*pi)

**1.2.2创建矩阵**

向量和矩阵是数值分析的基本组成部分。要创建一个新的矩阵并将其存储在一个变量中，以便以后可以引用它，键入命令

octave:1> A = [ 1, 1, 2; 3, 5, 8; 13, 21, 34 ]

Octave将以整齐排列的列打印矩阵作为响应。Octave使用逗号或空格分隔一行中的条目，并使用分号或回车分隔一行和下一行。以分号结束命令告诉Octave不打印命令的结果。例如,

octave:2> B = rand (3, 2);

将创建一个3行2列的矩阵，每个元素设置为0到1之间的随机值。

要显示变量的值，只需在提示符处输入变量的名称。例如，要显示存储在矩阵B中的值，可以输入命令

octave:3> B

**1.2.3矩阵运算**

Octave使用标准的数学符号，与低级语言相比，它的优点是运算符可以作用于标量、向量、矩阵或n维数组。例如，要将矩阵A乘以标量值，请键入命令

octave:4> 2 \* A

要将两个矩阵A和B相乘，请键入命令

octave:5> A \* B

并形成矩阵乘积的transpose (A) \* A，键入命令

octave:6> A' \* A

**1.2.4求解线性方程组**

线性方程组在数值分析中无处不在。要解线性方程组Ax = b，使用左除法算子' \ ':

x = A \ b

这在概念上等同于inv (A) \* b，但避免了直接计算矩阵的逆。

如果系数矩阵是奇异的，Octave将打印一条警告消息并计算最小范数解。

一个简单的例子来自化学，需要得到平衡的化学方程。考虑一下氢和氧燃烧产生水。

上面的等式是不准确的。质量守恒定律要求每一种分子的数量在方程的左右两边保持平衡。用氢和氧的单独方程写出变量总反应，我们会发现:

在Octave的解决方案只需要三个步骤。

octave:1> A = [ 2, 0; 0, 2 ];

octave:2> b = [ 2; 1 ];

octave:3> x = A \ b

**1.2.5微分方程的积分**

Octave内置函数用于求解非线性微分方程的形式

初始条件

为了让Octave对这种形式的方程进行积分，你必须首先给出函数f(x,t)的定义。这很简单，可以通过直接在命令行中输入函数体来实现。例如，下面的命令定义了一对有趣的非线性微分方程的右侧函数。注意，当您输入一个函数时，Octave会响应一个不同的提示，表明它正在等待您完成输入

octave:1> function xdot = f (x, t)

>

> r = 0.25;

> k = 1.4;

> a = 1.5;

> b = 0.16;

> c = 0.9;

> d = 0.8;

>

> xdot(1) = r\*x(1)\*(1 - x(1)/k) - a\*x(1)\*x(2)/(1 + b\*x(1));

> xdot(2) = c\*a\*x(1)\*x(2)/(1 + b\*x(1)) - d\*x(2);

>

> endfunction

给定初始条件

octave:2> x0 = [1; 2];

输出时间的集合作为列向量(注意，第一个输出时间对应于上面给出的初始条件)

octave:3> t = linspace (0, 50, 200)';

微分方程组的积分很容易:

octave:4> x = lsode (“f”, x0, t);

函数lsode使用利弗莫尔求解常微分方程，在A. C. Hindmarsh, ODEPACK, a Systematized Collection of ODE Solvers, in: Scientific Computing, R. S. Stepleman et al. (Eds.), North-Holland, Amsterdam, 1983, pages 55–64.中描述。

**1.2.6生成图形输出**

要以图形方式显示前面示例的解决方案，可以使用该命令

octave:1> plot (t, x)

Octave将自动创建一个单独的窗口来显示绘图。

要保存已显示在屏幕上的绘图，请使用print命令。例如,

print foo.pdf

将创建一个名为foo.pdf的文件，其中包含以可移植文档格式呈现的当前绘图。命令

help print

解释print命令的更多选项，并提供其他输出文件格式的列表。

**1.2.7帮助和文档**

Octave有一个广泛的帮助设施。以打印形式提供的相同文档也可以从Octave提示符中获得，因为两种形式的文档都是从相同的输入文件创建的。

为了获得良好的帮助，您首先需要知道要使用的命令的名称。这个函数的名称可能并不总是很明显，但是键入help --list是一个很好的开始。这将显示Octave当前会话中可用的所有操作符、关键字、内置函数和可加载函数。另一种方法是使用lookfor函数(在获取帮助的命令中有描述)搜索文档。

一旦您知道了希望使用的函数的名称，您就可以通过简单地将名称作为帮助的参数包含在函数中来获得更多的帮助。例如,

help plot

将显示plot函数的帮助文本。

在Octave的帮助工具中，允许您从Octave中读取打印手册的完整文本的部分通常使用一个称为Info的单独程序。当您调用Info时，您将被放入包含整个Octave手册的菜单驱动程序中。本手册中提供了Info的使用帮助，请参见获取帮助命令。

**1.2.8编辑您输入的内容**

在Octave提示符下，您可以使用Emacs或vi样式的编辑命令收回、编辑和重新发出以前的命令。默认的键绑定使用emacs风格的命令。例如，要回忆前面的命令，按Control-p(简称C-p)。这样做通常会返回前一行输入。C-n会调出下一行输入，C-b会将光标移到本行上，C-f会将光标移到本行上，以此类推。

本手册给出了命令行编辑功能的完整描述，请参阅命令行编辑。

**1.3约定**

本节解释本手册中使用的符号约定。您可能想跳过这一节，稍后再参考它。

**1.3.1字体**

Octave代码的示例如下字体或形式:svd (a)。表示变量或函数参数的名称以这种字体或形式出现:first-number。您在shell提示符中键入的命令以这种字体或形式显示:' octave --no-init-file '。在Octave提示符下输入的命令有时会以这种字体或形式出现:foo –bar --baz。键盘上的特定键以这种字体或形式显示:RET。

**1.3.2求值符号**

在本手册的示例中，您计算的表达式的结果用'⇒'表示。例如:

sqrt (2)

⇒ 1.4142

您可以将其理解为“sqrt(2)的计算结果为1.4142”。

在某些情况下，表达式返回的矩阵值是这样显示的

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

⇒ [ 1, 0; 0, 1 ]

在其他情况下，它们是这样显示的

eye (3)

⇒ 1 0 0

0 1 0

1. 0 1

以便清楚地显示结构的结果。

有时为了帮助描述一个表达式，会显示另一个表达式产生相同的结果。表达式的完全等价用“≡”表示。例如:

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

≡

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

≡

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

**1.3.3打印符号**

本手册中的许多示例在评估时都会打印文本。在本手册中，由示例产生的打印文本用' -| '表示。通过计算表达式返回的值显示为'⇒'(在下一个示例中为1)，并在单独的行中显示。

printf ("foo %s\n", "bar")

-| foo bar

⇒ 1

**1.3.4错误信息**

一些例子显示错误。这通常会在终端上显示一条错误消息。错误信息显示在以error: 开头的行中。

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

error: fieldnames: Invalid input argument

**1.3.5描述的格式**

本手册对函数和命令的描述格式统一。描述的第一行包含项目的名称，后面跟着它的参数(如果有的话)。如果有多种方法调用函数，则列出每种允许的形式。

接下来的几行是描述，有时还有例子。

**1.3.5.1功能举例说明**

在函数描述中，被描述的函数的名称首先出现。在同一行后面跟着一个参数列表。用于参数的名称也在描述的主体中使用。

在列举了所有调用形式之后，下一行是对函数的简洁的一句话总结。

在总结之后，可能会有关于输入和输出的文档、函数使用示例、关于所使用算法的说明以及对相关函数的引用。

下面是虚函数foo的描述:

: foo (x)

: foo (x, y)

: foo (x, y, …)

从y中减去x，然后将剩余的参数添加到结果中。

输入x必须是数字标量、矢量或数组。

如果没有提供可选输入y，则默认为19。

示例：

foo (1, [3, 5], 3, 9)

⇒ [ 14, 16 ]

foo (5)

⇒ 14

更一般的说：

foo (w, x, y, …)

≡

x - w + y + …

**参见**: bar

任何名称包含类型名称(例如，整数或矩阵)的参数都应属于该类型。名为object的参数可以是任何类型。具有其他类型名称的参数(例如，new\_file)在函数描述中进行了具体讨论。在某些部分中，在开始部分描述了几个函数参数的共同特征。

**1.3.5.2命令示例说明**

命令是可以调用的函数，而不需要用圆括号括住它们的参数。命令描述的格式与功能描述类似。例如，下面是Octave的diary命令的描述:

**: diary**

**: diary on**

**: diary off**

**: diary filename**

**: [status, diaryfile] = diary**

记录所有命令和它们产生的输出的列表，混合在一起，就像它们出现在终端上一样。

有效的选项有:

On 在当前工作目录下的一个名为diary的文件中开始记录会话。

Off 停止在日志文件中记录会话。

Filename 将会话记录在文件名中。

在没有输入或输出参数的情况下，diary切换当前日志状态。

如果请求输出参数，则diary忽略输入并返回当前状态。布尔状态指示记录是打开还是关闭，diaryfile是存储会话的文件的名称。

**参见：**history, evalc。

**2新手入门**

本章解释了Octave的一些基本特性，包括如何启动Octave会话，在命令提示符下获得帮助，编辑命令行，以及编写可以在shell中作为命令执行的Octave程序。

**2.1从命令行调用Octave**

通常，Octave是通过运行程序 **'Octave'** 来交互使用的，不带任何参数。一旦启动，Octave从终端读取命令，直到您告诉它退出。

您还可以在命令行上指定文件的名称，Octave将从指定的文件中读取和执行命令，然后在完成时退出。

您可以通过使用下一节中描述的命令行选项进一步控制Octave的启动方式，Octave本身可以提醒您可用的选项。输入 **'octave --help'** 显示所有可用选项并简要描述它们的用法(' **octave –h**是一个更短的等价写法 ')

**2.1.1命令行选项**

下面是Octave接受的命令行选项的完整列表。

*--built-in-docstrings-file filename*

指定包含Octave内置函数的文档字符串的文件名。该值通常是正确的，只有在特殊情况下才需要指定。

*--debug*

*-d*

进入解析器调试模式。使用此选项将导致Octave的解析器打印大量关于它读取的命令的信息，并且可能仅在您实际尝试调试解析器时才有用。

*--doc-cache-file filename*

指定要使用的文档缓存文件的名称。命令行上指定的filename的值将覆盖环境中找到的OCTAVE\_DOC\_CACHE\_FILE的任何值，但不会覆盖系统中使用doc\_cache\_file函数的任何命令或用户启动文件。

*--echo-commands*

*-x*

在执行命令时回显命令。

*--eval code*

计算代码并在完成时退出，除非使用 *--persist*指定。

*--exec-path path*

指定搜索要运行的程序的路径。命令行上指定的path值将覆盖环境中找到的OCTAVE\_EXEC\_PATH的任何值，但不会覆盖系统或用户启动文件中调用*EXEC\_PATH*函数的任何命令。

*--gui*

启动图形用户界面(GUI)。

*--help*

*-h*

打印简短的帮助信息并退出。

*--image-path path*

在图像搜索路径的头部添加path。命令行上指定的*path*值将覆盖环境中找到的*OCTAVE\_IMAGE\_PATH*的任何值，但不会覆盖系统或用户启动文件中调用*IMAGE\_PATH*函数的任何命令。

*--info-file filename*

指定要使用的info文件的名称。命令行中指定的*filename*值将覆盖环境中找到的*OCTAVE\_INFO\_FILE*的任何值，但不会覆盖系统中使用*info\_file*函数的任何命令或用户启动文件。

*--info-program program*

指定要使用的info程序的名称。命令行中指定的*program*值将覆盖环境中找到的*OCTAVE\_INFO\_PROGRAM*的任何值，但不会覆盖系统或用户启动文件中使用*info\_program*函数的任何命令。

*--interactive*

*-i*

强制交互行为。这对于通过远程shell命令或在Emacs shell缓冲区中运行Octave非常有用。

*--line-editing*

强制readline用于命令行编辑。

*--no-gui*

禁用图形用户界面(GUI)，使用命令行界面(CLI)。这是默认行为，但是这个选项对于覆盖之前的 *--gui*可能很有用。

*--no-history*

*-H*

禁用命令行历史记录。

*--no-init-file*

不读取初始化文件 *~/.octaverc* 和 *.octaverc* 。

*--no-init-path*

不要初始化函数文件的搜索路径以包含默认位置。

*--no-line-editing*

禁用命令行编辑。

*--no-site-file*

不要读取站点范围的*octaverc*初始化文件。

*--no-window-system*

*-W*

禁用包括图形在内的窗口系统。这就形成了一个严格的终端环境。

*--norc*

*-f*

不要在启动时读取任何系统或用户初始化文件。这相当于同时使用两个选项*--no-init-file*和*--no-site-file。*

*--path path*

*-p path*

在函数文件的搜索路径的头部添加path。命令行上指定的*path*值将覆盖在环境中找到的任何*OCTAVE\_PATH*值，但不会覆盖系统或用户启动文件中通过某个路径函数设置内部加载路径的任何命令。

*--persist*

在*--eval*或从命令行上命名的文件中读取之后进入交互模式。

*--silent*

*--quiet*

*-q*

不要在启动时打印常见的问候语和版本信息。

*--texi-macros-file filename*

指定包含Texinfo宏的文件名供makeinfo使用。

*--traditional*

*--braindead*

为了与MATLAB兼容，请将用户首选项的初始值设置为以下值

PS1 = ">> "

PS2 = ""

beep\_on\_error = true

confirm\_recursive\_rmdir = false

crash\_dumps\_octave\_core = false

optimize\_diagonal\_matrix = false

optimize\_permutation\_matrix = false

optimize\_range = false

fixed\_point\_format = true

history\_timestamp\_format\_string = "%%-- %D %I:%M %p --%%"

print\_empty\_dimensions = false

print\_struct\_array\_contents = true

save\_default\_options = "-mat-binary"

struct\_levels\_to\_print = 0

并禁用以下警告

Octave:abbreviated-property-match

Octave:colon-nonscalar-argument

Octave:data-file-in-path

Octave:empty-index

Octave:function-name-clash

Octave:possible-matlab-short-circuit-operator

请注意，这不会启用*Octave:language-extension*警告，如果您希望被告知编写在Octave中工作的代码而不是MATLAB(请参阅warning, warning\_ids)，则可能需要此警告。

--verbose

-V

打开详细输出。

--version

-v

打印程序版本号并退出。

file

从*file*执行命令。完成后退出，除非还指定了*--persist* 。

Octave还包括几个函数，它们返回有关命令行的信息，包括参数的数量和所有选项。

*: args =* ***argv*** *()*

返回传递给Octave的命令行参数。

例如，如果您使用命令调用Octave

octave --no-line-editing --silent

*argv*将返回一个字符串单元格数组，其中包含元素*--no-line-editing*和*-–silent*。

如果您编写可执行的Octave脚本，*argv*将返回传递给脚本的参数列表。有关如何创建可执行的Octave脚本的示例，请参阅可执行的Octave程序。

**参见：**program\_name, cmdline\_options。

*: opt\_struct =* ***cmdline\_options*** *()*

返回一个结构，包含传递给Octave的命令行参数的详细信息。

编程注意:这个函数提供了大量关于Octave解析命令行选项的信息，对于调试Octave可能更有用，而不是一般使用。

**参见：**argv, program\_name。

*: name =* ***program\_name*** *()*

返回program\_invocation\_name返回值的filename组件。

**参见：**program\_invocation\_name, argv。

*: name =* ***program\_invocation\_name*** *()*

返回在shell提示符处键入的字符串，以运行Octave。

该字符串可以包括路径组件以及程序文件名。

如果从命令行执行脚本（例如，*octave foo.m*）或使用可执行的octave脚本，则程序名称将设置为脚本的名称。有关如何创建可执行的Octave脚本的示例，请参阅可执行的Octave程序。

**参见：**program\_name, argv。

下面是一个使用这些函数来重现调用Octave的命令行的示例。

printf ("%s", program\_name ());

arg\_list = argv ();

for i = 1:nargin

printf (" %s", arg\_list{i});

endfor

printf ("\n");

有关如何从单元格数组检索对象的说明，请参见索引单元格数组，有关变量*nargin*的信息请参见定义函数。

**2.1.2启动文件**

当Octave启动时，它会从以下列表中的文件中查找要执行的命令。这些文件可以包含任何有效的Octave命令，包括函数定义。

*octave-home/share/octave/site/m/startup/octaverc*

其中*Octave -home*是安装Octave的目录（默认为*/usr/local*）。提供此文件是为了全局地对您站点上的所有用户对已安装的所有Octave版本的默认Octave环境进行更改。在对该文件进行更改时应小心，因为您站点上的所有Octave用户都会受到影响。默认文件可以被环境变量*OCTAVE\_SITE\_INITFILE*覆盖。

*octave-home/share/octave/version/m/startup/octaverc*

其中*Octave -home*是安装Octave的目录（默认为*/usr/local*）， *version*是Octave的版本号。提供此文件是为了使默认Octave环境的更改可以针对特定版本Octave的所有用户全局进行。在对该文件进行更改时应小心，因为您站点上的所有Octave用户都会受到影响。默认文件可以被环境变量*OCTAVE\_VERSION\_INITFILE*覆盖。

*config-dir/octave/octaverc*

其中*config-dir*是用户本地配置文件的平台相关位置（例如，在许多类unix操作系统上为*$XDG\_CONFIG\_HOME*，在Windows上为*%APPDATA%*）。

*~/.octaverc*

该文件用于对默认的Octave环境进行个人更改。

*.octaverc*

该文件可用于更改特定项目的默认Octave环境。Octave在读取*~/.octaverc*后在当前目录中搜索该文件。cd命令在*~/.octaverc*文件将影响Octave搜索 *.octaverc*的目录。

如果你在主目录下启动Octave，命令从文件*~/.octaverc*只执行一次。

*startup.m*

该文件用于对默认的Octave环境进行个人更改。它的执行是为了与MATLAB兼容，但是*~/.octaverc*是配置更改的首选位置。

如果您使用 *--verbose*选项而不使用 *--silent*选项调用Octave，则在读取每个启动文件时将显示一条消息。

启动文件总是在系统的区域设置字符集中处理(独立于设置的m-file编码，例如，在GUI属性中)。换句话说，系统的区域设置字符集一直有效，直到用户手动设置m文件编码(例如，在一个启动文件中)并触发对任何相关m文件的重新解析。Octave可以通过mfile\_encoding函数强制使用新的编码:

\_\_mfile\_encoding\_\_ ("utf-8"); # set new encoding

clear ("functions"); # re-parse all .m files in the new encoding

这将改变用于解释所有随后运行的启动文件和m文件(不包括当前正在执行的文件)的编码。

**2.2退出Octave**

关机是用*exit*或*quit*命令启动的（它们是等价的）。与启动类似，Octave有一个关机过程，可以由用户脚本文件自定义。在关机期间，Octave将搜索脚本文件*finish.m*在函数加载路径中。保存所有工作空间变量或清理临时文件的命令可以放在那里。在关机时执行的其他函数可以用*atexit*注册。

: **quit**

: **quit** *cancel*

: **quit** *force*

: **quit** *("cancel")*

: **quit** *("force")*

: **quit** *(status)*

: **quit** *(status, "force")*

: **exit***(…)*

退出当前的Octave会话。

如果提供了可选的整数值*status*，将该值作为Octave的退出状态传递给操作系统。默认值为零。

退出时，Octave将尝试运行m-file *finish.m*，如果存在的话。保存工作区或清理临时文件的用户命令可以放在该文件中。或者，可以使用*atexit*命令安排另一个m文件运行。如果在执行*finish.m*文件时发生错误，Octave不退出，控制返回到命令提示符。

如果提供了可选参数"*cancel*"，则Octave不会退出，并将控制权返回给命令提示符。这个功能允许*finish.m*文件取消退出进程。

如果用户偏好在退出前请求确认，Octave将显示一个对话框，并为用户提供取消退出过程的选项。

如果提供了可选参数“*force*”，则不要求任何确认，并跳过*finish.m*文件。

编程注意:exit是quit的别名，可以互换使用。

**参见：**atexit。

: **atexit** *(fcn)*

: **atexit** *(fcn, true)*

: **atexit** *(fcn, false)*

: *status =* **atexit** *(fcn, false)*

注册一个函数，当Octave退出时调用。

例如,

function last\_words ()

disp ("Bye bye");

endfunction

atexit ("last\_words");

当Octave退出时，将打印消息”*Bye bye*”。

附加参数*flag*将在Octave退出时从要调用的函数列表中注册或取消注册*fcn*。如果*flag*为真，则注册该函数，如果*flag*为假，则未注册。例如，在注册了上面的*last\_words*函数之后，

atexit ("last\_words", false);

将从列表中删除该函数，并且Octave在退出时不会调用last\_words。

可选输出状态仅在注销函数时可用。如果注销成功，该值为true，否则为false。

编程简注:*atexit*只从列表中删除函数的第一次出现;如果使用*atexit*将一个函数多次放入列表中，那么它也必须多次从列表中删除。

**参见：**quit。

**2.3获取帮助命令**

本手册的全文可通过命令*doc*从Octave提示符中获得。此外，用户编写的单个函数和变量的文档也可以通过*help*命令获得。本节介绍用于读取手册和用户提供的函数和变量的文档字符串的命令。有关如何记录编写的函数的更多信息，请参阅函数文件。

: **help** *name*

: **help** *--list*

: **help** *.*

: **help**

: *help\_text =* **help** *(…)*

显示名称的帮助文本。

例如，*help help*命令打印一条描述*help*命令的短消息。

给定单个参数 *--list*，列出Octave当前会话中可用的所有操作符、关键字、内置函数和可加载函数。

给定单个参数，列出Octave当前会话中可用的所有操作符。

如果在没有任何参数的情况下调用，帮助将显示有关如何从命令行访问帮助的说明。

*help*命令可以提供关于大多数操作符的信息，但是name必须用单引号或双引号括起来，以防止Octave解释器对name起作用。例如，*help* "+"显示加法运算符的帮助。

**参见：**doc, for, which, info。

: **doc** *function\_name*

: **doc**

使用GNU Info浏览器，直接从打印手册的在线版本显示函数*function\_name*的文档。

如果调用时不带参数，则从一开始显示手册。

例如，在本手册的在线版本中，命令*doc rand*在*rand*节点启动GNU Info浏览器。

一旦GNU Info浏览器开始运行，就可以使用命令*C-h*来帮助使用它。

**参见：**help。

: **lookfor** *str*

: **lookfor** *-all str*

: *[fcn, help1str] =* **lookfor** *(str)*

: *[fcn, help1str] =* **lookfor** *("-all", str)*

在当前函数搜索路径中所有函数的文档中搜索字符串*str*。

默认情况下，*lookfor*只在找到的每个函数的帮助字符串的第一句话中查找*str*。可以使用*“-all”*参数搜索每个函数的整个帮助文本。所有搜索不区分大小写。

当不带输出参数调用时，*lookfor*将匹配函数的列表打印到终端。否则，输出参数*fcns*包含函数名，*help1str*包含每个函数的帮助字符串的第一句话。

编程简注: *lookfor*正确识别帮助文本第一句话的能力取决于函数帮助的格式。所有 Octave 核心函数的格式都是正确的，但不能保证外部包和用户提供的函数也是如此。因此，使用*“-all”*参数可能是必要的，以找到不是 Octave 一部分的相关函数。

缓存文档文件大大提高了查找速度。更多信息请参见doc\_cache\_create。

**参见：**help, doc, which, path, doc\_cache\_create。

要查看当前版本Octave的新功能，请使用*news*功能。

: **news**

: **news** *package*

显示Octave或已安装包的当前NEWS文件。

当不带参数调用时，显示Octave的NEWS文件。

当给定一个包名为*package*时，显示该包的当前NEWS文件。

**参见：**ver, pkg。

: **info** *()*

显示GNU Octave社区的联系信息。

: **warranty** *()*

描述复制和分发Octave的条件。

以下函数可用于更改用于显示文档的程序，以及可以在何处找到文档。

: *val =* **info\_file** *()*

: *old\_val =* **info\_file** *(new\_val)*

: *old\_val =* **info\_file** *(new\_val, "local")*

查询或设置指定Octave info文件名的内部变量。

默认值为*octave -home/share/info/octave.info*，其中*octave -home*为Octave安装的根目录。默认值可以被环境变量*OCTAVE\_INFO\_FILE*或命令行参数 *--info-file FNAME*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：**info\_program, doc, help, makeinfo

: *val =***info\_program** *()*

: *old\_val =* **info\_program** *(new\_val)*

: *old\_val =* **info\_program** *(new\_val, "local")*

查询或设置指定要运行的info程序名称的内部变量。

默认值为*info*。默认值可以被环境变量*OCTAVE\_INFO\_PROGRAM*或命令行参数*--info-program NAME*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：**info\_file, doc, help, makeinfo\_program。

: *val =* **makeinfo\_program** *()*

: *old\_val =* **makeinfo\_program** *(new\_val)*

: *old\_val =* **makeinfo\_program** *(new\_val, "local")*

查询或设置指定Octave运行的程序名称的内部变量，以格式化包含Texinfo标记命令的帮助文本。

默认值为*makeinfo*。

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：**texi\_macros\_file, info\_file, info\_program, doc, help。

: *val =* **texi\_macros\_file** *()*

: *old\_val =* **texi\_macros\_file** *(new\_val)*

: *old\_val =***texi\_macros\_file** *(new\_val, "local")*

查询或设置内部变量，该变量指定包含Texinfo宏的文件的名称，这些宏在传递给makeinfo之前被附加到文档字符串中。

默认值为*octave-home/share/octave/version/etc/macros*。其中*octave -home*是Octave安装的根目录，*version*是Octave的版本号。默认值可以被环境变量*OCTAVE\_TEXI\_MACROS\_FILE*或命令行参数 *--texi-macros-file FNAME*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：**makeinfo\_program。

: *val =* **doc\_cache\_file** *()*

: *old\_val =* **doc\_cache\_file** *(new\_val)*

: *old\_val =* **doc\_cache\_file** *(new\_val, "local")*

查询或设置指定Octave文档缓存文件名称的内部变量。

缓存文件可以显著提高查找命令的性能。默认值为*octave-home/share/octave/version/etc/doc-cache*，其中*octave-home*为Octave安装的根目录，*version*为Octave的版本号。默认值可以被环境变量*OCTAVE\_DOC\_CACHE\_FILE*或命令行参数*--doc-cache-file FNAME*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：**doc\_cache\_create, lookfor, info\_program, doc, help, makeinfo\_program。

**参见：**lookfor。

: *val =* **built\_in\_docstrings\_file** *()*

: *old\_val =* **built\_in\_docstrings\_file** *(new\_val)*

: *old\_val =* **built\_in\_docstrings\_file** *(new\_val, "local")*

查询或设置内部变量，该变量指定包含内置Octave函数的文档字符串的文件名。

默认值为octave-home/share/octave/version/etc/built-in-docstrings，其中*octave-home*是octave安装的根目录，*version*是octave的版本号。默认值可以被环境变量*OCTAVE\_BUILT\_IN\_DOCSTRINGS\_FILE*或命令行参数 *--built-in-docstrings-file FNAME*覆盖。

注意:此变量仅在Octave初始化自身时使用。在运行Octave的会话期间修改它将没有影响。

: *val =* **suppress\_verbose\_help\_message** *()*

: *old\_val =***suppress\_verbose\_help\_message** *(new\_val)*

: *old\_val =* **suppress\_verbose\_help\_message** *(new\_val, "local")*

查询或设置内部变量，该变量控制Octave是否将额外的帮助信息添加到*help*命令和内置命令的使用消息的输出末尾。

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。

以下函数主要在Octave内部用于生成文档。这里记录它们是为了完整性，也因为它们可能偶尔对用户有用。

: **doc\_cache\_create** *(out\_file, directory)*

: **doc\_cache\_create** *(out\_file)*

: **doc\_cache\_create** *()*

为*directory*中的所有函数生成文档缓存。

为目录中的所有函数生成一个文档缓存，该缓存可以是单个字符串或字符串的单元数组。该缓存用于加速*lookfor*函数。

缓存保存在*out\_file*文件中，如果没有给出默认值*doc-cache*。

如果没有给出目录(或者是空矩阵)，则生成用于内置函数、操作符和关键字的缓存。

**参见：**doc\_cache\_file, lookfor, path。

: *[text, format] =* **get\_help\_text** *(name)*

返回函数名的原始帮助文本。

原始帮助文本以*text*形式返回，格式以*format*形式返回。格式为字符串，可以是*"texinfo"*、*"html"*或*"plain text"*之一。

**参见：**get\_help\_text\_from\_file。

: *[text, format] =* **get\_help\_text\_from\_file** *(fname)*

从文件名返回原始帮助文本。

原始帮助文本以*text*形式返回，格式以*format*形式返回。格式为字符串，可以是*"texinfo"*、*"html"*或*"plain text"*之一。

**参见：**get\_help\_text。

: *text =* **get\_first\_help\_sentence** *(name)*

: *text =* **get\_first\_help\_sentence** *(name, max\_len)*

: *[text, status] =* **get\_first\_help\_sentence** *(…)*

返回函数帮助文本的第一句话。

第一句话定义为函数声明之后的文本，直到第一个句点(“.”)或第一次出现两个连续的换行符(“\n\n”)。文本被截断为*max\_len*的最大长度，默认为80。如果必须截断文本，则将文本的最后三个字符替换为”…”，以表明有更多的文本可用。

可选的输出参数*status*返回由*makeinfo*报告的状态。如果只请求一个输出参数，并且*status*为非零，则显示警告。

作为一个例子，这个帮助文本的第一句话是

get\_first\_help\_sentence ("get\_first\_help\_sentence")

-| ans = Return the first sentence of a function's help text.

**2.4命令行编辑**

Octave使用GNU Readline库提供了一套广泛的命令行编辑和历史特性。本手册只描述最常见的特性。此外，所有的编辑功能都可以根据用户的判断绑定到不同的按键上。本手册假设默认的Emacs绑定没有变化。有关自定义Readline的更多信息和完整的特性列表，请参阅GNU Readline Library手册。

要插入打印字符(字母、数字、符号等)，只需键入该字符。Octave将在光标处插入字符并向前移动光标。

许多命令行编辑功能使用控制字符进行操作。例如，字符*Control-a*将光标移动到行首。如果需要输入*C-a*，请先按*CTRL*，再按*a*。以下章节中，*control -a*等控制字符写成*C-a*。

另一组命令行编辑函数使用元字符。若要输入*M-u*，请按住META键并按下*u*。根据键盘的不同，META键可能被标记为ALT甚至WINDOWS。如果您的终端没有META键，您仍然可以使用以*ESC*开头的双字符序列键入META字符。因此，要输入M-u，您可以键入ESC u。在具有真正Meta键的终端上也允许使用*ESC*字符序列。以下章节将*Meta-u*等Meta字符写成*M-u*。

**2.4.1移动光标**

以下命令允许您操作光标。

*C-b*

光标向左移一字符位置

*C-f*

光标向前移一字符位置。

BACKSPACE

删除光标左侧的字符。

DEL

删除光标下方的字符。

*C-d*

删除光标下方的字符。

*M-f*

向前进一个字。

*M-b*

向后移动一个单词。

*C-a*

移动至行首。

*C-e*

移动至行尾

*C-l*

清除屏幕，在顶部重新打印当前行。

*C-\_*

*C-/*

撤消最后一个操作。您可以撤消所有操作，直到返回空行。

*M-r*

撤消对这一行所做的所有更改。这就像多次输入“undo”命令回到开始一样。

上表描述了编辑输入行所需的最基本的可能击键。在大多数终端上，您还可以使用左右方向键来代替*C-f*和*C-b*来向前和向后移动。

注意*C-f*是如何将一个字符向前移动，而*M-f*是如何将一个单词向前移动的。这是一个松散的约定，控制击键操作字符，而元击键操作单词。

函数*clc*将允许您从Octave程序中清除屏幕。

: **clc** *()*

: **home** *()*

清空终端屏幕，将光标移至左上角。

编程注意:home是clc的别名，可以互换使用。

**2.4.2 Killing（剪切）和Yanking（粘贴）**

killing文本是指将文本从行中删除，但保存起来以备以后使用，通常是通过将其Yanking行中。如果命令的描述说它`杀死`文本，那么你可以确定以后可以在不同(或相同)的地方获取文本。

下面是删除文本的命令列表。

*C-k*

终止从当前光标位置到行尾的文本。

*M-d*

Kill从光标到当前单词的末尾，如果在单词之间，则到下一个单词的末尾。

*M-DEL*

Kill从光标处开始到前一个单词的开头，或者如果在单词之间，到前一个单词的开头。

*C-w*

从光标到前一个空格处终止。这与*M-DEL*不同，因为单词边界不同。

还有，这里是如何把文本拉回一行。Yanking是指从终止缓冲区中复制最近终止的文本。

*C-y*

将最近终止的文本拉回光标处的缓冲区。

*M-y*

旋转止动环，猛拉新的顶部。只有当前面的命令是*C-y*或*M-y*时才能这样做。

当你使用kill命令时，文本会保存在*kill-ring*中。任意数量的连续击杀将所有击杀的文本保存在一起，这样当你把它拉回来时，你就可以一扫而光。击杀环不是特定于线路的;你在之前输入的行中删除的文本可以在以后输入另一行时恢复。

**2.4.3文本修改命令**

以下命令可用于输入原本具有特殊含义的字符(例如，TAB, C-q等)，或用于快速纠正键入错误。

*C-q*

*C-v*

将键入的下一个字符逐字添加到该行。这是如何插入像C-q这样的东西。

*M-TAB*

插入制表符。

*C-t*

将光标前的字符向前拖动到光标处的字符上，同时也将光标向前移动。如果游标位于行尾，则将其前面的两个字符调换位置。

*M-t*

将光标后面的单词拖过光标前面的单词，同时将光标移动到该单词上。

*M-u*

将光标后面的字符大写到当前(或后面)单词的末尾，将光标移动到单词的末尾。

*M-l*

将光标后面的字符小写到当前(或后面)单词的末尾，将光标移动到单词的末尾。

*M-c*

将光标后面的字符大写(如果光标位于单词之间，则将下一个单词的开头大写)，将光标移动到单词的末尾。

**2.4.4为您设置Readline类型**

下面的命令允许Octave为您完成命令和变量名。

TAB

尝试在光标之前的文本上完成补全。Octave可以完成命令和变量的名称。

*M-?*

列出光标前文本可能的补全。

: *val =* **completion\_append\_char** *()*

: *old\_val =* **completion\_append\_char** *(new\_val)*

: *old\_val =* **completion\_append\_char** *(new\_val, "local")*

查询或设置添加到成功的命令行补全尝试中的内部字符变量。

默认值是" "(一个空格)。

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。

: *completion\_list =* **completion\_matches** *("hint")*

根据字符序列提示为Octave生成可能的单词补全。

这个函数是为Emacs之类的程序提供的，这些程序可能会控制Octave并处理用户输入。例如:

completion\_matches ("sine")

⇒

sinetone

sinewave

编程简注:当调用此函数时，Octave中的当前命令数不会增加。这是一个特性，而不是一个bug。

**2.4.5历史操作命令**

Octave通常会跟踪您键入的命令，以便您可以回忆以前的命令来编辑或再次执行它们。当您退出Octave时，您最近键入的命令(不超过由变量*history\_size*指定的数量)将保存在一个文件中。当Octave启动时，它从变量*history\_file*命名的文件中加载一个初始命令列表。

下面是简单浏览和搜索历史列表的命令。

*LFD*

*RET*

不管光标在哪里，都接受当前行。如果该行不为空，则将其添加到历史列表中。如果该行是历史行，则将历史行恢复到其原始状态。

*C-p*

在历史列表中向上移动。

*C-n*

在历史列表中向下移动。

*M-<*

移动到历史的第一行。

*M->*

移动到输入历史记录的末尾，也就是您要输入的行!

*C-r*

从当前行开始向后搜索，并根据需要在历史中“向上”移动。这是一个增量搜索。

*C-s*

从当前线开始向前搜索，并在必要时向下移动历史。

在大多数终端上，还可以使用向上和向下箭头键来代替*C-p*和*C-n*来浏览历史记录列表。

除了用于在历史列表中移动的键盘命令外，Octave还提供了三个功能，用于查看、编辑和重新运行历史列表中的命令块。

: **history**

: **history** *opt1 …*

: *H =* **history** *()*

: *H =* **history** *(opt1, …)*

如果调用时不带参数，*history*将显示您已执行的命令列表。

有效的选项有:

*n*

*-n*

只显示最近的*n*行历史记录。

*-c*

清除历史记录列表。

*-q*

不要给显示的历史行编号。这对于使用X窗口系统剪切和粘贴命令非常有用。

*-r file*

读取文件*file*，将其内容附加到当前历史列表中。如果省略名称，则使用默认的历史文件(通常为*~/.octave\_hist*)。

*-w file*

将当前历史记录写入文件*file*。如果省略名称，则使用默认的历史文件(通常为*~/.octave\_hist*)。

例如，要显示您最近键入的五个命令，而不显示行号，请使用命令*history -q 5*。

如果使用单个输出参数调用，历史记录将作为单元格字符串保存到该参数中，而不会输出到屏幕。

**参见：**edit\_history, run\_history。

: **edit\_history**

: **edit\_history** *cmd\_number*

: **edit\_history** *first last*

使用由变量*editor*命名的编辑器编辑历史列表。

要编辑的命令首先被复制到临时文件中。当您退出编辑器时，Octave将执行文件中保留的命令。使用*edit\_history*来定义函数通常比直接在命令行中输入更方便。一旦退出编辑器，就会执行命令块。要避免执行任何命令，只需在离开编辑器之前从缓冲区中删除所有行即可。

当不带参数调用时，编辑先前执行的命令;使用一个参数编辑指定的命令*cmd\_number*;使用两个参数编辑*first*和*last*之间的命令列表。命令号说明符也可以是负数，其中-1表示最近执行的命令。下面是等价的，并编辑最近执行的命令。

edit\_history

edit\_history -1

在使用范围时，为第一个命令指定一个比最后一个命令更大的数字将反转命令列表，然后将它们放入要编辑的缓冲区中。

**参见：**run\_history, history。

: **run\_history**

: **run\_history** *cmd\_number*

: **run\_history** *first last*

从历史列表中运行命令。

当不带参数调用时，运行先前执行的命令;

使用一个参数，运行指定的命令*cmd\_number*;

使用两个参数，运行*first*和*last*之间的命令列表。命令号说明符也可以是负数，其中-1表示最近执行的命令。例如:命令

run\_history

OR

run\_history -1

再次执行最近的命令。命令

run\_history 13 169

执行命令13到169。

为第一个命令指定一个比最后一个命令更大的数字，将在执行命令之前反转命令列表。例如:

disp (1)

disp (2)

run\_history -1 -2

⇒

2

1

**参见：**edit\_history, history。

Octave还允许您自定义历史记录保存的时间、地点和方式的详细信息。

: *val* *=* **history\_save** *()*

: *old\_val =* **history\_save** *(new\_val)*

: *old\_val =* **history\_save** *(new\_val, "local")*

查询或设置控制命令行上输入的命令是否保存在历史文件中的内部变量。

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。

**参见：**history\_control, history\_file, history\_size, history\_timestamp\_format\_string。

: *val =* **history\_control** *()*

: *old\_val =* **history\_control** *(new\_val)*

查询或设置指定命令如何保存到历史列表的内部变量。

默认值是一个空字符串，但可以被环境变量*OCTAVE\_HISTCONTROL*覆盖。

*history\_control*的值是一个以冒号分隔的值列表，这些值控制命令如何保存在历史列表中。如果值列表包含*ignorespace*，则以空格字符开头的行不会保存在历史列表中。*ignoredup*的值将导致不保存与前一个历史记录条目匹配的行。*ignoreboth*的值是*ignorespace*和*ignoredups*的简写。*erasedups*的值将导致在保存当前行之前从历史列表中删除与当前行匹配的所有先前行。任何不在上面列表中的值都将被忽略。如果*history\_control*为空字符串，则根据*history\_save*的值将所有命令保存在历史列表中。

**参见：**history\_file, history\_size, history\_timestamp\_format\_string, history\_save。

: *val =* **history\_file** *()*

: *old\_val =* **history\_file** *(new\_val)*

查询或设置用于保存命令历史记录的文件名的内部变量。

在当前Octave会话期间发出的所有命令都将被写入这个新文件(如果当前的*history\_save*设置允许这样做)。

默认值是*$DATA/octave/history*，其中*$DATA*是(漫游)用户数据文件的平台特定位置(例如，*$XDG\_DATA\_HOME*，或者，如果没有设置，~/.local/share(类unix操作系统)或%APPDATA% (Windows操作系统)。默认值可以被环境变量*OCTAVE\_HISTFILE*覆盖。

编程简注:

如果希望永久更改Octave历史文件的位置，则需要在每个新的Octave会话中发出*history\_file*命令。这可以通过使用Octave的*.octaverc*启动文件来实现。

如果您还想从这个不同的历史文件中读取过去Octave会话的保存的历史命令，那么您需要在设置历史文件的新值之后使用额外的命令*history -r*。Octave启动文件中的示例代码可能如下所示:

history\_file ("~/new/.octave\_hist");

if (exist (history\_file ()))

history ("-r", history\_file());

endif

**参见：**history、history\_control、history\_save、history\_size、history\_timestamp\_format\_string。

: *val =* **history\_size** *()*

: *old\_val =* **history\_size** *(new\_val)*

查询或设置指定要在历史文件中存储多少条目的内部变量。

默认值是*1000*，但是可以被环境变量*OCTAVE\_HISTSIZE*覆盖。

**参见：**history\_file, history\_timestamp\_format\_string, history\_save。

: *val =***history\_timestamp\_format\_string** *()*

: *old\_val =* **history\_timestamp\_format\_string** *(new\_val)*

: *old\_val =* **history\_timestamp\_format\_string** *(new\_val， "local")*

查询或设置内部变量，该变量指定在Octave退出时写入历史文件的注释行的格式字符串。

格式字符串被传递给*strftime*。默认值为

"# Octave VERSION, %a %b %d %H:%M:%S %Y %Z <USER@HOST>"

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：**strftime, history\_file, history\_size, history\_save。

: *val =* **EDITOR** *()*

: *old\_val =* **EDITOR** *(new\_val)*

: *old\_val =* **EDITOR** *(new\_val, "local")*

查询或设置指定默认文本编辑器的内部变量。

默认值是在Octave启动时从环境变量*EDITOR*中获取的。如果环境变量没有初始化，*EDITOR*将被设置为*”emacs”*。

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：**edit, edit\_history。

**2.4.6自定义readline**

Octave使用GNU Readline库进行命令行编辑和历史特性。Readline非常灵活，可以通过命令的配置文件进行修改(请参阅GNU Readline库了解确切的命令语法)。默认的配置文件通常是*~/.inputrc*。

Octave提供了两个命令来初始化Readline，从而改变命令行行为。

: **readline\_read\_init\_file** *()*

: **readline\_read\_init\_file** *(file)*

读取readline库初始化文件*file*。

如果省略*file*，则读取默认初始化文件(通常为*~/.inputrc*)。

详细信息请参见GNU Readline Library中的Readline Init File。

**参见：**readline\_re\_read\_init\_file。

: **readline\_re\_read\_init\_file** *()*

重新读取上次读取的readline库初始化文件。

详细信息请参见GNU Readline库中的Readline Init File。

**参见：**readline\_read\_init\_file。

**2.4.7自定义提示信息**

以下变量可用于自定义命令行提示符的外观。Octave允许通过插入一些反斜杠转义的特殊字符来定制提示符，这些字符解码如下:

‘\t’

时间

‘\d’

日期

‘\n’

通过打印相当于回车和换行符的内容来开始新行。

‘\s’

程序的名称(通常只是*'octave'*)。

‘\w’

当前工作目录。

‘\W’

当前工作目录的基本名称。

‘\u’

当前用户的用户名。

‘\h’

主机名，直到第一个' . '。

‘\H’

主机名

‘\#’

该命令的命令号，从Octave开始计数。

‘\!’

该命令的历史编号。这与*'\#'*的不同之处在于，当Octave启动时，历史列表中的命令数量。

‘\$’

如果有效UID为0，则为*’#’*，否则为*’$’*。

‘\nnn’

八进制字符码为*nnn*的字符。

‘\\’

反斜杠键

: *val =* **PS1** *()*

: *old\_val =* **PS1** *(new\_val)*

: *old\_val =* **PS1** *(new\_val, "local")*

查询或设置主提示字符串。

当交互执行时，Octave在准备读取命令时显示主提示符。

主提示字符串的默认值是*'octave:\#>'*。要更改它，请使用如下命令

PS1 ('\u@\H> ')

这将导致用户 *'boris'* 登录主机 *'kremvax.kgb.su'* 的提示 *'boris@kremvax>'* 。注意，在双引号字符串中输入反斜杠需要两个反斜杠。看到字符串。

如果终端支持ANSI转义序列，也可以使用它们。这对于给提示上色很有用。例如,

PS1 ('\[\033[01;31m\]\s:\#> \[\033[0m\]')

将给默认的Octave提示显示红色。

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：**PS2, PS4。

: *val =* **PS2** *()*

: *old\_val =* **PS2** *(new\_val)*

: *old\_val =* **PS2** *(new\_val, "local")*

查询或设置辅助提示字符串。

当Octave需要额外的输入来完成命令时，将打印次要提示符。例如，如果您正在键入跨越几行的*for*循环，Octave将在第一行之后的每行开头打印第二个提示符。辅助提示字符串的默认值是*”>”*。

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：**PS1, PS4。

: *val =* **PS4** *()*

: *old\_val =* **PS4** *(new\_val)*

: *old\_val =* **PS4** *(new\_val, "local")*

查询或设置使能回显命令时，输出的前缀字符串。

默认值为*”+ ”*。有关回显命令的描述，请参阅日记和回显命令。

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：**echo, PS1, PS2。

**2.4.8日志和Echo命令**

Octave的日记功能允许您通过记录您键入的输入和Octave在单独的文件中产生的输出来保留所有或部分交互会话的日志。

: **diary**

: **diary** *on*

: **diary** *off*

: **diary** *filename*

: *[status, diaryfile] =* **diary**

记录所有命令和它们产生的输出的列表，混合在一起，就像它们出现在终端上一样。

有效的选项有:

*On*

在当前工作目录下的一个名为diary的文件中开始记录会话。

*Off*

停止在日志文件中记录会话。

*Filename*

将会话记录在文件名中。

在没有输入或输出参数的情况下，*diary*切换当前日志状态。

如果请求输出参数，则*diary*忽略输入并返回当前状态。布尔状态指示记录是打开还是关闭，*diaryfile*是存储会话的文件的名称。

**参见：**history, evalc。

有时在函数或脚本中查看正在执行的命令是很有用的。这对于调试某些类型的问题特别有帮助。

: **echo**

: **echo** *on*

: **echo** *off*

: **echo** *on all*

: **echo** *off all*

: **echo** *function on*

: **echo** *function off*

控制命令在执行时是否显示。

有效的选项有:

*on*

启用命令在脚本文件中执行时的回显。

*off*

禁用命令在脚本文件中执行时的回显。

*on all*

启用命令在脚本文件和函数中执行时的回显。

*off all*

禁用命令在脚本文件和函数中执行时的回显。

*function on*

启用命令在指定函数中执行时的回显。

*function off*

禁用命令在指定函数中执行时的回显。

如果不带参数，*echo*将切换当前的echo状态。

**参见：**PS4。

**2.5 Octave如何报告错误**

对于无效的程序，Octave报告两种错误。

如果Octave无法理解您键入的内容，则会发生语法错误。例如，如果你拼错了关键字，

octave:13> function z = f (x, y) z = x ||| 2; endfunction

Octave将立即响应如下消息:

parse error:

syntax error

>>> function z = f (x, y) z = x ||| y; endfunction

对于大多数解析错误，Octave使用插入符号( *'^'* )来标记行中无法理解输入的点。在这种情况下，Octave生成一条错误消息，因为逻辑或操作符( || )的关键字拼写错误。它在第三个 ’|’处标记了错误，因为在此之前的代码是正确的，但最后的 ’|’ 不被理解。

另一类错误消息发生在计算时。这些错误称为运行时错误，有时也称为求值错误，因为它们发生在程序运行或求值时。例如，如果在纠正了前面函数定义中的错误后，键入

octave:13> f ()

octave会响应

error: `x' undefined near line 1 column 24

error: called from:

error: f at line 1, column 22

这个错误消息有几个部分，并提供了相当多的信息来帮助您定位错误的来源。消息从最内层的错误点生成，并提供封闭表达式和函数调用的回溯。

在上面的示例中，第一行表示在某个函数或表达式的第1行和第24列附近发现名为 *’x’* 的变量未定义。对于函数中发生的错误，从包含函数定义的文件的开头开始计算行数。对于在封闭函数之外发生的错误，行号表示输入行号，通常显示在主提示字符串中。

错误消息的第二行和第三行表明错误发生在函数*f*中。如果函数*f*是从另一个函数(例如，*g*)中调用的，则错误列表将以多一行结束:

error: g at line 1, column 17

这些函数调用列表可以很容易地跟踪程序在错误发生之前的路径，并在再次尝试之前纠正错误。

**2.6可执行的Octave程序**

一旦你学会了Octave，你可能想要使用 ’*#!*’ 脚本机制。您可以在GNU系统和许多Unix系统上这样做。

当您想要编写一个用户可以调用而不知道该程序是用Octave语言编写的程序时，自包含的Octave脚本非常有用。Octave脚本也用于数据文件的批处理。一旦在Octave的交互部分中开发并测试了算法，就可以将其提交到可执行脚本中，并在新的数据文件上反复使用。

作为可执行Octave脚本的一个简单示例，您可以创建一个名为*hello*的文本文件，包含以下行:

#! octave-interpreter-name -qf

# a sample Octave program

printf ("Hello, world!\n");

(其中*Octave -interpreter-name*应替换为Octave二进制文件的完整路径和名称)。注意，这只会在*' #!'*出现在文件的最开头。在使文件可执行之后(在Unix系统上使用*chmod*命令)，你可以简单地输入:

hello

在shell中，系统会安排运行Octave，就好像你输入了:

octave hello

以 ’#!’ 开头的行'列出要运行的解释器的完整路径和文件名，以及要传递给该解释器的可选初始命令行参数。然后，操作系统使用给定的参数和所执行程序的完整参数列表运行解释器。列表中的第一个参数是Octave可执行文件的完整文件名。参数列表的其余部分要么是Octave的选项，要么是数据文件，或者两者都是。*' -qf* '选项通常在独立的Octave程序中指定，以防止它们打印正常的启动消息，并防止它们根据特定用户的内容而表现不同 *~/.octaverc*文件。参见从命令行调用Octave。

请注意，某些操作系统可能会限制 *' #! '* 之后可识别的字符数量。此外，出现在 *’#!’* 中的参数行被不同的shell /系统解析为不同的。它们中的大多数将所有参数组合在一个字符串中，并将其作为单个参数传递给解释器。在本例中，使用以下脚本:

#! octave-interpreter-name -q -f # comment

相当于在命令行输入:

octave "-q -f # comment"

这将产生一条错误消息。不幸的是，Octave无法确定它是从命令行还是从*’ #! ’*脚本，所以在使用 *' #!’* 的机制。

**2.6.1向可执行脚本传递参数**

注意，当Octave从可执行脚本启动时，内置函数argv返回一个单元格数组，其中包含传递给可执行Octave脚本的命令行参数，而不是传递给Octave解释器的' #!’的台词。例如，下面的程序将重现用于执行脚本的命令行，而不是*' -qf '*。

#! /bin/octave -qf

printf ("%s", program\_name ());

arg\_list = argv ();

for i = 1:nargin

printf (" %s", arg\_list{i});

endfor

printf ("\n");

**2.6.2双用途可执行脚本和Octave函数**

要编写m-file，使其在从shell调用时充当可执行程序，或在Octave中调用时充当普通函数，请使用由*argv*函数初始化的默认输入参数。

如果从shell调用函数，Octave将不会向函数传递任何输入参数，因此使用默认实参。但是当从解释器调用函数时，任何参数都会传递给函数，并且这些参数会覆盖默认值。

此外，该文件必须以扩展名*.m*结尾，以便解释器将其识别为Octave函数。最后，*argv*的输出是一个字符串单元数组。在处理之前，可能需要使用*str2double*或*str2num*将其转换为数值。

作为一个完整的示例，请考虑位于文件*mysin.m*中的以下代码。

#! /bin/octave -qf

function retval = mysin (x = str2double (argv(){end}))

retval = sin (x)

endfunction

可以从shell调用

mysin.m 1.5

或源自Octave with

mysin (1.5)

**2.7 Octave程序中的注释**

注释是为了人类读者而包含在程序中的一些文本，它不是程序的可执行部分。注释可以解释程序的功能和工作原理。几乎所有的编程语言都有注释的规定，因为没有注释程序通常很难理解。

**2.7.1单行注释**

在Octave语言中，注释以尖锐的符号字符 *’# ’* 或百分比符号 ’%’ 开始，并持续到行尾。任何紧跟着尖锐符号或百分号的文本都将被Octave解释器忽略而不执行。下面的示例显示了整行和部分行注释。

function countdown

# Count down for main rocket engines

disp (3);

disp (2);

disp (1);

disp ("Blast Off!"); # Rocket leaves pad

endfunction

**2.7.2块注释**

可以通过在匹配的 *' #{'* 和 *' #}'* 或 *' %{'* 和 *'%}'* 标记之间包含代码来注释整个代码块。例如,

function quick\_countdown

# Count down for main rocket engines

disp (3);

#{

disp (2);

disp (1);

#}

disp ("Blast Off!"); # Rocket leaves pad

endfunction

会产生一个非常快的倒计时，从 *’3’* 到 *’Blast Off’* ，因为 ’disp(2);’ 和 ‘disp(1);’ 不会执行。

块注释标记必须作为一行中唯一的字符单独出现(空格除外)，以便正确解析。

**2.7.3注释和帮助系统**

*help*命令(参见获取帮助的命令)能够在函数中找到第一个注释块，并将其作为文档字符串返回。这意味着用于获取内置函数帮助的相同命令也可用于正确格式化的用户定义函数。例如，定义下面的函数*f*后，

function xdot = f (x, t)

# usage: f (x, t)

#

# This function defines the right-hand

# side functions for a set of nonlinear

# differential equations.

r = 0.25;

…

endfunction

命令help f产生如下输出

usage: f (x, t)

This function defines the right-hand

side functions for a set of nonlinear

differential equations.

尽管可以将注释行放入由键盘组成的一次性Octave程序中，但它通常不是很有用，因为注释的目的是帮助您或其他人稍后理解程序。

*help*解析器目前只识别单行注释(参见单行注释)，而不识别初始帮助文本的块注释。

**3 数据类型**

Octave的所有版本都包含许多内置数据类型，包括实数和复杂的标量和矩阵、字符串、数据结构类型以及可以包含所有数据类型的数组。

还可以通过编写少量c++代码来定义新的专用数据类型。在某些系统上，新的数据类型可以在Octave运行时动态加载，所以没有必要为了添加一个新类型而重新编译所有的Octave。有关Octave动态链接功能的更多信息，请参阅外部代码接口。用户定义的数据类型描述了为Octave定义新数据类型必须做的事情。

: *typestr =* **typeinfo** *(expr)*

: *cstr =* **typeinfo** *()*

以字符串形式返回表达式*expr*的类型。

如果省略*expr*，则返回包含所有当前安装的数据类型的字符串的单元格数组。

**参见：**class, isa。

**3.1内置数据类型**

标准的内置数据类型是实数和复杂的标量和矩阵、范围、字符串、数据结构类型和单元格数组。在未来的版本中可能会添加其他内置数据类型。如果您需要一个目前没有作为内置类型提供的专用数据类型，我们鼓励您编写自己的用户定义数据类型，并将其贡献给Octave的未来版本。

变量的数据类型可以通过使用以下函数来确定和更改。

: *classname =* **class** *(obj)*

: *cls =* **class** *(s, classname)*

: *cls =* **class** *(s, classname, parent1, …)*

返回对象*obj*的类，或者使用结构*s*和name (string) *classname*中的字段创建一个类。

附加参数指定新类派生的父类列表。

**参见：**typeinfo, isa。

: *tf =* **isa** *(obj, classname)*

如果*obj*是类*classname*的对象，则返回true。

*Classname*也可以是以下类类别之一:

*"float"*

由“双精度”和“单精度”类组成的浮点值。

*"integer"*

整数值，包含(u)int8， (u)int16， (u)int32， (u)int64类。

*"numeric"*

由浮点数或整数值组成的数值。

如果*classname*是字符串的单元格数组，则返回一个相同大小的逻辑数组，对于*obj*所属的每个类都包含true。

**参见：**class, typeinfo。

: *y =* **cast** *(x, "type")*

: *y =* **cast** *(x, "like", var)*

将*x*转换为数据类型*type*。

输入*x*可以是可转换为目标类的类的标量、向量或矩阵(见下文)。

如果在*"like"*后面指定变量*var*，则*x*被转换为相同的数据类型和稀疏性属性。如果var是复数，*x*也是复数。

Var可以是，type可以命名以下任何一个内置数值类:

"double"

"single"

"logical"

"char"

"int8"

"int16"

"int32"

"int64"

"uint8"

"uint16"

"uint32"

"uint64"

可以修改值*x*以适应新类型的范围。

例子:

cast (-5, "uint8")

⇒ 0

cast (300, "int8")

⇒ 127

编程简注:这个函数依赖于对象*x*有一个名为*type*的转换方法。用户定义的类可能只实现上面所示的全部类型列表的一个子集。在这种情况下，可能需要调用两次强制类型转换才能获得所需的类型。例如，到double类型的转换几乎总是实现的，但到uint8类型的转换可能没有实现。在这种情况下，下面的代码将工作:

cast (cast (user\_defined\_val, "double"), "uint8")

**参见：**typecast, int8, uint8, int16, uint16, int32, uint32, int64, uint64, double, single, logical, char, class, typeinfo。

: *y =* **typecast** *(x, "class")*

返回一个新数组*y*，将内存中*x*的数据解释为数字类*class*的数据。

类*x*和*class*都必须是内置的数字类之一:

"logical"

"char"

"int8"

"int16"

"int32"

"int64"

"uint8"

"uint16"

"uint32"

"uint64"

"double"

"single"

"double complex"

"single complex"

后两个词只与*class*连用;它们表明请求的是复值结果。复杂数组以连续实数对的形式存储在内存中。整数类型的大小由其位计数给出。*logical*和*char*通常都是一个字节宽;然而，c++并不能保证这一点。如果您的系统符合IEEE标准，则single和double将分别为4字节和8字节宽。*”logical”* 在课堂上是不允许的。

如果输入是行向量，则返回值是行向量，否则返回值是列向量。

如果*x*的位长度不能被*class*的位长度整除，则会发生错误。

在小端机器上使用类型转换的一个示例是

x = uint16 ([1, 65535]);

typecast (x, "uint8")

⇒ [ 1, 0, 255, 255]

**参见：**cast, bitpack, bitunpack, swapbytes。

: *y =* **swapbytes** *(x)*

交换值的字节顺序，从小端序转换为大端序，反之亦然。

例如:

swapbytes (uint16 (1:4))

⇒ 256 512 768 1024

参见:typecast, cast。

: *y =* **bitpack** *(x, class)*

返回一个新数组*y*，将逻辑数组*x*解释为数字类*class*的数据的原始位模式。

类必须是一个内置的数字类:

"double"

"single"

"double complex"

"single complex"

"char"

"int8"

"int16"

"int32"

"int64"

"uint8"

"uint16"

"uint32"

"uint64"

*x*的元素个数应该能被*class*的位长度整除。如果不是，多余的位将被丢弃。比特按重要性递增的顺序出现，例如，*x(1)*是位0,*x(2)*是位1，等等。

如果*x*是行向量，结果是行向量，否则是列向量。

**参见：**bitunpack, typecast。

: *y =* **bitunpack** *(x)*

返回与*x*的原始位模式对应的逻辑数组*y*。

*X*必须属于一个内置的数字类:

"double"

"single"

"char"

"int8"

"int16"

"int32"

"int64"

"uint8"

"uint16"

"uint32"

"uint64"

如果x是行向量，结果就是行向量;否则，它就是一个列向量。

**参见：**bitpack, typecast。

**3.1.1数字对象**

Octave的内置数值对象包括实数、复数和整数标量和矩阵。所有内置的浮点数字数据目前都存储为双精度数字。在使用IEEE浮点格式的系统上，可以存储大约在2.2251e-308到1.7977e+308范围内的值，相对精度大约为2.2204e-16。确切的值分别由变量*realmin*、*realmax*和*eps*给出。

矩阵对象可以是任何大小，并且可以动态地重塑和调整大小。使用各种强大的索引特性，很容易提取单个行、列或子矩阵。参见索引表达式。

有关详细信息，请参阅数字数据类型。

**3.1.2数据丢失**

可以使用*NA*(“Not Available”的缩写)在Octave中显式地表示缺失的数据。只有当数据表示为浮点数时，才能表示缺失的数据。在这种情况下，缺失的数据被表示为*NaN*表示的一种特殊情况。

: *val =* ***NA***

: *val =* ***NA*** *(n)*

: *val =* ***NA*** *(n, m)*

: *val =* ***NA*** *(n, m, k, …)*

: *val =* ***NA*** *(…, "like", var)*

: *val =* ***NA*** *(…, class)*

返回一个标量、矩阵或n维数组，其元素都等于用于指定缺失值的特殊常量。

注意，NA总是比较不等于NA (NA != NA)。要查找NA值，请使用*isna*函数。

当不带参数调用时，返回值为*' NA '*的标量。

当使用单个参数调用时，返回具有指定维度的方阵。

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

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

可选参数*class*指定返回类型，可以是*"double"*或*"single"*。

**参见：**isna。

: *tf =* **isna** *(x)*

返回一个逻辑数组，当*x*的元素是NA(缺失)值时为true，当元素不是NA值时为false。

例如:

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

⇒ [ 0, 0, 1, 0 ]

**参见：**isnan, isinf, isfinite。

**3.1.3字符串对象**

Octave中的字符串由用双引号或单引号括起来的字符序列组成。在内部，Octave目前将字符串存储为字符矩阵。所有适用于矩阵对象的索引操作也适用于字符串。

有关更多信息，请参阅字符串。

**3.1.4数据结构对象**

Octave的数据结构类型可以帮助你组织不同类型的相关对象。当前的实现使用关联数组，索引仅限于字符串，但语法更像c风格的结构。

有关更多信息，请参阅结构。

**3.1.5 Cell Array对象**

Octave中的Cell Array是一种通用的数组，可以保存任意数量的不同数据类型。

有关更多信息，请参阅单元格数组。

**3.2自定义数据类型**

有一天，我希望扩展它，包括对Octave管理用户定义数据类型机制的完整描述。在此之前，您将不得不通过阅读Octave的*src*目录中的*ov.h*、*ops.h*和相关文件中的代码来解决这个问题。

**3.3对象大小**

下面的函数允许您确定变量或表达式的大小。这些函数是为所有对象定义的。当操作没有意义时，返回−1。例如，Octave的数据结构类型没有行或列，因此*rows*和*columns*函数对结构参数返回−1。

: *n =* **ndims** *(A)*

返回A的维数。

对于任何数组，结果总是大于或等于2。不计算拖尾单维，即拖尾维*d*大于2且*size(A, d) = 1*。

ndims (ones (4, 1, 2, 1))

⇒ 3

**参见:**大小。

: *nc =* **columns** *(A)*

返回*A*的列数。

这相当于*size (A, 2)*。

**参见：**行，大小，长度，数字，isscalar, isvector, ismatrix。

: *nr =* **rows** *(A)*

返回*A*的行数。

这相当于*size (A, 1)*

**参见：**列，大小，长度，数字，isscalar, isvector, ismatrix。

: *n =* **numel** *(A)*

: *n =* **numel** *(A, idx1, idx2, …)*

返回对象*A*中的元素个数。

可选地，如果提供了索引idx1、idx2、…，则返回由索引产生的元素数量

A(idx1, idx2, …)

注意，索引不一定是标量数。例如,

a = 1;

b = ones (2, 3);

numel (a, b)

将返回6，因为这是使用b进行索引的方法数。或者索引可以是字符串":"，它表示冒号操作符。例如,

A = ones (5, 3);

numel (A, 2, ":")

将返回3，因为第二行有三个列项。

当对象以cs-list索引的左值出现时，也会调用此方法，即object{…}或object(…).field。

**参见：**尺寸，长度，长度。

: *n =* **length** *(A)*

返回对象*A*的长度。

空对象的长度为0，标量的长度为1，向量的元素个数为1。对于矩阵或n维对象，长度是沿最大维度的元素数(相当于*max (size (A))*)。

**参见：**nummel, size。

: *sz =* **size** *(A)*

: *dim\_sz =* **size** *(A, dim)*

: *dim\_sz =* **size** *(A, d1, d2, …)*

: *[rows, cols, …, dim\_N\_sz] =* **size** *(…)*

返回一个行向量，其中包含对象A的每个维度的大小(元素数)。

当给定第二个参数*dim*时，返回相应维度的大小。如果dim是一个向量，则返回每个相应的维度。还可以将多个维度指定为单独的参数。

对于单个输出参数，*size*返回一个行向量。当使用多个输出参数调用*size*时，*size*返回第N个参数中维度N的大小。行数(维度1)在第一个参数中返回，列数(维度2)在第二个参数中返回，以此类推。如果A中的维度多于输出参数，则*size*返回最终输出参数中剩余维度中元素的总数。如果请求的维度dim大于A中的维度数，则size返回1(而不是0)。

例1:单行矢量输出

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

⇒ [ 3, 2 ]

示例2:第二维(列)中的元素数量

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

⇒ 2

示例3:输出参数的数量==维度的数量

[nr, nc] = size ([1, 2; 3, 4; 5, 6])

⇒ nr = 3

⇒ nc = 2

示例4:输出参数的数量<维度的数量

[nr, remainder] = size (ones (2, 3, 4, 5))

⇒ nr = 2

⇒ remainder = 60

例5:维度中的元素数>实际维度数

sz4 = size (ones (2,3)， 4)

⇒sz4 = 1

**参见：**nummel, ndims, length, rows, columns, size\_equal, common\_size。

: *tf =* **isempty** *(A)*

如果A是空对象(它的任何一个维度都为零)，则返回true。

**参见：**isnull, isa。

: *tf =* **isnull** *(x)*

如果x是一个特殊的空数组、字符串或单引号字符串，则返回true。

在右侧带有这样一个空值的索引赋值应该删除数组元素。当重载用户定义类的索引赋值方法(*subsassign*)时，使用此函数代替*isempty*。Isnull用于区分这两种情况:

A(I) = []

和

X = []; A(I) = X

在第一次赋值中，右侧是[]，这是一个特殊的空值。只要索引I不为空，这段代码应该从A中删除元素，而不是执行赋值。

在第二个赋值中，右边是空的(因为X是[])，但它不是空的。这段代码应该将空值赋给A中的元素。

Octave内置char类中的一个示例演示了正确使用isnull时解释器的行为。

str = "Hello World";

nm = "Wally";

str(7:end) = nm # indexed assignment

⇒ str = Hello Wally

str(7:end) = "" # indexed deletion

⇒ str = Hello

参见:isempty, isindex。

: *sz =* **sizeof** *(val)*

返回*val*的大小，以字节为单位。

**参见：**whos。

: *TF =* **size\_equal** *(A, B)*

: *TF =* **size\_equal** *(A, B, …)*

如果所有参数的维度一致，则返回true。

后面的单例维度将被忽略。当带单个参数或不带参数调用*size\_equal*时，返回true。

**参见：**size, numel, ndims, common\_size。

: *B =* **squeeze** *(A)*

从*A*中删除单例维度并返回结果。

请注意，为了与MATLAB兼容，所有对象都至少具有两个维度，并且行向量保持不变。

**参见：**reshape。

**4数字数据类型**

数字常量可以是标量、矢量或矩阵，并且可以包含复数值。

数字常数的最简单形式是标量，它是一个数字。请注意，默认情况下，数值常量在Octave中以IEEE 754双精度(binary64)浮点格式表示(复杂常量存储为binary64值对)。但是，可以按照整数数据类型中描述的方式表示实整数。

如果数字常量是实整数，则可以用十进制、十六进制或二进制表示法定义它。十六进制表示法以' 0x '或' 0x '开头，二进制表示法以' 0b '或' 0b '开头，否则采用十进制表示法。因此，' 0b '不是一个十六进制数，事实上，它根本不是一个有效的数字。

为了更好的可读性，数字可以用下划线分隔符“\_”来分隔，这将被Octave解释器忽略。下面是一些实值整型常量的例子，它们都表示相同的值，并在内部存储为binary64:

42 # decimal notation

0x2A # hexadecimal notation

0b101010 # binary notation

0b10\_1010 # underscore notation

round (42.1) # also binary64

在十进制记数法中，数字常数可以表示为十进制分数，甚至可以表示为科学(指数)记数法。请注意，这对于十六进制或二进制表示法是不可能的。同样，在下面的示例中，所有数字常量表示相同的值:

.105

1.05e-1

.00105e+2

与大多数编程语言不同，复杂的数值常数表示为实部和虚部的和。虚部由一个实值数值常数表示，后面紧跟着一个复值指示器(' i '、' j '、' i '或' j '，它代表sqrt(-1))。数字常量和复数值指示符之间不允许有空格。一些表示相同值的复杂数值常量的示例:

3 + 42i

3 + 42j

3 + 42I

3 + 42J

3.0 + 42.0i

3.0 + 0x2Ai

3.0 + 0b10\_1010i

0.3e1 + 420e-1i

: *y =* **double** *(x)*

将*x*转换为双精度类型。

**参见：**single。

: *z =* **complex** *(x)*

: *z =* **complex** *(re, im)*

从实参数返回复数值。

使用一个实参数*x*，返回复数结果*x + 0i*。

使用2个实参数，返回复数结果*re + imi.complex*通常比*a + b\*i*这样的表达式更方便。例如:

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

⇒ [ 1 + 3i 2 + 4i ]

**参见：**real, image, iscomplex, abs, arg。

**4.1矩阵**

在Octave中定义一个值矩阵是很容易的。矩阵的大小是自动确定的，因此没有必要显式地说明尺寸。表达式

a = [1, 2; 3, 4]

矩阵中的结果

/ \

| 1 2 |

a = | |

| 3 4 |

\ /

矩阵的元素可以是任意表达式，只要在组合各个部分时维度都是有意义的。例如，给定上面的矩阵，表达式

[ a, a ]

生成矩阵

ans =

1 2 1 2

3 4 3 4

但是表达方式

[ a, 1 ]

产生错误

error: number of rows must match (1 != 2) near line 13, column 6

(当然，假设这个表达式是作为第13行第一行输入的)。

在分隔矩阵表达式的方括号内，Octave会查看周围的上下文，以确定空格和换行符是否应该转换为元素和行分隔符，或者干脆忽略，因此像

a = [ 1 2

3 4 ]

将工作。然而，一些可能的混淆来源仍然存在。例如，在表达式

[ 1 - 1 ]

' - '被视为二进制运算符，结果是标量0，但在表达式中

[ 1 -1 ]

' - '被视为一元运算符，结果是向量*[1，-1]*。类似地，表达式

[ sin (pi) ]

将被解析为

[ sin, (pi) ]

并且会导致错误，因为调用sin函数时没有参数。要解决这个问题，必须省略sin和左括号之间的空格，或者将表达式括在一组括号中:

[ (sin (pi)) ]

单引号字符(" '，用作转置操作符并用于分隔字符串)周围的空白也会引起混淆。给定a = 1，表达式

[ 1 a' ]

导致单引号字符被视为转置运算符，结果是向量[1,1]，但表达式

[ 1 a ' ]

产生错误消息

parse error:

syntax error

>>> [ 1 a ' ]

^

因为不这样做会在解析有效表达式时造成麻烦

[ a 'foo' ]

为了清晰起见，最好总是使用逗号和分号来分隔矩阵元素和行。

在编译Octave时，矩阵中元素的最大数量是固定的。允许的数量可以通过*sizemax*函数查询。请注意，其他因素(例如机器上可用的内存量)可能会将矩阵的最大大小限制在较小的范围内。

: *max\_numel =* **sizemax** *()*

返回数组大小所允许的最大值。

如果Octave使用64位索引编译，则结果为int64类，否则为int32类。最大数组大小略小于*intmax*报告的相关类允许的最大值。

**参见：**intmax。

当您键入矩阵或其值为矩阵的变量的名称时，Octave的响应是将矩阵打印出来，行和列排列整齐。如果矩阵的行太大而无法在屏幕上显示，Octave将分割矩阵并在每个部分之前显示标题，以指示要显示哪些列。您可以使用以下变量来控制输出的格式。

: *val =* **output\_precision** *()*

: *old\_val =* **output\_precision** *(new\_val)*

: *old\_val =* **output\_precision** *(new\_val, "local")*

查询或设置内部变量，该变量指定要为数字输出显示的有效数字的最小数目。

请注意，无论*output\_precision*的值设置如何，双精度值显示的精度位数限制为16，单精度值显示的精度位数限制为7。同样，调用*format*函数更改数值显示也可以更改*output\_precision*的设定值。

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：**format, fixed\_point\_format。

通过使用不同的*output\_precision*值，可以实现广泛的输出样式。可以使用*format*函数设置合理的组合。参见基本输入和输出。

: *val =* **split\_long\_rows** *()*

: *old\_val =* **split\_long\_rows** *(new\_val)*

: *old\_val =* **split\_long\_rows** *(new\_val, "local")*

查询或设置内部变量，该变量控制在显示到终端窗口时是否可以拆分矩阵的行。

如果行被分割，Octave将以一系列较小的片段显示矩阵，每个片段都可以在终端宽度的限制内适合，并且每个行集都被标记，以便您可以轻松地看到当前正在显示哪些列。例如:

octave:13> rand (2,10)

ans =

Columns 1 through 6:

0.75883 0.93290 0.40064 0.43818 0.94958 0.16467

0.75697 0.51942 0.40031 0.61784 0.92309 0.40201

Columns 7 through 10:

0.90174 0.11854 0.72313 0.73326

0.44672 0.94303 0.56564 0.82150

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：**格式。

当数值变得非常大或非常小时，Octave会自动切换到科学记数法。这保证了您将看到矩阵中每个值的几个有效数字。如果希望看到以定点格式打印的矩阵中的所有值，可以使用*fixed\_point\_format*函数。但是不建议这样做，因为这样会产生容易被误解的输出。

: *val =* **fixed\_point\_format** *()*

: *old\_val =* **fixed\_point\_format** *(new\_val)*

: *old\_val =* **fixed\_point\_format** *(new\_val, "local")*

查询或设置控制Octave是否使用缩放格式打印矩阵值的内部变量。

缩放格式在选择的输出的第一行上打印缩放因子，使得最大的矩阵元素可以用一个前导数字书写。例如:

fixed\_point\_format (true)

logspace (1, 7, 5)'

ans =

1.0e+07 \*

0.00000

0.00003

0.00100

0.03162

1.00000

注意，第一个值看起来是0，而实际上是1。由于可能造成混淆，您应该小心启用*fixed\_point\_format*。

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：**format, output\_precision。

**4.1.1空矩阵**

矩阵可以有一个或两个维度为零，空矩阵上的操作由Carl de Boor在《一个空的练习》中描述，SIGNUM，第25卷，第2-6页，1990年，C. N. Nett和W. M. Haddad在《一个空矩阵概念的系统理论的适当实现》中描述，IEEE自动控制学报，第38卷，第5期，1993年5月。简而言之，给定一个标量s，一个m × n矩阵m (mxn)和一个m × n空矩阵[](mxn)(其中一个或两个维度都等于零)，以下是成立的:

s \* [](mxn) = [](mxn) \* s = [](mxn)

[](mxn) + [](mxn) = [](mxn)

[](0xm) \* M(mxn) = [](0xn)

M(mxn) \* [](nx0) = [](mx0)

[](mx0) \* [](0xn) = 0(mxn)

默认情况下，空矩阵的尺寸与空矩阵符号*'[]'*一起打印。内置变量*print\_empty\_dimensions*控制这种行为。

: *val =* **print\_empty\_dimensions** *()*

: *old\_val =* **print\_empty\_dimensions** *(new\_val)*

: *old\_val =* **print\_empty\_dimensions** *(new\_val, "local")*

查询或设置内部变量，该变量控制空矩阵的维度是否与空矩阵符号*”[]”*一起打印。

例如，表达式

zeros (3, 0)

将打印

ans = [](3x0)

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：**格式。

空矩阵也可以在赋值语句中使用，作为删除矩阵的行或列的方便方法。参见赋值表达式。

当Octave解析矩阵表达式时，它检查列表中的元素以确定它们是否都是常量。如果是，则用单个矩阵常数替换列表。

**4.2范围**

*range*是一种方便的方法来表示具有等间距元素的行向量。范围表达式由范围内第一个元素的值、元素之间的增量的可选值以及范围内元素不会超过的最大值定义。基数、增量和极限由冒号( *':'* 字符)分隔，可以包含任何算术表达式和函数调用。如果省略增量，则假定增量为1。例如，范围

1 : 5

定义一组值[1,2,3,4,5]和范围

1 : 3 : 5

定义一组值[1,4]。

虽然范围常量指定了行向量，但Octave通常不会将范围常量转换为向量，除非有必要这样做。这允许您在一个典型的工作站上编写像*1:10000*这样的常量，而无需使用80,000字节的存储空间。

当范围出现在向量内(即方括号内)时，有必要将其转换为向量的常见示例。例如，然而

x = 0 : 0.1 : 1;

定义x为类型为range的变量，占用24字节的内存，即表达式

y = [ 0 : 0.1 : 1];

定义y为matrix类型，占用88字节内存。

这种节省空间的优化可以使用*optimize\_range*函数禁用。

: *val =* **optimize\_range** *()*

: *old\_val =* **optimize\_range** *(new\_val)*

: *old\_val =* **optimize\_range***(new\_val, "local")*

查询或设置是否使用特殊的节省空间的格式来存储范围。

默认值为true。如果此选项设置为false, Octave将以完整矩阵的形式存储范围。

当从带有*”local”*选项的函数内部调用时，将在本地更改函数及其调用的任何子例程的设置。退出该功能时，恢复原来的设置。

**参见：**optimize\_diagonal\_matrix, optimize\_permutation\_matrix。

请注意，范围的上限(或下限，如果增量为负)并不总是包含在值集中。这在某些上下文中可能很有用。例如:

## x is some predefined range or vector or matrix or array

x(1:2:end) += 1; # increment all odd-numbered elements

x(2:2:end) -= 1; # decrement all even-numbered elements

无论*x*的元素个数是否为奇数，上面的代码都能正常工作，因此不需要区别对待这两种情况。

Octave使用浮点运算来计算范围内的值。因此，使用浮点值定义范围可能会导致以下缺陷:

a = -2

b = (0.3 - 0.2 - 0.1)

x = a : b

由于浮点舍入，b可能完全等于零，也可能不等于零，如果不等于零，则可能大于零或小于零，因此最终范围x可能也可能不包括零作为其最终值。类似的:

x = 1.80 : 0.05 : 1.90

y = 1.85 : 0.05 : 1.90

并不像看起来那么容易预测。从Octave 8.3开始，得到的结果是x有三个元素(1.80、1.85和1.90)，而y只有一个元素(1.85但不是1.90)。因此，当在范围中使用浮点数时，改变范围的开始可以很容易地影响范围的结束，即使在上面的例子中没有触及结束值。

为了避免在范围内使用浮点时出现这样的问题，您应该使用以下模式之一。这是对前面代码的更改:

x = (0:2) \* 0.05 + 1.80

y = (0:1) \* 0.05 + 1.85

使其在跨平台、编译器和编译器设置中更安全、更可重复。如果您知道元素的数量，还可以使用*linspace*函数(参见Special Utility Matrices)，它将包括范围的端点。您也可以巧妙地使用 round、floor、ceil、fix 等函数来设置限制和增量，而不会受到浮点数四舍五入的影响。例如，前面的例子可以通过以下方法之一变得更安全，更容易重复:

a = -2

b = round ((0.3 - 0.2 - 0.1) \* 1e12) / 1e12 # rounds to 12 digits

c = floor (0.3 - 0.2 - 0.1) # floors as integer

d = floor ((0.3 - 0.2 - 0.1) \* 1e12) / 1e12 # floors at 12 digits

x = a : b

y = a : c

z = a : d

当向一个范围添加一个标量，从中减去一个标量(或从一个标量减去一个范围)并乘以一个标量时，如果Octave确定这样做是安全的，它将尝试避免对范围进行解包，并将结果也保持为一个范围。例如，做

a = 2\*(1:1e7) - 1;

将产生与*1:2:27 . 07 -1*相同的结果，但不会形成包含1000万个元素的向量。

不允许在冒号表示法中使用0作为增量，如*1:0:1*，因为在确定范围元素的数量时会发生除零的情况。但是，零增量的范围(即所有元素相等)是有用的，特别是在索引中，并且Octave允许使用内置函数来构造它们。注意，因为一个范围必须是一个行向量，*ones(1,10)*产生一个范围，而*ones(10,1)*不会。

当Octave解析一个范围表达式时，它检查表达式的元素以确定它们是否都是常量。如果是，则用单个范围常数替换范围表达式。

**4.3单精度数据类型**

Octave包括对单精度数据类型的支持，并且Octave中的大多数函数接受单精度值并返回单精度答案。使用*single*函数创建单个精度变量。

: y = *single* (x)

将*x*转换为单精度类型。

**参见：**double。

例如:

sngl = single (rand (2, 2))

⇒ sngl =

0.37569 0.92982

0.11962 0.50876

class (sngl)

⇒ single

许多函数也可以直接返回单个精度值。例如

ones (2, 2, "single")

zeros (2, 2, "single")

eye (2, 2, "single")

rand (2, 2, "single")

NaN (2, 2, "single")

NA (2, 2, "single")

Inf (2, 2, "single")

都将返回单精度矩阵。

**4.4整型数据类型**

Octave支持整数矩阵作为双精度的替代方案。可以使用8、16、32或64位表示的有符号和无符号整数。应该注意的是，大多数计算都需要浮点数据，这意味着在涉及数值计算时，整数经常会改变类型。由于这个原因，整数最常用于存储数据，而不是用于计算。

一般来说，大多数整数矩阵是通过将现有矩阵转换为整数来创建的。下面的例子展示了如何将一个矩阵转换为32位整数。

float = rand (2, 2)

⇒ float = 0.37569 0.92982

0.11962 0.50876

integer = int32 (float)

⇒ integer = 0 1

0 1

可以看到，在转换时，浮点值被四舍五入到最接近的整数。

: *tf =* **isinteger** *(x)*

如果*x*是整数对象(int8, uint8, int16等)则返回true。

注意*isinteger(14)*为false，因为Octave中的数字常量是双精度浮点值。

**参见：**isfloat, ischar, islogical, isstring, isnumeric, isa。

: *y =* **int8** *(x)*

将*x*转换为8位整数类型。

**参见：**uint8, int16, uint16, int32, uint32, int64, uint64。

: *y =* **uint8** *(x)*

将*x*转换为无符号8位整数类型。

**参见：**int8, int16, uint16, int32, uint32, int64, uint64。

: *y =* **int16** *(x)*

将*x*转换为16位整数类型。

**参见：**int8, uint8, uint16, int32, uint32, int64, uint64。

: *y =* **uint16** *(x)*

将*x*转换为无符号16位整数类型。

**参见：**int8, uint8, int16, int32, uint32, int64, uint64。

: *y =* **int32** *(x)*

将*x*转换为32位整数类型。

**参见：**int8, uint8, int16, uint16, uint32, int64, uint64。

: *y =* **uint32** *(x)*

将*x*转换为无符号32位整数类型。

**参见：**int8, uint8, int16, uint16, int32, int64, uint64。

: *y =* **int64** *(x)*

将*x*转换为64位整数类型。

**参见：**int8, uint8, int16, uint16, int32, uint32, uint64。

: *y =* **uint64** *(x)*

将*x*转换为无符号64位整数类型。

**参见：**int8, uint8, int16, uint16, int32, uint32, int64。

: *Imax =* **intmax** *()*

: *Imax =* **intmax** *("type")*

: *Imax =* **intmax** *(var)*

返回可由特定整数类型表示的最大整数。

输入要么是指定整数类型的字符串“type”，要么是现有的整数变量var。

type的可能值为

*"int8"*

有符号8位整数。

*"int16"*

有符号16位整数。

*"int32"*

32位有符号整型

*"int64"*

有符号64位整数。

*"uint8"*

无符号8位整数。

*"uint16"*

无符号16位整数

*"uint32"*

32位无符号整数

*"uint64"*

无符号64位整数。

*type*的默认值是*"int32"*。

示例代码-查询一个已有的变量

x = int8 (1);

intmax (x)

⇒ 127

**参见：**intmin, flintmax。

: *Imin =* **intmin** *()*

: *Imin =* **intmin** *("type")*

: *Imin =* **intmin** *(var)*

返回可由特定整数类型表示的最小整数。

输入要么是指定整数类型的字符串*”type”*，要么是现有的整数变量*var*。

type的可能值为

*"int8"*

有符号8位整数。

*"int16"*

有符号16位整数。

*"int32"*

32位有符号整型

*"int64"*

有符号64位整数。

*"uint8"*

无符号8位整数。

*"uint16"*

无符号16位整数

*"uint32"*

32位无符号整数

*"uint64"*

无符号64位整数。

*type*的默认值是*"int32"*。

示例代码-查询一个已有的变量

x = int8 (1);

intmin (x)

⇒ -128

**参见：**intmax, flintmax。

: *Imax =* **flintmax** *()*

: *Imax =* **flintmax** *("double")*

: *Imax =* **flintmax** *("single")*

: *Imax =* **flintmax** *(var)*

返回浮点值中可以连续表示的最大整数。

输入要么是指定浮点类型的字符串，要么是现有的浮点变量*var*。

默认类型是*"double"*，但*"single"*也是一个有效的选项。在IEEE 754兼容系统上，*flintmax*为*2^{53}*表示*"double"*，2^{24}表示*"single"*。

示例代码-查询一个已有的变量

x = single (1);

flintmax (x)

⇒ 16777216

**参见：**intmax, realmax, realmin。

**4.4.1整数运算**

虽然许多数值计算不能在整数中进行，但Octave确实支持整数上的加法和乘法等基本运算。操作符 + 、- 、.\* 和 ./ 可用于同一类型的整数。因此，可以将两个32位整数相加，但不能将一个32位整数和一个16位整数相加。

在做整数运算时，应该考虑下溢和溢出的可能性。当不能使用所选的整数类型表示计算结果时，就会发生这种情况。例如，当使用无符号整数时，不可能表示*10 - 20*的结果。Octave确保整数计算的结果是最接近真实结果的整数。因此，当使用无符号整数时，*10 - 20*的结果是零。

在进行整数除法时，Octave会将结果四舍五入到最接近的整数。这与大多数编程语言不同，在大多数编程语言中，结果通常是最接近的整数。因此，*int32 (5) ./ int32(8)* 的结果是1。

: *C =* **idivide** *(A, B, op)*

使用不同舍入规则的整数除法。

整数除法(如A ./ B)的标准行为是将结果四舍五入到最接近的整数。这并不总是期望的行为，*idivide*允许对整数元素进行逐元素的除法，并对由*op*标志确定的小数部分进行不同的处理。op是一个具有以下值之一的字符串:

*"fix"*

计算A / B，小数部分向零四舍五入。

*"round"*

计算A ./ B，将小数部分四舍五入到最接近的整数。

*"floor"*

计算A / B，将小数部分四舍五入到负无穷。

*"ceil"*

计算A / B，将小数部分四舍五入到正无穷。

如果未指定*op*，则默认为*"fix"*。演示这些舍入规则的示例如下

idivide (int8 ([-3, 3]), int8 (4), "fix")

⇒ 0 0

idivide (int8 ([-3, 3]), int8 (4), "round")

⇒ -1 1

idivide (int8 ([-3, 3]), int8 (4), "floor")

⇒ -1 0

idivide (int8 ([-3, 3]), int8 (4), "ceil")

⇒ 0 1

**参见：**ceiling, floor, fix, round, ldivide, rdivide。

**4.5位操作**

Octave提供了许多用于逐位操作数值的函数。设置和获取单个比特值的基本函数是*bitset*和*bitget*。

: *B =* **bitset** *(A, n)*

: *B =* **bitset** *(A, n, val)*

设置或重置*A*中无符号整数的*n*位。

最低有效位*n* = 1。*val* = 0复位位，*val* = 1复位位。如果没有指定*val*，则默认为1(设置位)。所有输入必须是相同的大小或标量。

例1:设置多个位

x = bitset (1, 3:5)

⇒ x =

5 9 17

dec2bin (x)

⇒

00101

01001

10001

例2:复位和设置位

x = bitset ([15 14], 1, [0 1])

⇒ x =

14 15

**参见：**bitand, bitor, bitxor, bitget, bitcmp, bitshift, intmax, flintmax。

: *b =* **bitget** *(A, n)*

返回*A*中无符号整数*n*位的位值。

最低有效位*n* = 1。

bitget (100, 8:-1:1)

⇒ 0 1 1 0 0 1 0 0

**参见：**bitand, bitor, bitxor, bitset, bitcmp, bitshift, intmax, flintmax。

Octave所有按位操作的参数可以是标量或数组，但*bitcmp*除外，其*k*参数必须是标量。如果多个参数是一个数组，则所有参数必须具有相同的形状，并且将位操作符分别应用于参数的每个元素。如果至少有一个参数是标量，一个是数组，则标量参数是重复的。因此

bitget (100, 8:-1:1)

等同于

bitget (100 \* ones (1, 8), 8:-1:1)

应该注意的是，传递给Octave的位操作函数的所有值都被视为整数。因此，即使上面的*bitset*示例传递了浮点值*10*，它也被视为位*[1,0,1,0]*，而不是本机浮点格式表示*10*的位。

由于可以由数字表示的最大值对于位操作很重要，特别是在形成掩码时，Octave提供了两个实用函数：*flintmax*用于浮点整数，*intmax*用于整数对象(*uint8*, *int64*等)。

Octave还包括基本的按位' and '， ' or '和' exclusive or '操作符。

: *z =* **bitand** *(x, y)*

返回非负整数的位与运算。

*x*, *y*必须在[0,intmax]范围内

**参见：**bitor, bitxor, bitset, bitget, bitcmp, bitshift, intmax, flintmax。

: *z =* **bitor** *(x, y)*

返回非负整数*x*和*y*的位或。

**参见：**bitor, bitxor, bitset, bitget, bitcmp, bitshift, intmax, flintmax。

: *z =* **bitxor** *(x, y)*

返回非负整数*x*和*y*的按位异或。

**参见：**bitand, bitor, bitset, bitget, bitcmp, bitshift, intmax, flintmax。

按位' not '运算符是一元运算符，它对值的每个位执行逻辑负运算。为了使其有意义，必须定义值被否定的掩码。Octave的按位' not '运算符是*bitcmp*。

: *C =* **bitcmp** *(A, k)*

返回*A*中整数的*k*位补码。

如果省略*k*，则假设*k = log2 (flintmax) + 1*。

bitcmp (7,4)

⇒ 8

dec2bin (11)

⇒ 1011

dec2bin (bitcmp (11, 6))

⇒ 110100

**参见：**bitand, bitor, bitxor, bitset, bitget, bitcmp, bitshift, flintmax。

Octave还包括按位左移和右移值的能力。

: *B =* **bitshift** *(A, k)*

: *B =* **bitshift** *(A, k, n)*

返回*a*中*n*位无符号整数的*k*位移位。

正*k*导致左移;右移的一个负值。

如果省略*n*，则默认为64。*n*必须在[1,64]的范围内。

bitshift (eye (3), 1)

⇒

2 0 0

0 2 0

0 0 2

bitshift (10, [-2, -1, 0, 1, 2])

⇒ 2 5 10 20 40

**参见：**bitand, bitor, bitxor, bitset, bitget, bitcmp, intmax, flintmax。

从值的任何一端移出的位都将丢失。Octave还使用算术移位，其中值的符号位在右移期间保留。例如:

bitshift (-10, -1)

⇒ -5

bitshift (int8 (-1), -1)

⇒ -1

注意，*bitshift (int8 (-1), -1)*是-1，因为-1在int8数据类型中的位表示是[1,1,1,1,1,1]。

**4.6逻辑值**

Octave内置了对逻辑值的支持，也就是说，变量要么为真，要么为假。当比较两个变量时，结果将是一个逻辑值，其值取决于比较是否为真。

基本逻辑运算为&、|和!，分别对应“逻辑与”、“逻辑或”和“逻辑否”。这些操作都遵循通常的逻辑规则。

也可以使用逻辑值作为标准数值计算的一部分。在这种情况下，*true*被转换为1,*false*被转换为0，两者都使用双精度浮点数表示。因此，*true\*22 - false/6*的结果是22。

逻辑值也可用于索引矩阵和单元格数组。使用逻辑数组进行索引时，结果将是一个向量，其中包含与逻辑数组的真实部分对应的值。参见逻辑索引。

逻辑值也可以通过将数值对象强制转换为逻辑值或使用*true*或*false*函数来构造。

: *TF =* **logical** *(x)*

将数字对象x转换为逻辑类型。

任何非零值将被转换为true(1)，而零值将被转换为false(0)。非数字值NaN无法转换并会产生错误。

兼容性注意:Octave接受复数值作为输入，而MATLAB会发出错误。

**参见：**double, single, char。

: *val =* **true** *(x)*

: *val =* **true** *(n, m)*

: *val =* **true** *(n, m, k, …)*

: *val =* **true** *(…, "like", var)*

返回一个矩阵或n维数组，其元素都是逻辑1。

如果使用单个标量整数参数调用，则返回指定大小的方阵。

如果使用两个或多个标量整型参数或整型值的向量调用，则返回具有给定维度的数组。

如果在*"like"*后面指定逻辑变量*var*，则输出*val*将具有与*var*相同的稀疏性。

**参见：**false。

: *val =* **false** *(x)*

: *val =* **false** *(n, m)*

: *val =* **false** *(n, m, k, …)*

: *val =* **false** *(…, "like", var)*

返回一个矩阵或n维数组，其元素都是逻辑0。

如果使用单个标量整数参数调用，则返回指定大小的方阵。

如果使用两个或多个标量整型参数或整型值的向量调用，则返回具有给定维度的数组。

如果在*"like"*后面指定逻辑变量*var*，则输出*val*将具有与*var*相同的稀疏性。

**参见：**true。

**4.7自动转换数据类型**

许多运算符和函数都可以使用混合数据类型。例如,

uint8 (1) + 1

⇒ 2

single (1) + 1

⇒ 2

min (single (1), 0)

⇒ 0

其中结果分别是uint8, single和single类型。这样做是为了MATLAB兼容性。有效的混合操作定义如下:

**Mixed Operation Result**

double OP single single

double OP integer integer

double OP char double

double OP logical double

single OP integer integer

single OP char single

single OP logical single

当函数期望使用double类型，但传递的是其他类型时，自动转换依赖于函数:

a = det (int8 ([1 2; 3 4]))

⇒ a = -2

class (a)

⇒ double

a = eig (int8 ([1 2; 3 4]))

⇒ error: eig: wrong type argument 'int8 matrix'

当两个操作数都是整数但宽度不同时，在某些情况下会将它们转换为更宽的位宽，而在其他情况下会抛出错误:

a = min (int8 (100), int16 (200))

⇒ 100

class (a)

⇒ int16

int8 (100) + int16 (200)

⇒ error: binary operator '+' not implemented

for 'int8 scalar' by 'int16 scalar' operations

对于两个整数操作数，它们通常需要都是有符号的，或者都是无符号的。混合有符号和无符号通常会导致错误，即使它们具有相同的位宽。

min (int16 (100), uint16 (200))

⇒ error: min: cannot compute min (int16 scalar, uint16 scalar)

在混合类型索引赋值的情况下，类型不会改变。例如,

x = ones (2, 2);

x(1, 1) = single (2)

⇒ x = 2 1

1 1

其中*x*仍然是双精度类型。

**4.8数字对象的谓词**

由于变量的类型可能在程序执行期间发生变化，因此有必要在运行时进行类型检查。这样做还允许您根据输入的类型更改函数的行为。例如，如果输入是实数，那么这个简单的*abs*实现返回输入的绝对值，如果输入是复数，则返回输入的长度。

function a = abs (x)

if (isreal (x))

a = sign (x) .\* x;

elseif (iscomplex (x))

a = sqrt (real(x).^2 + imag(x).^2);

endif

endfunction

以下函数可用于确定变量的类型。

: *tf =* **isnumeric** *(x)*

如果*x*是数字对象，即整数、实数或复杂数组，则返回true。

逻辑数组和字符数组不被认为是数字数组。

**参见：**isinteger, isfloat, isreal, iscomplex, ischar, islogical, isstring, iscell, isstruct, isa。

: *tf =* **islogical** *(x)*

: *tf =* **isbool** *(x)*

如果*x*是逻辑对象则返回true。

编程说明：isbool 是 islogical 的别名，可以互换使用。

**参见：**ischar, isfloat, isinteger, isstring, isnumeric, isa。

: *tf =* **isfloat** *(x)*

如果*x*是浮点数值对象则返回true。

类double或single的对象是浮点对象。

**参见：**isinteger, ischar, islogical, isnumeric, isstring, isa。

: *tf =* **isreal** *(x)*

如果*x*是非复矩阵或标量，则返回true。

为了与MATLAB兼容，这包括逻辑和字符矩阵。

**参见：**iscomplex, isnumeric, isa。

: *tf =* **iscomplex** *(x)*

如果*x*是一个复数数值对象，则返回true。

**参见：**isreal, isnumeric, ischar, isfloat, islogical, isstring, isa。

: *tf =* **ismatrix** *(x)*

如果*x*是二维数组则返回true。

矩阵是任何类型的数组，其中ndims(x)==2，并且大小(x)返回[M，N]且M和N为非负数。

**参见：**isscalar, isvector, iscell, isstruct, issparse, isa。

: *tf =* **isvector** *(x)*

如果*x*是向量则返回true。

矢量是其中一个维度等于 1（1xN 或 Nx1）的任何类型的二维数组。由于这个定义，一个 1x1 的对象（标量）也是一个矢量。

**参见：**isscalar, ismatrix, iscolumn, isrow, size。

: *tf =* **isrow** *(x)*

如果*x*是行向量则返回true。

行矢量是任何类型的二维数组，其大小 （x） 返回 [1， N] 且非负 N。

**参见：**iscolumn, isscalar, isvector, ismatrix, size。

: *tf =* **iscolumn** *(x)*

如果*x*是列向量则返回true。

列矢量是任何类型的二维数组，其大小 （x） 返回 [N， 1] 且非负 N。

**参见：**isrow, isscalar, isvector, ismatrix, size。

: *tf =* **isscalar** *(x)*

如果*x*是标量则返回true。

标量是大小 （x） 返回的任何类型的单元素对象 [1， 1]。

**参见：**isvector, ismatrix, size。

: *tf =* **issquare** *(x)*

如果x是二维正方形数组则返回true。

方形数组是任何类型的二维数组，其大小 （x） 返回 [N， N]，其中 N 是非负整数。

**参见：**isscalar, isvector, ismatrix, size。

: *tf =* **issymmetric** *(A)*

: *tf =* **issymmetric** *(A, tol)*

: *tf =* **issymmetric** *(A, "skew")*

: *tf =* **issymmetric** *(A, "skew", tol)*

如果 A 是 tol 指定的容差范围内的对称或偏对称数值矩阵，则返回 true。

默认容差为零(使用更快的代码)。

要检查的对称类型可以通过附加输入 *”nonskew”* (默认)指定为规则对称，或*”skew”*为不对称。

背景:如果矩阵的转置等于原矩阵*A == A.'* ，则矩阵是对称的。如果给定公差，则对称性由norm (A - A.', Inf) / norm (A, Inf) < tol。

如果矩阵的转置等于原矩阵的负数，则矩阵是偏对称的:A == -A。如果给定公差，则斜对称性由 norm (A + A.', Inf) / norm (A, Inf) < tol。

**参见：**ishermitian, isdefinite。

: *tf =* **ishermitian** *(A)*

: *tf =* **ishermitian** *(A, tol)*

: *tf =* **ishermitian** *(A, "skew")*

: *tf =* **ishermitian** *(A, "skew", tol)*

如果 A 是 tol 指定的容差范围内的厄米特数值矩阵或偏厄米特数值矩阵，则返回 true。

默认容差为零(使用更快的代码)。

要检查的对称类型可以通过附加输入*”nonskew”*(默认)指定为常规厄米图，或*”skew”*为斜厄米图。

背景：如果一个矩阵的复共轭转置等于原矩阵*A == A'*，那么这个矩阵就是厄米矩阵。如果给定公差，则计算为norm (A - A', Inf) / norm (A, Inf) < tol。

如果一个矩阵的复共轭转置等于原矩阵的负矩阵*A == -A'*，那么这个矩阵就是斜厄米矩阵。如果给定公差，则计算为norm (A + A', Inf) / norm (A, Inf) < tol。

**参见：**issymmetric, isdefinite。

: *tf =* **isdefinite** *(A)*

: *tf =* **isdefinite** *(A, tol)*

如果 A 是 tol 指定的公差范围内的对称正定数值矩阵，则返回 true。

如果省略*tol*，则使用100 \* eps \* norm (A, "fro")。

背景：正定矩阵的特征值都大于零。一个正的半定矩阵的特征值都大于等于零。如果满足以下两个条件，则矩阵*A*很可能是正半定的。

isdefinite (A) ⇒ 0

isdefinite (A + 5\*tol, tol) ⇒ 1

**参见：**issymmetric, ishermitian。

: *tf =* **isbanded** *(A, lower, upper)*

如果 A 是一个数值矩阵，其条目限制在主对角线下方的下对角线和主对角线上方的上对角线之间，则返回 true。

*Lower*和*upper*必须是非负整数。

**参见：**isdiag, istril, istriu, bandwidth。

: *tf =* **isdiag** *(A)*

如果 A 是对角线数值矩阵，则返回 true，该矩阵定义为二维数组，其中主对角线上方和下方的所有元素均为零。

**参见：**isband, istril, istriu, diag, bandwidth。

: *tf =* **istril** *(A)*

如果 A 是下三角形数值矩阵,则返回 true。

下三角矩阵只在主对角线及以下有非零元素。

**参见：**istriu, is带状，isdiag, tril，带宽。

: *tf =* **istriu** *(A)*

如果 A 是上三角形数值矩阵,则返回 true。

一个上三角矩阵只在主对角线及以上有非零元素。

**参见：**isdiag, isband, istril, triu, bandwidth。

: *tf =* **isprime** *(x)*

返回一个逻辑数组，当*x*的元素为质数时为真，当元素为非质数时为假。

素数通常被定义为大于1的正整数(例如，2,3，…)，并且只能被其本身和1整除。Octave扩展了这个定义，包括负整数和复数值。如果一个负整数的正整数是素数，那么它就是素数。它等价于*isprime (abs (x))*。

如果*class(x)* 是复的，那么在高斯整数的域中测试素数(https://en.wikipedia.org/wiki/Gaussian\_integer)。一些非复整数在一般意义上是素数，但在高斯整数的范围内不是。例如， *5 = (1+2i)\*(1-2i)* 表明5不是素数，因为它有一个因数，而不是它本身和1。在同一矩阵中同时测试复数值和实值时要小心。

例子:

isprime (1:6)

⇒ 0 1 1 0 1 0

isprime ([i, 2, 3, 5])

⇒ 0 0 1 0

编程简注:*isprime*适用于*abs(x) < 2^64*范围内的所有*x*。大于*flintmax*的输入强制转换为*uint64*。

对于较大的输入，如果你已经安装并加载了Symbolic包，则使用' sym ':

isprime (sym ('58745389709258902525390450') + (0:4))

⇒ 0 1 0 0 0

注意:MATLAB不扩展素数的定义，如果给定负输入或复杂输入，将产生错误。

**参见：**质数，因子，gcd, lcm。

: tf = isuniform (v)

: [tf, delta] = isuniform (v)

如果实矢量 v 间隔均匀，则返回 true，否则返回 false。

如果所有元素之间的平均差 （delta） 相同，并且在 4 \* eps （max （abs （v））） 的公差范围内，则矢量是均匀的。

可选的输出增量是组件之间的均匀差值。如果矢量不均匀，则 delta 为 NaN。对于浮点输入，delta 与 v 属于同一类，对于整数、逻辑和字符输入，delta 与 double 类相同。

编程说明：对于空输入或标量输入的特殊情况，输出始终为 false。如果任何元素为 NaN，则输出为 false。如果 delta 小于计算出的相对容差，则使用 eps 的绝对容差。

参见：linspace、colon。

如果您不知道变量的属性，而是希望知道定义了哪些变量，并收集关于工作空间本身的其他信息，请参阅变量的状态。

**5 字符串**

字符串常量由用双引号或单引号括起来的一系列字符组成。例如，下面两个表达式

"parrot"

'parrot'

表示内容为*' parrot '*的字符串。Octave中的字符串可以是任意长度。

由于单引号也用于转置运算符(参见算术运算符)，但双引号在Octave中没有其他用途，因此最好使用双引号来表示字符串。

可以使用定义矩阵的表示法将字符串连接起来。例如，表达式

[ "foo" , "bar" , "baz" ]

生成内容为*' foobarbaz '*的字符串。有关创建矩阵的详细信息，请参阅数字数据类型。

虽然字符串原则上可以存储任意内容，但大多数函数都希望它们是UTF-8编码的Unicode字符串。

此外，可以创建字符串而不实际写入文本。函数*blanks*创建一个只由空白字符(ASCII码32)组成的给定长度的字符串。

: *str =* **blanks** *(n)*

返回一个包含*n*个空格的字符串。

例如:

blanks (10);

whos ans

⇒

Attr Name Size Bytes Class

==== ==== ==== ===== =====

ans 1x10 10 char

**参见：**repmat。

**5.1字符串常量中的转义序列**

在双引号字符串中，反斜杠字符用于引入表示其他字符的*escape sequences*（转义序列：指的是一个转义字符和随后的字符组合，这些字符组合在输出时不按照原样输出，而是指定了特定的行为。）。例如，*' \n '*在双引号字符串中嵌入换行符，*' \" '*嵌入双引号字符。在单引号字符串中，反斜杠不是特殊字符。下面是一个例子:

double ("\n")

⇒ 10

double ('\n')

⇒ [ 92 110 ]

下面是Octave中使用的所有转义序列的表(在双引号字符串中)。它们与C编程语言中使用的相同。

\\

表示字面反斜杠' \ '。

*\"*

表示字面双引号字符“"”。

*\'*

表示字面单引号字符" '。

*\0*

表示空字符，控件-@，ASCII码0。

*\a*

表示“alert”字符，control-g, ASCII码7。

*\b*

表示退格，control-h, ASCII码8。

*\f*

表示一个formfeed, control- 1, ASCII码12。

*\n*

表示换行符，control-j, ASCII码10。

*\r*

表示一个回车，control-m, ASCII码13。

*\t*

表示水平制表符，control-i, ASCII码9。

*\v*

表示垂直制表符，control-k, ASCII码11。

*\nnn*

表示八进制值nnn，其中nnn是0到7之间的1到3位数字。例如，ASCII ESC(转义)字符的代码是' \033 '。

*\xhh…*

表示十六进制值hh，其中hh是十六进制数字(' 0 '到' 9 '以及' A '到' F '或' A '到' F ')。与ANSI C中的相同结构一样，转义序列一直持续到看到第一个非十六进制数字为止。但是，使用两个以上的十六进制数字会产生未定义的结果。

在单引号字符串中只有一个转义序列:您可以连续使用两个单引号字符插入一个单引号字符。例如,

'I can''t escape'

⇒ I can't escape

在脚本中，如果需要，可以使用is\_dq\_string和is\_sq\_string来区分这两种不同的字符串类型。

: *tf =* **is\_dq\_string** *(x)*

如果*x*是双引号字符串则返回true。

**参见：**is\_sq\_string, ischar。

: *tf =* **is\_sq\_string** *(x)*

如果*x*是单引号的字符字符串,返回true。

**参见：**is\_dq\_string,ischar。

**5.2字符数组**

Octave使用的字符串表示形式是一个字符数组，因此在内部字符串 *”dddddddddd”* 实际上是一个长度为10的行向量，在所有位置都包含值100(100是“d”的ASCII码)。这有助于将其明显地推广到字符矩阵。使用字符矩阵，可以在一个变量中表示相同长度字符串的集合。在Octave中使用的约定是字符矩阵中的每一行都是一个单独的字符串，但是让每一列代表一个字符串是同样可能的。

创建字符矩阵最简单的方法是将几个字符串放在一起形成一个矩阵。

collection = [ "String #1"; "String #2" ];

这将创建一个2 × 9的字符矩阵。

函数ischar可用于测试对象是否是字符矩阵。

: *tf =* **ischar** *(x)*

如果x是字符数组则返回true。

**参见：**isfloat, isinteger, islogical, isnumeric, isstring, iscellstr, isa。

: *tf =* **isstring** *(s)*

如果s是字符串数组则返回true。

字符串数组是一种数据类型，它在数组中的每个元素处存储字符串(字符的行向量)。它与字符数组不同，字符数组是n维数组，其中每个元素是单个1x1字符。它也不同于字符串的单元格数组，单元格数组在每个元素上存储字符串，但使用单元格索引'{}'来访问元素，而不是使用普通数组索引'()'的字符串数组。

编程简注：Octave还没有实现字符串数组，所以这个函数总是返回false。

**参见：**ischar, iscellstr, isfloat, isinteger, islogical, isnumeric, isa。

要测试一个对象是否是字符串(即，一个1xN行的字符向量，而不是字符矩阵)，你可以将ischar函数与isrow函数结合使用，如下例所示:

ischar (collection)

⇒ 1

ischar (collection) && isrow (collection)

⇒ 0

ischar ("my string") && isrow ("my string")

⇒ 1

一个相关的问题是，当从不同长度的字符串创建字符矩阵时会发生什么。答案是Octave在比最长字符串短的字符串的末尾放置空白字符。使用string\_fill\_char函数可以使用与空白字符不同的字符。

: *val =* **string\_fill\_char** *()*

: *old\_val =* **string\_fill\_char** *(new\_val)*

: *old\_val =* **string\_fill\_char** *(new\_val, "local")*

查询或设置用于将字符矩阵的所有行填充为相同长度的内部变量。

取值必须为单个字符，默认为" "(一个空格)。例如:

string\_fill\_char ("X");

[ "these"; "are"; "strings" ]

⇒ "theseXX"

"areXXXX"

"strings"

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。

在这种情况下，控制文本对齐的另一个有用函数是strjust函数。

: *str =* **strjust** *(s)*

: *str =* **strjust** *(s, pos)*

返回文本s，根据pos进行对齐，pos可以是“左”、“中”或“右”。

如果省略pos，则默认为"right"。

空字符被空格替换。所有其他字符数据都被视为非空白。

例子:

strjust (["a"; "ab"; "abc"; "abcd"])

⇒

" a"

" ab"

" abc"

"abcd"

**参见：**deblank, strrep, strtrim, untabify。

这显示了字符矩阵的一个问题。不可能表示不同长度的字符串。解决方案是使用字符串的单元格数组，这在字符串的单元格数组中有描述。

**5.3字符串操作**

Octave支持多种操作字符串的函数。由于字符串只是一个矩阵，因此可以使用标准操作符完成简单的操作。下面的示例展示了如何用下划线替换所有空白字符。

quote = ...

"First things first, but not necessarily in that order";

quote( quote == " " ) = "\_"

⇒ quote =

First\_things\_first,\_but\_not\_necessarily\_in\_that\_order

对于更复杂的操作，如搜索、替换和通用正则表达式，Octave提供了以下函数。

**5.3.1常用字符串操作**

以下函数可用于执行常见的String操作。

: *y =* **lower** *(s)*

: *y =* **tolower** *(s)*

返回字符串或单元格字符串s的副本，其中每个大写字符替换为相应的小写字符;非字母字符保持不变。

例如:

lower ("MiXeD cAsE 123")

⇒ "mixed case 123"

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

**参见：**upper。

: *y =* **upper** *(s)*

: *y =* **toupper** *(s)*

返回字符串或单元格字符串s的副本，其中每个小写字符替换为相应的大写字符;非字母字符保持不变。

例如:

upper ("MiXeD cAsE 123")

⇒ "MIXED CASE 123"

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

**参见：**lower。

: *s =* **deblank** *(s)*

从s中删除尾随空格和空。

如果s是一个矩阵，deblank将每行裁剪为最长字符串的长度。如果s是字符串的单元格数组，则对每个字符串元素进行递归操作。

例子:

deblank (" abc ")

⇒ " abc"

deblank ([" abc "; " def "])

⇒ [" abc " ; " def"]

**参见：**strtrim。

: *s =* **strtrim** *(s)*

从s中删除前导和尾随空格。

如果s是一个矩阵，strtrim将每一行裁剪为最长字符串的长度。如果s是字符串的单元格数组，则对每个字符串元素进行递归操作。

例如:

strtrim (" abc ")

⇒ "abc"

strtrim ([" abc "; " def "])

⇒ ["abc " ; " def"]

**参见：**deblank。

: *s =* **strtrunc** *(s, n)*

将字符串s截断为长度n。

如果s是字符矩阵，则调整列数。

如果s是字符串的单元格数组，则对每个单元格元素执行操作，并返回新的单元格数组。

: *str =* **untabify** *(t)*

: *str =* **untabify** *(t, tw)*

: *str =* **untabify** *(t, tw, deblank)*

将t中的制表符替换为空格。

输入t可以是二维字符数组，也可以是字符串的单元数组。输出与输入是同一个类。

制表符宽度由tw指定，默认为8。

如果可选参数deblank为true，则将从字符数据的末尾删除空格。

下面的示例读取一个文件，并写入该文件的非表化版本，去掉末尾的空格。

fid = fopen ("tabbed\_script.m");

text = char (fread (fid, "uchar")');

fclose (fid);

fid = fopen ("untabified\_script.m", "w");

text = untabify (strsplit (text, "\n"), 8, true);

fprintf (fid, "%s\n", text{:});

fclose (fid);

**参见：**strjust, strsplit, deblank。

: *newstr =* **do\_string\_escapes** *(string)*

将字符串中的转义序列转换为它们所表示的字符。

转义序列以前导反斜杠('\')开头，后面跟着1-3个字符(例如。， "\n" =>换行符)。

**参见：**undo\_string\_escapes。

: *newstr =* **undo\_string\_escapes** *(string)*

将字符串中的特殊字符转换回其转义形式。

例如，表达式

bell = "\a";

将警报字符(control-g, ASCII码7)的值赋给字符串变量bell。如果这个字符串被打印出来，系统将会敲响终端铃声(如果可能的话)。这通常是期望的结果。然而，有时能够打印字符串的原始表示是有用的，用它们的转义序列替换特殊字符。例如,

octave:13> undo\_string\_escapes (bell)

ans = \a

将不可打印的警告字符替换为其可打印的表示形式。

**参见：**do\_string\_escapes。

**5.3.2连接字符串**

字符串可以使用矩阵表示法连接(参见字符串，字符数组)，这通常是最自然的方法。例如:

fullname = [fname ".txt"];

email = ["<" user "@" domain ">"];

在每种情况下，都很容易看到最终字符串的样子。这种方法也是最有效的。当使用矩阵连接时，解析器立即开始连接字符串，而不必处理函数调用的开销和相关函数的输入验证。

换行函数可用于连接字符串，以便在显示时显示为多行文本。

: *c =* **newline**

返回与换行符对应的字符。

这相当于"\n"

示例代码

joined\_string = [newline "line1" newline "line2"]

⇒

line1

line2

**参见：**strcat, strjoin, strsplit。

此外，还有其他几个用于连接字符串对象的函数，它们在特定情况下很有用:char、strvcat、strcat和cstrcat。最后，可以使用通用的连接函数:参见cat、horzcat和vertcat。

除了cstrcat之外，所有字符串连接函数都通过为每个元素(或多字节序列)取相应的UTF-8字符来将数字输入转换为字符数据，如下例所示:

char ([98, 97, 110, 97, 110, 97])

⇒ banana

有关区域设置编码与UTF-8之间的转换，请参见unicode2native和native2unicode。

Char和strcat垂直连接，而strcat和cstrcat水平连接。例如:

char ("an apple", "two pears")

⇒ an apple

two pears

strcat ("oc", "tave", " is", " good", " for you")

⇒ octave is good for you

Char在输出中为输入中的每个空字符串生成一个空行。另一方面，Strvcat消除了空字符串。

char ("orange", "green", "", "red")

⇒ orange

green

red

strvcat ("orange", "green", "", "red")

⇒ orange

green

red

除了cstrcat之外的所有字符串连接函数也接受单元格数组数据(参见单元格数组)。Char和strvcat将单元格数组转换为字符数组，而strcat在单元格数组的单元格内进行连接:

char ({"red", "green", "", "blue"})

⇒ red

green

blue

strcat ({"abc"; "ghi"}, {"def"; "jkl"})

⇒

{

[1,1] = abcdef

[2,1] = ghijkl

}

Strcat删除参数后面的空白(单元格数组内除外)，而cstrcat不保留空白。这两种行为都是有用的，可以从下面的例子中看到:

strcat (["dir1";"directory2"], ["/";"/"], ["file1";"file2"])

⇒ dir1/file1

directory2/file2

cstrcat (["thirteen apples"; "a banana"], [" 5$";" 1$"])

⇒ thirteen apples 5$

a banana 1$

注意，在上面的cstrcat示例中，空格来自字符串数组中字符串的内部表示(参见字符数组)。

: *C =* **char** *(A)*

: *C =* **char** *(A, …)*

: *C =* **char** *(str1, str2, …)*

: *C =* **char** *(cell\_array)*

从一个或多个数字矩阵、字符矩阵或单元格数组创建字符串数组。

参数垂直连接。返回的值将根据需要填充空白，以使字符串数组的每行具有相同的长度。空输入字符串是重要的，将在输出中连接。

对于数字输入，每个元素被转换为相应的ASCII字符。如果输入超出ASCII范围(0-255)，则会导致范围错误。

对于单元格数组，每个元素是单独连接的。通过char转换的单元格数组大多可以用cellstr转换回来。例如:

char ([97, 98, 99], "", {"98", "99", 100}, "str1", ["ha", "lf"])

⇒ ["abc "

" "

"98 "

"99 "

"d "

"str1"

"half"]

参见:strvcat, cellstr。

: *C =* **strvcat** *(A)*

: *C =* **strvcat** *(A, …)*

: *C =* **strvcat** *(str1, str2, …)*

: *C =* **strvcat** *(cell\_array)*

从一个或多个数字矩阵、字符矩阵或单元格数组创建字符数组。

参数垂直连接。返回的值将根据需要填充空白，以使字符串数组的每行具有相同的长度。与char不同，空字符串将被删除并且不会出现在输出中。

对于数字输入，每个元素被转换为相应的ASCII字符。如果输入超出ASCII范围(0-255)，则会导致范围错误。

对于单元格数组，每个元素是单独连接的。通过strvcat转换的单元格数组大多可以用cellstr转换回来。例如:

strvcat ([97, 98, 99], "", {"98", "99", 100}, "str1", ["ha", "lf"])

⇒ ["abc "

"98 "

"99 "

"d "

"str1"

"half"]

**参见:**char, strcat, cstrcat。

: *str =* **strcat** *(s1, s2, …)*

返回一个包含水平连接的所有参数的字符串。

如果参数是单元格字符串，则strcat返回一个连接了各个单元格的单元格字符串。对于数字输入，每个元素被转换为相应的ASCII字符。任何字符串输入的尾随空格在字符串连接之前被消除。注意，单元格字符串值没有空格。

例如:

strcat ("|", " leading space is preserved", "|")

⇒ | leading space is preserved|

strcat ("|", "trailing space is eliminated ", "|")

⇒ |trailing space is eliminated|

strcat ("homogeneous space |", " ", "| is also eliminated")

⇒ homogeneous space || is also eliminated

s = [ "ab"; "cde" ];

strcat (s, s, s)

⇒

"ababab "

"cdecdecde"

s = { "ab"; "cd " };

strcat (s, s, s)

⇒

{

[1,1] = ababab

[2,1] = cd cd cd

}

**参见:**cstrcat, char, strvcat。

: *str =* **cstrcat** *(s1, s2, …)*

返回一个字符串，其中包含所有参数水平连接，并保留尾随空格。

例如:

cstrcat ("ab ", "cd")

⇒ "ab cd"

s = [ "ab"; "cde" ];

cstrcat (s, s, s)

⇒ "ab ab ab "

"cdecdecde"

**参见:**strcat, char, strvcat。

**5.3.3分割和连接字符串**

: *str =* **substr** *(s, offset)*

: **substr** *(s, offset, len)*

返回s的子字符串，从字符号偏移开始，长度为len个字符。

偏移量的位置编号从1开始。如果offset为负值，则从距离字符串末尾很远的地方开始提取。

如果省略len，则子字符串扩展到s的末尾。如果len为负值，则提取到字符串末尾len以内的字符

例子:

substr ("This is a test string", 6, 9)

⇒ "is a test"

substr ("This is a test string", -11)

⇒ "test string"

substr ("This is a test string", -11, -7)

⇒ "test"

这个函数是按照Perl中等效函数的模式设计的。

: *[tok, rem] =* **strtok** *(str)*

: *[tok, rem] =* **strtok** *(str, delim)*

查找字符串str中的所有字符，但不包括字符串delim中的第一个字符。

STR也可以是字符串的单元数组，在这种情况下，函数对每个单独的字符串执行，并返回一个由令牌和余数组成的单元数组。

前导分隔符将被忽略。如果未指定delim，则假定为空白。

如果请求rem，则它包含字符串的其余部分，从第一个分隔符开始。

例子:

strtok ("this is the life")

⇒ "this"

[tok, rem] = strtok ("14\*27+31", "+-\*/")

⇒

tok = 14

rem = \*27+31

**参见:**index, strsplit, strchr, isspace。

: *[cstr] =* **strsplit** *(str)*

: *[cstr] =* **strsplit** *(str, del)*

: *[cstr] =* **strsplit** *(…, name, value)*

: *[cstr, matches] =* **strsplit** *(…)*

使用del指定的分隔符拆分字符串str并返回子字符串的单元格字符串数组。

如果未指定分隔符，则在空格{" "，"\f"， "\n"， "\r"， "\t"， "\v"}处分割字符串。否则，分隔符del必须是字符串或字符串的单元格数组。默认情况下，输入字符串s中的连续分隔符被折叠成一个分隔符，从而产生单个分隔符。

支持的名称/值对参数有:

可为true(默认值)或false的折叠分隔符。

分隔符类型，可以取值为"simple"(默认)或" regulareexpression "。一个简单的分隔符与文本完全匹配。否则，将使用regexp中列出的正则表达式语法。

第二个可选输出matches返回在原始字符串中匹配的分隔符。

使用简单分隔符的例子:

strsplit ("a b c")

⇒

{

[1,1] = a

[1,2] = b

[1,3] = c

}

strsplit ("a,b,c", ",")

⇒

{

[1,1] = a

[1,2] = b

[1,3] = c

}

strsplit ("a foo b,bar c", {" ", ",", "foo", "bar"})

⇒

{

[1,1] = a

[1,2] = b

[1,3] = c

}

strsplit ("a,,b, c", {",", " "}, "collapsedelimiters", false)

⇒

{

[1,1] = a

[1,2] =

[1,3] = b

[1,4] =

[1,5] = c

}

使用正则表达式分隔符的示例:

strsplit ("a foo b,bar c", ',|\s|foo|bar', ...

"delimitertype", "regularexpression")

⇒

{

[1,1] = a

[1,2] = b

[1,3] = c

}

strsplit ("a,,b, c", '[, ]', "collapsedelimiters", false, ...

"delimitertype", "regularexpression")

⇒

{

[1,1] = a

[1,2] =

[1,3] = b

[1,4] =

[1,5] = c

}

strsplit ("a,\t,b, c", {',', '\s'}, "delimitertype", "regularexpression")

⇒

{

[1,1] = a

[1,2] = b

[1,3] = c

}

strsplit ("a,\t,b, c", {',', ' ', '\t'}, "collapsedelimiters", false)

⇒

{

[1,1] = a

[1,2] =

[1,3] =

[1,4] = b

[1,5] =

[1,6] = c

}

**参见:**strsplit, strjoin, strtok, regexp。

: *[cstr] =* **ostrsplit** *(s, sep)*

: *[cstr] =* **ostrsplit** *(s, sep, strip\_empty)*

使用一个或多个分隔符拆分字符串s，并返回字符串的单元格数组。

连续分隔符和边界处的分隔符将导致空字符串，除非strip\_empty为真。strip\_empty默认值为false。

二维字符数组在分隔符和原始列边界处分开。

例子:

ostrsplit ("a,b,c", ",")

⇒

{

[1,1] = a

[1,2] = b

[1,3] = c

}

ostrsplit (["a,b" ; "cde"], ",")

⇒

{

[1,1] = a

[1,2] = b

[1,3] = cde

}

**参见:**strsplit, strtok。

: *str =* **strjoin** *(cstr)*

: *str =* **strjoin** *(cstr, delimiter)*

将单元格字符串数组cstr的元素连接为单个字符串。

如果未指定分隔符，则cstr的元素之间用空格分隔。

如果将分隔符指定为字符串，则使用该字符串连接单元格字符串数组。支持转义序列。

如果delimiter是长度比cstr小一个的单元格字符串数组，则cstr的元素通过交错分隔符的单元格字符串元素来连接。不支持转义序列。

strjoin ({'Octave','Scilab','Lush','Yorick'}, '\*')

⇒ 'Octave\*Scilab\*Lush\*Yorick'

**参见:**strsplit。

**5.3.4字符串搜索**

由于字符串是字符数组，因此字符串之间的比较逐个元素地进行，如下例所示:

GNU = "GNU's Not UNIX";

spaces = (GNU == " ")

⇒ spaces =

0 0 0 0 0 1 0 0 0 1 0 0 0 0

要确定两个字符串是否相同，必须使用strcmp函数。它比较完整的字符串，并且区分大小写。strncmp只比较前N个字符(N作为参数给定)。Strcmpi和strncmpi是用于不区分大小写比较的对应函数。

: *tf =* **strcmp** *(str1, str2)*

如果字符串str1和str2相同，则返回1，否则返回0。

如果str1或str2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数组(大小相同或只有一个元素)、字符矩阵或字符串。

注意:为了与MATLAB兼容，如果字符串相等，Octave的strcmp函数返回1，否则返回0。这与相应的C库函数正好相反。

**参见:**strcmpi, strncmp, strncmpi。

: *tf =* **strncmp** *(str1, str2, n)*

如果字符串str1和str2的前n个字符相同，则返回1，否则返回0。

strncmp ("abce", "abcd", 3)

⇒ 1

如果str1或str2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数组(大小相同或只有一个元素)、字符矩阵或字符串。

strncmp ("abce", {"abcd", "bca", "abc"}, 3)

⇒ [1, 0, 1]

注意:为了与MATLAB兼容，如果字符串相等，Octave的strncmp函数返回1，否则返回0。这与相应的C库函数正好相反。

**参见:**strncmpi, strcmp, strcmpi。

: *tf =* **strcmpi** *(str1, str2)*

如果字符串str1和str2相同(不考虑字母大小写)则返回1，否则返回0。

如果str1或str2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数组(大小相同或只有一个元素)、字符矩阵或字符串。

注意:为了与MATLAB兼容，如果字符串相等，Octave的strcmp函数返回1，否则返回0。这与相应的C库函数正好相反。

注意:不支持国家字母。

**参见:**strcmp, strncmp, strncmpi。

: *tf =* **strncmpi** *(str1, str2, n)*

如果s1和s2的前n个字符相同(不考虑字母大小写)，则返回1，否则返回0。

如果str1或str2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数组(大小相同或只有一个元素)、字符矩阵或字符串。

注意:为了与MATLAB兼容，如果字符串相等，Octave的strncmpi函数返回1，否则返回0。这与相应的C库函数正好相反。

注意:不支持国家字母。

**参见:**strncmp, strcmp, strcmpi。

除了这些比较函数之外，还有更专门的函数来查找字符串中搜索模式的索引位置。

: *retval =* **startsWith** *(str, pattern)*

: *retval =* **startsWith** *(str, pattern, "IgnoreCase", ignore\_case)*

检查字符串是否以pattern开头。

返回一个逻辑值数组，该数组指示输入str(单个字符串或字符串的单元数组)中的哪个字符串以输入模式(单个字符串或字符串的单元数组)开头。

如果参数"IgnoreCase"的值为true，则该函数将忽略str和pattern的字母大小写。默认情况下，比较区分大小写。

例子:

## one string and one pattern while considering case

startsWith ("hello", "he")

⇒ 1

## one string and one pattern while ignoring case

startsWith ("hello", "HE", "IgnoreCase", true)

⇒ 1

## multiple strings and multiple patterns while considering case

startsWith ({"lab work.pptx", "data.txt", "foundations.ppt"},

{"lab", "data"})

⇒ 1 1 0

## multiple strings and one pattern while considering case

startsWith ({"DATASHEET.ods", "data.txt", "foundations.ppt"},

"data", "IgnoreCase", false)

⇒ 0 1 0

## multiple strings and one pattern while ignoring case

startsWith ({"DATASHEET.ods", "data.txt", "foundations.ppt"},

"data", "IgnoreCase", true)

⇒ 1 1 0

**参见:**endsWith, regexp, strncmp, strncmpi。

: *retval =* **endsWith** *(str, pattern)*

: *retval =* **endsWith** *(str, pattern, "IgnoreCase", ignore\_case)*

检查字符串是否以pattern结尾。

返回一个逻辑值数组，该数组指示输入str(单个字符串或字符串的单元数组)中的哪个字符串以输入模式(单个字符串或字符串的单元数组)结束。

如果参数"IgnoreCase"的值为true，则该函数将忽略str和pattern的字母大小写。默认情况下，比较区分大小写。

例子:

## one string and one pattern while considering case

endsWith ("hello", "lo")

⇒ 1

## one string and one pattern while ignoring case

endsWith ("hello", "LO", "IgnoreCase", true)

⇒ 1

## multiple strings and multiple patterns while considering case

endsWith ({"tests.txt", "mydoc.odt", "myFunc.m", "results.pptx"},

{".docx", ".odt", ".txt"})

⇒ 1 1 0 0

## multiple strings and one pattern while considering case

endsWith ({"TESTS.TXT", "mydoc.odt", "result.txt", "myFunc.m"},

".txt", "IgnoreCase", false)

⇒ 0 0 1 0

## multiple strings and one pattern while ignoring case

endsWith ({"TESTS.TXT", "mydoc.odt", "result.txt", "myFunc.m"},

".txt", "IgnoreCase", true)

⇒ 1 0 1 0

**参见:**startsWith, regexp, strncmp, strncmpi。

: *v =* **findstr** *(s, t)*

: *v =* **findstr** *(s, t, overlap)*

这个功能已经过时了。使用strfind代替。

返回两个字符串s和t中较长字符串中出现较短字符串的所有位置的向量。

如果可选参数overlap为true(默认值)，则返回的向量可以包含重叠的位置。例如:

findstr ("ababab", "a")

⇒ [1, 3, 5];

findstr ("abababa", "aba", 0)

⇒ [1, 5]

注意:findstr已过时。在所有新代码中使用strfind。

**参见:**strfind, strmatch, strcmp, strncmp, strcmpi, strncmpi, find。

: *idx =* **strchr** *(str, chars)*

: *idx =* **strchr** *(str, chars, n)*

: *idx =* **strchr** *(str, chars, n, direction)*

: *[i, j] =* **strchr** *(…)*

在字符串str中搜索所设置字符中出现的字符。

返回值以及n和direction参数的行为与find中的行为相同。

在大多数情况下，这将比使用regexp更快。

**参见:**find。

: *n =* **index** *(s, t)*

: *n =* **index** *(s, t, direction)*

返回字符串t在字符串s中第一次出现的位置，如果没有找到，则返回0。

S也可以是字符串数组或字符串的单元数组。

例如:

index ("Teststring", "t")

⇒ 4

如果direction为“first”，则返回找到的第一个元素。如果direction为“last”，则返回找到的最后一个元素。

**参见:**find, rindex。

: *n =* **rindex** *(s, t)*

返回字符串t在字符串s中最后出现的位置，如果没有出现则返回0。

S也可以是字符串数组或字符串的单元数组。

例如:

rindex ("Teststring", "t")

⇒ 6

rindex函数相当于方向设置为“最后”的index。

**参见:**find, index。

: *idx =* **unicode\_idx** *(str)*

返回一个数组，其中包含str中每个UTF-8编码字符的索引。

unicode\_idx ("aäbc")

⇒ [1, 2, 2, 3, 4]

: *idx =* **strfind** *(str, pattern)*

: *idx =* **strfind** *(cellstr, pattern)*

: *idx =* **strfind** *(…, "overlaps", val)*

: *idx =* **strfind** *(…, "forcecelloutput", val)*

在字符串str中搜索pattern，并返回向量idx中每个pattern出现的起始索引。

如果没有出现这种情况，或者pattern比str长，或者pattern本身为空，则idx为空数组[]。

可选参数"overlaps"决定模式是否可以匹配str中的每个位置(true)，还是只匹配完整模式的唯一出现(false)。默认为true。

如果指定了字符串单元格数组cellstr，则idx是一个向量单元格数组，如上所述。

可选参数"forcecelloutput"强制idx作为向量单元数组返回。默认为false。

例子:

strfind ("abababa", "aba")

⇒ [1, 3, 5]

strfind ("abababa", "aba", "overlaps", false)

⇒ [1, 5]

strfind ({"abababa", "bebebe", "ab"}, "aba")

⇒

{

[1,1] =

1 3 5

[1,2] = [](1x0)

[1,3] = [](1x0)

}

strfind ("abababa", "aba", "forcecelloutput", true)

⇒

{

[1,1] =

1 3 5

}

**参见:**regexp, regexpi, find。

: *idx =* **strmatch** *(s, A)*

: *idx =* **strmatch** *(s, A, "exact")*

这个功能已经过时了。请使用其他选项，例如strncmp或strcmp。

返回A中以字符串s开头的项的索引。

第二个参数A必须是字符串、字符矩阵或字符串的单元格数组。

如果没有给出第三个参数"exact"，则s只需要匹配A到s的长度。匹配时忽略s和A中的尾随空格和null。

例如:

strmatch ("apple", "apple juice")

⇒ 1

strmatch ("apple", ["apple "; "apple juice"; "an apple"])

⇒ [1; 2]

strmatch ("apple", ["apple "; "apple juice"; "an apple"], "exact")

⇒ [1]

注意:strmatch已经过时了(并且在MATLAB中与字符串的单元格数组一起使用时会产生不正确的结果)。在所有新代码中使用strncmp(正常情况)或strcmp(“精确”情况)。其他可能的替换，取决于应用程序，包括regexp或validatestring。

**参见:**strncmp, strcmp, regexp, strfind, validatestring。

**5.3.5搜索和替换字符串**

: *newstr =* **strrep** *(str, ptn, rep)*

: *newstr =* **strrep** *(cellstr, ptn, rep)*

: *newstr =* **strrep** *(…, "overlaps", val)*

将字符串str中模式ptn的所有出现替换为字符串rep并返回结果。

可选参数"overlaps"决定模式是否可以匹配str中的每个位置(true)，还是只匹配完整模式的唯一出现(false)。默认为true。

S也可以是字符串的单元格数组，在这种情况下，对每个元素进行替换并返回一个单元格数组。

例子:

strrep ("This is a test string", "is", "&%$")

⇒ "Th&%$ &%$ a test string"

**参见:**regexprep, strfind。

: *newstr =* **erase** *(str, ptn)*

删除str中出现的所有ptn。

STR和PTN可以是普通字符串、字符串的单元格数组或字符数组。

例子

## string, single pattern

erase ("Hello World!", " World")

⇒ "Hello!"

## cellstr, single pattern

erase ({"Hello", "World!"}, "World")

⇒ {"Hello", "!"}

## string, multiple patterns

erase ("The Octave interpreter is fabulous", ...

{"interpreter ", "The "})

⇒ "Octave is fabulous"

## cellstr, multiple patterns

erase ({"The ", "Octave interpreter ", "is fabulous"}, ...

{"interpreter ", "The "})

⇒ {"", "Octave ", "is fabulous"}

编程注意:erase删除字符串中出现重叠的模式的第一个实例。例如:

erase ("abababa", "aba")

⇒ "b"

有关处理重叠，请参见strrep。

**参见:**strrep, regexprep。

: *[s, e, te, m, t, nm, sp] =* **regexp** *(str, pat)*

: *[…] =* **regexp** *(str, pat, "opt1", …)*

正则表达式字符串匹配。

在UTF-8编码的str中搜索pat并返回任何匹配的位置和子字符串，如果没有则返回空值。

匹配的模式部分可以包含任何标准正则表达式操作符，包括:

.

匹配任意字符

\* + ? {}

重复运算符，表示

\*

匹配0次或多次

+

匹配一次或多次

?

匹配0或1次

{n}

正好匹配n次

{n,}

匹配n次或更多次

{m,n}

匹配m到n次

[…] [^…]

运营商列表。该模式将匹配在“[”和“]”之间列出的任何字符。如果第一个字符是“^”，那么模式是颠倒的，除了括号之间列出的字符之外的任何字符都将匹配。

下面定义的转义序列也可以在列表操作符中使用。例如，浮点数的模板可能是[-+.\d]+。

() (?:)

分组操作符。第一种形式(只有括号)也创建了一个令牌。

|

交替操作符。匹配一个选择的正则表达式。备选项必须由上面的分组操作符()分隔。

^ $

锚定操作符。要求模式出现在字符串的开始(^)或结束($)处。

另外，下列转义字符有特殊含义。

\d

匹配任意数字

\D

匹配任何非数字

\s

匹配任何空白字符

\S

匹配任何非空白字符

\w

匹配任何单词字符

\W

匹配任何非单词字符

\<

匹配单词的开头

\>

匹配一个单词的结尾

\B

单词内匹配

实现注意:为了与MATLAB兼容，即使在用单引号定义了pat时，也会展开pat中的转义序列(例如，"\n" => newline)。要禁用扩展，请在转义序列(例如，"\\n")之前使用第二个反斜杠(例如，"\\n")或使用regexptranslate函数。

regexp的输出默认顺序如下所示

s

每个匹配子字符串的起始索引

e

每个匹配子字符串的结束索引

te

每个匹配令牌的范围在pat中被(…)包围

m

每个匹配项的文本的单元格数组

t

匹配的每个标记的文本的单元格数组

nm

一个结构，包含每个匹配的命名令牌的文本，名称用作字段名。命名令牌用(?<name>…)表示。

sp

没有由match返回的文本的单元格数组，即，如果您基于pat拆分字符串，剩下的内容。

可以通过附加的opt参数选择特定的输出参数或输出参数的顺序。这些都是字符串，输出参数和可选参数之间的对应关系是

'start' s

'end' e

'tokenExtents' te

'match' m

'tokens' t

'names' nm

'split' sp

其他参数总结如下。

‘once’

只返回模式的第一次出现。

‘matchcase’

使匹配区分大小写。(默认)

或者，在模式中使用(?-i)。

‘ignorecase’

将模式与字符串匹配时忽略大小写。

或者，在模式中使用(?i)。

‘stringanchors’

匹配字符串开头和结尾的锚字符。(默认)

或者，在模式中使用(?-m)。

‘lineanchors’

匹配行首和行尾的锚字符。

或者，在模式中使用(?m)。

‘dotall’

模式。匹配包括换行符在内的所有字符。(默认)

或者，在模式中使用(?s)。

‘dotexceptnewline’

模式。匹配除换行符以外的所有字符。

或者，在模式中使用(?-s)。

‘literalspacing’

模式中的所有字符(包括空格)都是重要的，并用于模式匹配。(默认)

或者，在模式中使用(?-x)。

‘freespacing’

该模式可以包括任意空格和以字符“#”开头的注释。

或者，在模式中使用(?x)。

‘noemptymatch’

不返回零长度匹配。(默认)

‘emptymatch’

返回零长度匹配。

Regexp ('a'， 'b\*'， 'emptymatch')返回[1 2]，因为在位置1和字符串结束处有零个或多个'b'字符。

注意:模式搜索是通过递归函数完成的，当有大量匹配时，递归函数可能会溢出程序堆栈。例如,

regexp (repmat ('a', 1, 1e5), '(a)+')

可能导致段故障。作为替代方案，可以考虑构造减少匹配数量的模式搜索(例如，创造性地使用set complement)，然后用连续的regexp搜索进一步处理返回变量(现在大小减少了)。

Octave 的正则表达式实现基于 Perl 兼容的 Regular Expres-sions 库 （https://www.pcre.org/）。有关正则表达式语法的更完整列表，请参阅PCRE 语法快速参考摘要。

**参见:**regexpi, strfind, regexprep。

: *[s, e, te, m, t, nm, sp] =* **regexpi** *(str, pat)*

: *[…] =* **regexpi** *(str, pat, "opt1", …)*

不区分大小写的正则表达式字符串匹配。

在UTF-8编码的str中搜索pat并返回任何匹配的位置和子字符串，如果没有则返回空值。有关搜索模式语法的详细信息，请参阅regexp。

**参见:**regexp。

: *outstr =* **regexprep** *(string, pat, repstr)*

: *outstr =* **regexprep** *(string, pat, repstr, "opt1", …)*

用repstr替换字符串中出现的pattern部分。

该模式是regexp文档中的正则表达式。看到regexp。

所有字符串必须是UTF-8编码。

替换字符串可以包含$i，它替换匹配字符串中的第i组括号。例如,

regexprep ("Bill Dunn", '(\w+) (\w+)', '$2, $1')

返回“邓恩，比尔”

除了regexp的选项外，还有

‘once’

只替换结果中第一次出现的pat。

‘warnings’

这个选项是为了兼容性而存在的，但是被忽略了。

实现注意:为了与MATLAB兼容，即使在用单引号定义了pat时，也会展开pat中的转义序列(例如，"\n" => newline)。要禁用扩展，请在转义序列(例如，"\\n")之前使用第二个反斜杠(例如，"\\n")或使用regexptranslate函数。

**参见:**regexp, regexpi, strrep。

: *str =* **regexptranslate** *(op, s)*

翻译字符串以用于正则表达式。

这可能包括通配符替换或特殊字符转义。

该行为由op控制，op可以接受以下值

"wildcard"

通配符。、\*和?被替换为适合正则表达式的通配符。例如:

regexptranslate ("wildcard", "\*.m")

⇒ '.\*\.m'

"escape"

字符$.?对于正则表达式具有特殊意义的[]，将被转义，以便按字面意思处理。例如:

Regexptranslate ("escape"， "12.5")

⇒“12 \ 0。5”

**参见:**regexp, regexpi, regexprep。

**5.4字符串转换**

Octave为字符串提供了几种转换函数。

**5.4.1字符串编码**

: *native\_bytes =* **unicode2native** *(utf8\_str, codepage)*

: *native\_bytes =* **unicode2native** *(utf8\_str)*

使用codepage将UTF-8字符串utf8\_str转换为字节流。

字符向量utf8\_str使用codepage给出的代码页转换为字节流native\_bytes。字符串代码页必须是有效代码页的标识符。有效的代码页示例有"ISO-8859-1"、"Shift-JIS"或"UTF-16"。有关支持的代码页列表，请参阅https://www.gnu.org/software/libiconv。如果codepage省略或为空，则使用系统默认的代码页。

如果任何字符不能映射到代码页代码页中，则用该代码页的适当替换序列替换它们。

**参见：**native2unicode。

: *utf8\_str =* **native2unicode** *(native\_bytes, codepage)*

: *utf8\_str =* **native2unicode** *(native\_bytes)*

使用codepage将字节流native\_bytes转换为UTF-8。

矢量native\_bytes中的数字被四舍五入并裁剪为0到255之间的整数。然后将这个字节流映射到字符串codepage给出的代码页，并以字符串utf8\_str返回。Octave使用UTF-8作为其内部编码。字符串代码页必须是有效代码页的标识符。有效的代码页示例有"ISO-8859-1"、"Shift-JIS"或"UTF-16"。有关支持的代码页列表，请参阅https://www.gnu.org/software/libiconv。如果codepage省略或为空，则使用系统默认的代码页。

如果native\_bytes是一个字符串向量，则按原样返回。

**参见：**unicode2native。

**5.4.2数值数据和字符串**

除了字符串连接函数(参见连接字符串)将数值数据转换为相应的UTF-8编码字符外，还有几个函数将数值数据格式化为字符串。Mat2str和num2str转换实数或复数矩阵，而int2str转换整数矩阵。Int2str取复数值的实部，并将分数值四舍五入为整数。将数值数据格式化为字符串的一种更灵活的方法是sprintf函数(参见格式化输出，sprintf)。

: *s =* **mat2str** *(x, n)*

: *s =* **mat2str** *(x, n, "class")*

将实矩阵、复杂矩阵和逻辑矩阵格式化为字符串。

返回的字符串可以通过使用eval函数来重建原始矩阵。

数值的精度由n给出。如果n是一个标量，则矩阵的实部和虚部都以相同的精度打印。否则，n(1)定义实部的精度，n(2)定义虚部的精度。n的默认值是15。

如果给出了参数“class”，则在字符串中包含x的类，这样eval将导致构造相同类的矩阵。

mat2str ([ -1/3 + i/7; 1/3 - i/7 ], [4 2])

⇒ "[-0.3333+0.14i;0.3333-0.14i]"

mat2str ([ -1/3 +i/7; 1/3 -i/7 ], [4 2])

⇒ "[-0.3333+0i 0+0.14i;0.3333+0i -0-0.14i]"

mat2str (int16 ([1 -1]), "class")

⇒ "int16([1 -1])"

mat2str (logical (eye (2)))

⇒ "[true false;false true]"

isequal (x, eval (mat2str (x)))

⇒ 1

**参见:**sprintf, num2str, int2str。

: *str =* **num2str** *(x)*

: *str =* **num2str** *(x, precision)*

: *str =* **num2str** *(x, format)*

将数字(或数组)转换为字符串(或字符数组)。

可选的第二个参数可以给出输出中要使用的有效位数(精度)，也可以像sprintf一样给出格式模板字符串(format)(参见格式化输出)。Num2str还可以处理复数。

例子:

num2str (123.456)

⇒ 123.456

num2str (123.456, 4)

⇒ 123.5

s = num2str ([1, 1.34; 3, 3.56], "%5.1f")

⇒ s =

1.0 1.3

3.0 3.6

whos s

⇒ Variables in the current scope:

Attr Name Size Bytes Class

==== ==== ==== ===== =====

s 2x8 16 char

Total is 16 elements using 16 bytes

num2str (1.234 + 27.3i)

⇒ 1.234+27.3i

num2str函数不是很灵活。为了更好地控制结果，请使用sprintf(参见格式化输出)。

编程指出:

为了与MATLAB兼容，在返回字符串之前会去掉前导空格。

大于flintmax的整数可能无法正确显示。

对于complex x，格式字符串可能只包含一个输出转换规范，不包含其他任何内容。否则，结果将不可预测。

程序员指定的任何可选格式都可以不加修改地使用。这与基于内部启发式的格式篡改的MATLAB形成对比。

**参见:**sprintf, int2str, mat2str。

: *str =* **int2str** *(n)*

将整数(或整数数组)转换为字符串(或字符数组)。

int2str (123)

⇒ 123

s = int2str ([1, 2, 3; 4, 5, 6])

⇒ s =

1 2 3

4 5 6

whos s

⇒ Variables in the current scope:

Attr Name Size Bytes Class

==== ==== ==== ===== =====

s 2x7 14 char

Total is 14 elements using 14 bytes

这个函数不是很灵活。为了更好地控制结果，请使用sprintf(参见格式化输出)。

编程指出:

非整数在显示前被四舍五入为整数。只显示复数的实部。

**参见:**sprintf, num2str, mat2str。

: *d =* **str2double** *(str)*

将字符串转换为实数或复数。

字符串必须是以下格式之一，其中a和b是实数，复数单位为'i'或'j':

a + bi

a + b\*i

a + i\*b

bi + a

b\*i + a

i\*b + a

如果存在，a和/或b的形式为[+-]d[，]。d[[eE][+-]d]其中括号表示可选参数，'d'表示零或多个数字。特殊的输入值Inf、NaN和NA也可以接受。

STR可以是字符串、字符矩阵或单元格数组。对于字符数组，每行重复转换，并返回双精度或复数数组。s中的空行将被删除，并且不会在数字数组中返回。对于单元格数组，将处理每个字符串元素，并返回与str具有相同维度的双精度或复杂数组。

对于不可转换的标量或字符串，输入str2double返回NaN。类似地，对于字符数组输入，str2double为任何不能转换的s行返回NaN。对于单元格数组，str2double返回s中任何转换失败的元素的NaN。注意，混合字符串/数字单元格数组中的数字元素不是字符串，这些元素的转换将失败并返回NaN。

编程注意:str2double可以代替str2num，效率更高，并且避免了对未知数据使用eval的安全风险。

**参见:**str2num。

: *x =* **str2num** *(s)*

: *[x, state] =* **str2num** *(s)*

将字符串(或字符数组)s转换为数字(或数组)。

例子:

str2num ("3.141596")

⇒ 3.141596

str2num (["1, 2, 3"; "4, 5, 6"])

⇒ 1 2 3

4 5 6

当转换成功时，可选的第二个输出state在逻辑上为真。如果转换失败，则数字输出x为空，state为false。

注意:由于str2num使用eval函数进行转换，str2num将执行字符串s中包含的任何代码。使用str2double进行更安全和更快的转换。

对于字符串单元格数组，使用str2double。

**参见:**str2double, eval。

: *d =* **bin2dec** *(str)*

返回与字符串str表示的二进制数相对应的十进制数。

例如:

bin2dec ("1110")

⇒ 14

空格在转换过程中被忽略，并可用于使二进制数更具可读性。

bin2dec ("1000 0001")

⇒ 129

如果str是字符串矩阵，返回一个列向量，每行str有一个转换后的数字;无效行求值为NaN。

如果str是字符串的单元格数组，则返回一个列向量，其中每个单元格元素有一个转换后的数字。

**参见:**dec2bin, base2dec, hex2dec。

: *bstr =* **dec2bin** *(d)*

: *bstr =* **dec2bin** *(d, len)*

返回一个由1和0组成的字符串，表示将整数d转换为二进制数。

如果d是矩阵或单元格数组，则返回一个字符串矩阵，其中d中的每个元素都有一行，并用前导零填充到最大值的宽度。

第二个可选参数len指定结果中的最小位数。

对于d的负元素，返回两者补码的二进制值。根据输入的幅度，结果被填充到8、16、32或64位。正输入元素用前导零填充到相同的宽度。

例子:

dec2bin (14)

⇒ "1110"

dec2bin (-14)

⇒ "11110010"

编程提示：dec2bin 丢弃输入的任何小数部分。如果您还需要转换小数部分，请使用小数点后非零位数调用 dec2base。您还可以对小数输入使用固定或舍入，以确保可预测的舍入行为。

**参见:**bin2dec, dec2base, dec2hex。

: *hstr =* **dec2hex** *(d)*

: *hstr =* **dec2hex** *(d, len)*

返回一个字符串，表示将整数d转换为十六进制(base16)数。

如果 d 为负数，则返回 d 的十六进制补码。

如果 d 是矩阵或元胞数组，则返回一个字符串矩阵，其中 d 中的每个元素都有一行，并用前导零填充到最大值的宽度。

可选的第二个参数 len 指定结果中的最小位数。例子:

dec2hex (2748)

⇒ "ABC"

dec2hex (-2)

⇒ "FE"

编程提示：dec2hex 丢弃输入的任何小数部分。如果您还需要转换小数部分，请使用小数点后非零位数调用 dec2base。您还可以对小数输入使用固定或舍入，以确保可预测的舍入行为。

**参见:**hex2dec, dec2base, dec2bin。

: *d =* **hex2dec** *(str)*

返回对应于字符串str的十六进制数字的整数。

例如:

hex2dec ("12B")

⇒ 299

hex2dec ("12b")

⇒ 299

如果str是字符串矩阵，返回一个列向量，每行str有一个转换后的数字;无效行求值为NaN。

如果str是字符串的单元格数组，则返回一个列向量，其中每个单元格元素有一个转换后的数字。

**参见:**dec2hex, base2dec, bin2dec。

: *str =* **dec2base** *(d, base)*

: *str =* **dec2base** *(d, base, len)*

: *str* *=* **dec2base** *(d, base, len, decimals)*

返回一串 base 中对应于值 d 的符号。

dec2base (123, 3)

⇒ "11120"

如果 d 为负数，则结果将以补码表示法表示 d。例如，负二进制数是二进制补码，其他基数也是如此。

如果 d 是矩阵或元内存组，则返回一个字符串矩阵，其中包含 d 中每个元素一行，并用前导零填充到最大值的宽度。

如果 base 是字符串，则 base 的字符用作 d 数字的符号。空格（空格、制表符、换行符等）不能用作符号。

dec2base (123, "aei")

⇒ "eeeia"

可选的第三个参数 len 指定结果整数部分的最小位数。如果省略这一点，则 dec2base 使用足够的数字来容纳输入。

可选的第四个参数 decimals 指定表示输入小数部分的位数。如果省略此项，则将其设置为零，并且 dec2base 返回一个整数输出以实现向后兼容性。

dec2base (100\*pi, 16)

⇒ "13A"

dec2base (100\*pi, 16, 4)

⇒ "013A"

dec2base (100\*pi, 16, 4, 6)

⇒ "013A.28C59D"

dec2base (-100\*pi, 16)

⇒ "EC6"

dec2base (-100\*pi, 16, 4)

⇒ "FEC6"

dec2base (-100\*pi, 16, 4, 6)

⇒ "FEC5.D73A63"

编程提示：将负输入传递给 dec2base 时，最好显式指定所需输出的长度。

**参见:**base2dec, dec2bin, dec2hex。

: *d =* **base2dec** *(str, base)*

将str从以base为基数的数字字符串转换为十进制整数(以10为基数)。

base2dec ("11120", 3)

⇒ 123

如果str是字符串矩阵，则返回一个列向量，每行str中有一个值。如果一行包含无效符号，则对应的值将为NaN。

如果str是字符串的单元格数组，则返回一个列向量，其中每个单元格元素有一个值。

如果base是字符串，则base的字符用作str中数字的符号。空格(' ')不能用作符号。

base2dec ("yyyzx", "xyz")

⇒ 123

**参见:**dec2base, bin2dec, hex2dec。

: *s =* **num2hex** *(n)*

: *s =* **num2hex** *(n, "cell")*

将数字数组转换为十六进制字符串数组。

例如:

num2hex ([-1, 1, e, Inf])

⇒ "bff0000000000000

3ff0000000000000

4005bf0a8b145769

7ff0000000000000"

如果参数n是单精度数字或向量，则返回的字符串长度为8。例如:

num2hex (single ([-1, 1, e, Inf]))

⇒ "bf800000

3f800000

402df854

7f800000"

使用可选的第二个参数"cell"，返回字符串的单元格数组，而不是字符数组。

**参见:**hex2num, hex2dec, dec2hex。

: *n =* **hex2num** *(s)*

: *n =* **hex2num** *(s, class)*

将十六进制字符数组或字符串单元格数组类型转换为数字数组。

默认情况下，输入数组被解释为表示双精度值的十六进制数。如果给定的字符少于16个，则用'0'字符右填充字符串。

给定一个字符串矩阵，hex2num将每一行视为一个单独的数字。

hex2num (["4005bf0a8b145769"; "4024000000000000"])

⇒ [2.7183; 10.000]

选的第二个参数类可用于将输入数组解释为不同的值类型。可能的值有

Option Characters

"int8" 2

"uint8" 2

"int16" 4

"uint16" 4

"int32" 8

"uint32" 8

"int64" 16

"uint64" 16

"char" 2

"single" 8

"double" 16

例如：

hex2num (["402df854"; "41200000"], "single")

⇒ [2.7183; 10.000]

**参见:**num2hex, hex2dec, dec2hex。

: *[a, …] =* **strread** *(str)*

: *[a, …] =* **strread** *(str, format)*

: *[a, …] =* **strread** *(str, format, format\_repeat)*

: *[a, …] =* **strread** *(str, format, prop1, value1, …)*

: *[a, …] =* **strread** *(str, format, format\_repeat, prop1, value1, …)*

这个功能已经过时了。使用textcan代替。

从字符串中读取数据。

字符串str被分割成单词，这些单词在格式中与说明符重复匹配。第一个单词与第一个说明符匹配，第二个与第二个说明符匹配，依此类推。如果单词多于说明符，则重复该过程，直到处理完所有单词。

字符串格式描述了应该如何解析str中的单词。它可以包含以下说明符的任意组合:

%s

该单词被解析为字符串。

%f

%n

该单词被解析为数字并转换为双精度。

%d

%u

该单词被解析为数字并转换为int32。

%\*

%\*f

%\*s

这个词被跳过了。

对于%s和%d， %f， %n， %u和关联的%\*s…指定符，可选的宽度可以指定为%Ns等，其中n是> 1的整数。对于%f，使用%N这样的格式说明符。Mf是允许的。

literals

此外，格式可以包含文字字符串;这些在阅读时将被跳过。

与第一个说明符对应的解析词将在第一个输出参数中返回，其余说明符也是如此。

默认情况下，format为"%f"，这意味着从str读取数字。如果str仅包含数字字段，则会执行此操作。

例如，字符串

str = "\

Bunny Bugs 5.5\n\

Duck Daffy -7.5e-5\n\

Penguin Tux 6"

可以使用

[a, b, c] = strread (str, "%s %s %f");

可选的数字参数format\_repeat可用于限制读取的项数:

-1

(默认)读取所有字符串直到末尾。

N

读N遍旁白。0(零)是format\_repeat可以接受的值。

strread的行为可以通过属性-值对来改变。识别以下属性:

"commentstyle"

str的某些部分被视为注释，将被跳过。Value是注释样式，可以是以下任意值。

"shell"跳过从#字符到最近的行尾的所有内容。

"c" /\*和\*/之间的所有内容都被跳过。

"c++"跳过从//字符到最近的行尾的所有内容。

从%字符到最近的行尾的所有内容都被跳过。

用户提供的。两个选项:(1)一个字符串，或1x1单元格字符串:跳过它右边的所有内容;(2) 2x1单元格字符串数组:跳过左右字符串之间的所有内容。

"delimiter"

value中的任何字符都将用于将str拆分为单词(默认值=任何空格)。请注意，除非提供了“%s”格式转换说明符，否则空格会隐式地添加到分隔符集中;请参阅下面的“whitespace”参数。分隔符集不能为空;如果需要，Octave将替换一个空格作为分隔符。

"emptyvalue"

为非空格分隔的数据中的空数值返回的值。默认为NaN。当数据类型不支持NaN(例如int32)时，则默认为零。

"multipledelimsasone"

将一系列连续的分隔符(中间没有空格)视为单个分隔符。连续的分隔符系列不需要垂直“对齐”。

"treatasempty"

将value中字符串的单次出现(由分隔符或空格包围)视为缺失值。

"returnonerror"

如果value为true(1，默认值)，则忽略读取错误并正常返回。如果为false(0)，则返回错误。

"whitespace"

value中的任何字符都将被解释为空白并被修剪;定义空格的字符串必须用双引号括起来，以便正确处理像“\t”这样的特殊字符。在每个数据字段中，多个连续的空白字符被折叠成一个空格，并删除前导和尾随的空白。空格的默认值是" \b\r\n\t"(注意空格)。除非提供了至少一个“%s”格式转换说明符，否则总是将空格添加到分隔符集中;在这种情况下，只有在"delimiter"中明确指定的空白被保留为分隔符，并从空白字符集中删除。如果空白字符要保持原样(例如，字符串)，请为"whitespace"指定一个空值(即"");显然，空白不能作为分隔符。

当str中的字数不匹配格式转换说明符数的精确倍数时，strread的行为取决于str的最后一个字符:

last character = "\n"

用空字段或NaN填充数据列，以便所有列具有相同的长度

last character is not "\n"

数据列没有填充;Strread返回长度不等的列

**参见:**textscan, sscanf。

**5.4.3 JSON数据编码/解码**

JavaScript对象表示法，简称JSON，是一种非常常见的人类可读的结构化数据格式。GNU Octave通过以下两个函数支持对这种格式进行编码和解码。

: *JSON\_txt =* **jsonencode** *(object)*

: *JSON\_txt =* **jsonencode** *(…, "ConvertInfAndNaN", TF)*

: *JSON\_txt =* **jsonencode** *(…, "PrettyPrint", TF)*

将Octave数据类型编码为JSON文本。

输入对象是一个要编码的Octave变量。

输出JSON\_txt是包含编码对象结果的JSON文本。

如果ConvertInfAndNaN选项的值为true，则输出中的NaN、NA、-Inf和Inf values将被转换为“null”。如果为false，则它们将保持其原始值。该选项的默认值为true。

如果“PrettyPrint”选项的值为true，则输出文本将具有缩进和换行。如果为false，输出将被压缩，并且写入时不带空格。该选项的默认值为false。

编程指出:

不支持复数。

Classdef对象首先转换为结构体，然后进行编码。

要保留转义字符(例如“\n”)，请使用单引号字符串。

在双引号字符串中，空字符(“\0”)之后的每个字符将在编码期间被丢弃。

对数组进行编码和解码并不能保证保持数组的尺寸。特别地，行向量将被重塑为列向量。

编码和解码不能保证保留Octave数据类型，因为JSON支持的数据类型比Octave少。例如，如果您编码一个int8，然后解码它，您将得到一个双精度。

下表显示了从Octave数据类型到JSON数据类型的转换:

| **Octave data type** | **JSON data type** |
| --- | --- |
| logical scalar | Boolean |
| logical vector | Array of Boolean, reshaped to row vector |
| logical array | nested Array of Boolean |
| numeric scalar | Number |
| numeric vector | Array of Number, reshaped to row vector |
| numeric array | nested Array of Number |
| *NaN*, *NA*, *Inf*, *-Inf* when *"ConvertInfAndNaN" = true* | *"null"* |
| *NaN*, *NA*, *Inf*, *-Inf* when *"ConvertInfAndNaN" = false* | *"NaN"*, *"NaN"*, *"Infinity"*, *"-Infinity"* |
| empty array | *"[]"* |
| character vector | String |
| character array | Array of String |
| empty character array | *""* |
| cell scalar | Array |
| cell vector | Array, reshaped to row vector |
| cell array | Array, flattened to row vector |
| struct scalar | Object |
| struct vector | Array of Object, reshaped to row vector |
| struct array | nested Array of Object |
| classdef object | Object |

例子：

jsonencode ([1, NaN; 3, 4])

⇒ [[1,null],[3,4]]

jsonencode ([1, NaN; 3, 4], "ConvertInfAndNaN", false)

⇒ [[1,NaN],[3,4]]

## Escape characters inside a single-quoted string

jsonencode ('\0\a\b\t\n\v\f\r')

⇒ "\\0\\a\\b\\t\\n\\v\\f\\r"

## Escape characters inside a double-quoted string

jsonencode ("\a\b\t\n\v\f\r")

⇒ "\u0007\b\t\n\u000B\f\r"

jsonencode ([true; false], "PrettyPrint", true)

⇒ ans = [

true,

false

]

jsonencode (['foo', 'bar'; 'foo', 'bar'])

⇒ ["foobar","foobar"]

jsonencode (struct ('a', Inf, 'b', [], 'c', struct ()))

⇒ {"a":null,"b":[],"c":{}}

jsonencode (struct ('structarray', struct ('a', {1; 3}, 'b', {2; 4})))

⇒ {"structarray":[{"a":1,"b":2},{"a":3,"b":4}]}

jsonencode ({'foo'; 'bar'; {'foo'; 'bar'}})

⇒ ["foo","bar",["foo","bar"]]

jsonencode (containers.Map({'foo'; 'bar'; 'baz'}, [1, 2, 3]))

⇒ {"bar":2,"baz":3,"foo":1}

**参见:**jsondecode。

: *object =* **jsondecode** *(JSON\_txt)*

: *object =* **jsondecode** *(…, "ReplacementStyle", rs)*

: *object =* **jsondecode** *(…, "Prefix", pfx)*

: *object =* **jsondecode** *(…, "makeValidName", TF)*

解码JSON格式的文本。

输入的JSON\_txt是一个包含JSON文本的字符串。

输出对象是一个Octave对象，其中包含解码JSON\_txt的结果。

有关选项“ReplacementStyle”和“Prefix”的更多信息，请参见matlab.lang.makeValidName。

如果选项"makeValidName"的值为false，则名称将不会被matlab.lang.makeValidName更改，并且"ReplacementStyle"和"Prefix"选项将被忽略。

注意:解码和编码JSON文本不能保证复制原始文本，因为一些名称可能会被matlab.lang.makeValidName更改。

下表显示了从JSON数据类型到Octave数据类型的转换:

| **JSON data type** | **Octave data type** |
| --- | --- |
| Boolean | scalar logical |
| Number | scalar double |
| String | vector of characters |
| Object | scalar struct (field names of the struct may be different from the keys of the JSON object due to *matlab\_lang\_makeValidName* |
| null, inside a numeric array | *NaN* |
| null, inside a non-numeric array | empty double array *[]* |
| Array, of different data types | cell array |
| Array, of Booleans | logical array |
| Array, of Numbers | double array |
| Array, of Strings | cell array of character vectors (*cellstr*) |
| Array of Objects, same field names | struct array |
| Array of Objects, different field names | cell array of scalar structs |

例子：

jsondecode ('[1, 2, null, 3]')

⇒ ans =

1

2

NaN

3

jsondecode ('["foo", "bar", ["foo", "bar"]]')

⇒ ans =

{

[1,1] = foo

[2,1] = bar

[3,1] =

{

[1,1] = foo

[2,1] = bar

}

}

jsondecode ('{"nu#m#ber": 7, "s#tr#ing": "hi"}', ...

'ReplacementStyle', 'delete')

⇒ scalar structure containing the fields:

number = 7

string = hi

jsondecode ('{"nu#m#ber": 7, "s#tr#ing": "hi"}', ...

'makeValidName', false)

⇒ scalar structure containing the fields:

nu#m#ber = 7

s#tr#ing = hi

jsondecode ('{"1": "one", "2": "two"}', 'Prefix', 'm\_')

⇒ scalar structure containing the fields:

m\_1 = one

m\_2 = two

**参见:**jsonencode, matlab.lang.makeValidName。

**5.5字符类函数**

Octave还提供了以下字符类测试函数，这些函数以标准C库中的函数为模板。它们都对字符串数组进行操作，并返回0和1的矩阵。非零元素表示字符串数组中对应字符的条件为真。例如:

isalpha ("!Q@WERT^Y&")

⇒ [ 0, 1, 0, 1, 1, 1, 1, 0, 1, 0 ]

: *tf =* **isalnum** *(s)*

返回一个逻辑数组，如果s中的元素是字母或数字，则为真;如果不是，则为假。

这相当于(isalpha (s) | isdigit (s))。

参见:isalpha, isdigit, ispunct, isspace, isccontrol。

: *tf =* **isalpha** *(s)*

返回一个逻辑数组，当s中的元素为字母时为真，当元素为非字母时为假。

这相当于(islower (s) | isupper (s))。

参见:isdigit, ispunct, isspace, iscentral, isalnum, islower, isupper。

: *tf =* **isletter** *(s)*

返回一个逻辑数组，当s中的元素为字母时为真，当元素为非字母时为假。

这是isalpha函数的别名。

参见:isalpha, isdigit, ispunct, isspace, iscentral, isalnum。

: *tf =* **islower** *(s)*

返回一个逻辑数组，如果s中的元素是小写字母，则为真;如果不是小写字母，则为假。

参见:isupper, isalpha, isletter, isalnum。

: *tf =* **isupper** *(s)*

返回一个逻辑数组，如果s中的元素是大写字母，则为true;如果不是大写字母，则为false。

参见:islower, isalpha, isletter, isalnum。

: *tf =* **isdigit** *(s)*

返回一个逻辑数组，如果s中的元素是十进制数字(0-9)，则为true;如果不是十进制数字，则为false。

参见:isxdigit, isalpha, isletter, ispunct, isspace, isccontrol。

: *tf =* **isxdigit** *(s)*

返回一个逻辑数组，该数组为true，其中s的元素为十六进制数字(0-9和a- fa -f)。

参见:isdigit。

: *tf =* **ispunct** *(s)*

返回一个逻辑数组，如果s中的元素是标点符号，则为真，如果不是，则为假。

参见:isalpha, isdigit, isspace, isccontrol。

: *tf =* **isspace** *(s)*

返回一个逻辑数组，如果s中的元素是空白字符(空格、换行、换行、回车、制表符和垂直制表符)，则为真，如果不是，则为假。

参见:iscentral, ispunct, isalpha, isdigit。

: *tf =* **iscntrl** *(s)*

返回一个逻辑数组，如果s中的元素是控制字符，则为true;如果不是，则为false。

参见:ispunct, isspace, isalpha, isdigit。

: *tf =* **isgraph** *(s)*

返回一个逻辑数组，如果s中的元素是可打印字符(但不是空格字符)，则为true;如果不是，则为false。

参见:isprint。

: *tf =* **isprint** *(s)*

返回一个逻辑数组，如果s中的元素是可打印字符(包括空格字符)，则为true，如果不是，则为false。

参见:isgraph。

: *tf =* **isascii** *(s)*

返回一个逻辑数组，如果s中的元素是ASCII字符(十进制范围为0到127)，则为true，如果不是false。

: *tf =* **isstrprop** *(str, prop)*

*: tf =* **isstrprop** *(str, prop, 'ForceCellOutput', flag)*

测试字符串属性。

例如:

isstrprop ("abc123", "alpha")

⇒ [1, 1, 1, 0, 0, 0]

如果str是一个单元格数组，则isstrpop将递归地应用于单元格数组的每个元素。

数值数组被转换为字符串。

第二个参数prop必须是其中之一

"alpha"

对于字母字符(字母)为True。

"alnum"

"alphanum"

对于字母或数字字符为True。

"lower"

对于小写字母为True。

"upper"

对大写字母为True。

"digit"

对于十进制数字(0-9)为True。

"xdigit"

对于十六进制数字(a-fA-F0-9)为True。

"space"

"wspace"

对于空白字符(空格、换行、换行、回车、制表符、垂直制表符)为True。

"punct"

对于标点字符(打印除空格、字母或数字以外的字符)为True。

"cntrl"

对于控制字符为True。

"graph"

"graphic"

用于打印除空格以外的字符。

"print"

为打印字符(包括空格)。

"ascii"

对于ASCII编码范围内的字符为True。

如果给定了选项“ForceCellOutput”并且 flag 为 true，则返回单元格值而不是逻辑数组。

**参见:**isalpha, isalnum, islower, isupper, isdigit, isxdigit, isspace, ispunct, iscentral, isgraph, isprint, isascii。

**6数据容器**

Octave支持三种不同的机制来在同一个变量中包含任意数据类型:结构，它是类似c的，并且用命名字段进行索引;容器。Map对象，它以键/值对的形式存储数据;单元格数组，其中数组的每个元素可以具有不同的数据类型和/或形状。函数的多个输入参数和返回值被组织成另一个数据容器，即逗号分隔的列表。

**6.1结构**

Octave支持在结构中组织数据。当前的实现使用关联数组，索引仅限于字符串，但语法更像c风格的结构。

**6.1.1基本用法和示例**

下面是一些在Octave中使用数据结构的例子。

结构的元素可以是任何值类型。例如，这三个表达式

x.a = 1;

x.b = [1, 2; 3, 4];

x.c = "string";

创建一个包含三个元素的结构。”。'字符将结构名(在上面的例子x中)与字段名分开，并向Octave表明该变量是一个结构。要打印结构的值，你可以输入它的名字，就像其他变量一样:

x

⇒ x =

scalar structure containing the fields:

a = 1

b =

1 2

3 4

c = string

注意，Octave可以以任何顺序打印元素。

结构可以像其他变量一样被复制:

y = x

⇒ y =

scalar structure containing the fields:

a = 1

b =

1 2

3 4

c = string

因为结构本身就是值，所以结构元素也可以引用其他结构。下面的语句将字段d添加到结构x中。字段d的值本身是一个包含单个字段a的数据结构，其值为3。

x.d.a = 3;

x.d

⇒ ans =

scalar structure containing the fields:

a = 3

x

⇒ x =

scalar structure containing the fields:

a = 1

b =

1 2

3 4

c = string

d =

scalar structure containing the fields:

a = 3

注意，当Octave打印包含其他结构的结构的值时，只显示几个级别。例如:

a.b.c.d.e = 1;

a

⇒ a =

scalar structure containing the fields:

b =

scalar structure containing the fields:

c =

scalar structure containing the fields:

d: 1x1 scalar struct

这可以防止大型深嵌套结构的长而混乱的输出。可以使用函数struct\_levels\_to\_print设置嵌套结构的打印层数，并且可以使用函数print\_struct\_array\_contents来启用打印结构数组的内容。

: *val =* **struct\_levels\_to\_print** *()*

: *old\_val =* **struct\_levels\_to\_print** *(new\_val)*

: *old\_val =* **struct\_levels\_to\_print** *(new\_val, "local")*

查询或设置指定要显示的结构级别数量的内部变量。

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。

**参见:**print\_struct\_array\_contents。

: *val =* **print\_struct\_array\_contents** *()*

: *old\_val =* **print\_struct\_array\_contents** *(new\_val)*

: *old\_val =* **print\_struct\_array\_contents** *(new\_val, "local")*

查询或设置指定是否打印结构数组内容的内部变量。

如果为true，则打印结构数组元素的值。此变量不影响始终打印元素的标量结构。然而，在这两种情况下，打印将被限制为struct\_levels\_to\_print指定的级别数。

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:**struct\_levels\_to\_print。

函数可以返回结构。例如，下面的函数分离矩阵的实部和复部，并将它们存储在具有相同结构变量y的两个元素中。

function y = f (x)

y.re = real (x);

y.im = imag (x);

endfunction

当使用复值参数调用时，函数f返回包含原始函数参数实部和虚部的数据结构。

f (rand (2) + rand (2) \* I)

⇒ ans =

scalar structure containing the fields:

re =

0.040239 0.242160

0.238081 0.402523

im =

0.26475 0.14828

0.18436 0.83669

函数返回列表可以包含结构元素，并且它们可以像任何其他变量一样被索引。例如:

[ x.u, x.s(2:3,2:3), x.v ] = svd ([1, 2; 3, 4]);

x

⇒ x =

scalar structure containing the fields:

u =

-0.40455 -0.91451

-0.91451 0.40455

s =

0.00000 0.00000 0.00000

0.00000 5.46499 0.00000

0.00000 0.00000 0.36597

v =

-0.57605 0.81742

-0.81742 -0.57605

使用特殊形式的for语句(参见遍历结构元素)，也可以在循环中遍历结构的所有元素。

**6.1.2结构数组**

结构数组是结构的特定实例，其中结构的每个字段都由单元格数组表示。每个单元格数组都有相同的维度。从概念上讲，结构数组也可以看作是具有相同字段的结构数组。创建结构数组的一个示例如下

x(1).a = "string1";

x(2).a = "string2";

x(1).b = 1;

x(2).b = 2;

它创建了一个包含两个字段的1 × 2结构数组。另一种创建结构数组的方法是使用struct函数(参见创建结构)。和前面一样，要打印结构数组的值，你可以输入它的名字:

x

⇒ x =

{

1x2 struct array containing the fields:

a

b

}

结构数组的各个元素可以通过像x(1)这样的索引变量来返回，它返回一个包含两个字段的结构:

x(1)

⇒ ans =

{

a = string1

b = 1

}

此外，如果由它自己的一个字段名索引，结构数组可以返回一个逗号分隔的字段值列表(参见逗号分隔列表)。例如:

x.a

⇒

ans = string1

ans = string2

下面是另一个例子，在赋值语句的左侧使用逗号分隔的列表:

[x.a] = deal ("new string1", "new string2");

x(1).a

⇒ ans = new string1

x(2).a

⇒ ans = new string2

就像数值数组一样，可以使用向量作为索引(参见索引表达式):

x(3:4) = x(1:2);

[x([1,3]).a] = deal ("other string1", "other string2");

x.a

⇒

ans = other string1

ans = new string2

ans = other string2

ans = new string2

函数size将返回结构的大小。对于上面的例子

size (x)

⇒ ans =

1 4

可以用与数字数组类似的方式从结构数组中删除元素，方法是将元素赋值给空矩阵。例如

in = struct ("call1", {x, Inf, "last"},

"call2", {x, Inf, "first"})

⇒ in =

{

1x3 struct array containing the fields:

call1

call2

}

in(1) = [];

in.call1

⇒

ans = Inf

ans = last

**6.1.3创建结构**

除了索引操作符“。”，Octave还可以使用动态命名“(var)”或struct函数来创建结构。动态命名使用变量的字符串值作为字段名。例如:

a = "field2";

x.a = 1;

x.(a) = 2;

x

⇒ x =

{

a = 1

field2 = 2

}

动态索引还允许您使用任意字符串，而不仅仅是有效的Octave标识符(注意，这在MATLAB上不起作用):

a = "long field with spaces (and funny char$)";

x.a = 1;

x.(a) = 2;

x

⇒ x =

{

a = 1

long field with spaces (and funny char$) = 2

}

可以启用警告id Octave:language-extension来警告这种用法。参见warning\_ids。

更实际的情况是，所有对字符串进行操作的函数都可以在将字段名输入到数据结构之前构建正确的字段名。

names = ["Bill"; "Mary"; "John"];

ages = [37; 26; 31];

for i = 1:rows (names)

database.(names(i,:)) = ages(i);

endfor

database

⇒ database =

{

Bill = 37

Mary = 26

John = 31

}

创建结构的第三种方法是struct命令。Struct接受一对参数，其中对中的第一个参数是要包含在结构中的字段名，第二个参数是一个标量或单元格数组，表示要包含在结构或结构数组中的值。例如:

struct ("field1", 1, "field2", 2)

⇒ ans =

{

field1 = 1

field2 = 2

}

如果传递给struct的值是标量数组和单元格数组的混合，则扩展标量参数以创建具有一致维度的结构数组。例如:

s = struct ("field1", {1, "one"}, "field2", {2, "two"},

"field3", 3);

s.field1

⇒

ans = 1

ans = one

s.field2

⇒

ans = 2

ans = two

s.field3

⇒

ans = 3

ans = 3

如果你想创建一个结构体，其中包含一个单元格数组作为单独的字段，你必须把它包装在另一个单元格数组中，如下面的例子所示:

struct ("field1", {{1, "one"}}, "field2", 2)

⇒ ans =

{

field1 =

{

[1,1] = 1

[1,2] = one

}

field2 = 2

}

: *s =* **struct** *()*

: *s =* **struct** *(field1, value1, field2, value2, …)*

: *s =* **struct** *(obj)*

创建标量或数组结构并初始化其值。

变量field1, field2，…是指定字段名称的字符串，变量value1, value2，…可以是任何类型。

如果值是单元格数组，则创建一个结构数组并初始化其值。每个单元格值数组的尺寸必须匹配。单单元格和非单元格值重复，以便它们填充整个数组。如果单元格为空，则使用指定的字段名创建空结构数组。

如果参数是一个对象，则返回底层结构。

注意，语法针对结构数组进行了优化。考虑下面的例子:

struct ("foo", 1)

⇒ scalar structure containing the fields:

foo = 1

struct ("foo", {})

⇒ 0x0 struct array containing the fields:

foo

struct ("foo", { {} })

⇒ scalar structure containing the fields:

foo = {}(0x0)

struct ("foo", {1, 2, 3})

⇒ 1x3 struct array containing the fields:

foo

第一种情况是一个普通的标量结构——一个字段，一个值。第二个函数产生一个空的结构数组，只有一个字段，没有值，因为它传递了一个结构数组值的空单元格数组。当值是包含单个条目的单元格数组时，它将变成一个标量结构体，以该单个条目作为字段的值。这个单条目恰好是一个空单元格数组。

最后，如果值是一个非标量单元数组，那么struct将生成一个struct数组。

**参见:**cell2struct, fieldnames, getfield, setfield, rmfield, isfield, orderfields, isstruct, structfun。

函数isstruct可用于测试对象是结构体还是结构体数组。

: *tf =* **isstruct** *(x)*

如果x是结构体或结构数组则返回true。

**参见:**ismatrix, iscell, isa。

**6.1.4操纵结构**

下面给出了可以操作结构的字段的其他函数。

: *n =* **numfields** *(s)*

返回结构s的字段数。

参见:fieldnames。

: *names =* **fieldnames***(struct)*

: *names =* **fieldnames** *(obj)*

: *names =* **fieldnames** *(javaobj)*

: *names =* **fieldnames** *("javaclassname")*

返回字符串的单元格数组，其中包含指定输入中字段的名称。

当输入是一个结构结构体时，名称就是结构体的元素。

当输入是Octave对象obj时，名称是该对象的公共属性。

当输入是Java对象javaobj或包含Java类javaclassname的字符串时，这些名称是对象或类的公共字段(数据成员)。

**参见:**numfields, isfield, orderfields, struct, properties。

: *tf =* **isfield** *(x, "name")*

: *tf =* **isfield** *(x, name)*

如果x是一个结构体，并且它包含一个名为name的元素，则返回true。

如果name是字符串的单元格数组，则返回一个等维的逻辑数组。

参见:fieldnames。

: *sout =* **setfield** *(s, field, val)*

: *sout =* **setfield** *(s, sidx1, field1, fidx1, sidx2, field2, fidx2, …, val)*

返回结构s的副本，其中字段成员字段设置为值val。

例如:

s = struct ();

s = setfield (s, "foo bar", 42);

这相当于

s.("foo bar") = 42;

注意，普通的结构语法s.foo bar = 42不能在这里使用，因为字段名不是一个有效的Octave标识符，因为有空格字符。使用任意字符串作为字段名与MATLAB不兼容，如果启用了警告ID Octave:language-extension，这种用法将发出警告。参见warning\_ids。

使用第二个调用表单，设置结构数组的字段。输入sidx选择结构数组的一个元素，field指定所选元素的字段名，fidx选择字段中的哪个元素(对于数组或单元格数组)。可以重复输入sidx、field和fidx来寻址嵌套结构数组元素。结构数组索引和字段元素索引必须是单元格数组，而字段名称必须是字符串。

例如:

s = struct ("baz", 42);

setfield (s, {1}, "foo", {1}, "bar", 54)

⇒

ans =

scalar structure containing the fields:

baz = 42

foo =

scalar structure containing the fields:

bar = 54

该示例从一个普通标量结构开始，其中添加了一个嵌套标量结构。在所有情况下，如果没有指定结构索引sidx，它默认为1(标量结构)。因此，上面的例子可以更简洁地写成setfield (s， "foo"， "bar"， 54)

最后，一个嵌套结构数组的例子:

sa.foo = 1;

sa = setfield (sa, {2}, "bar", {3}, "baz", {1, 4}, 5);

sa(2).bar(3)

⇒

ans =

scalar structure containing the fields:

baz = 0 0 0 5

这里sa是一个结构数组，其元素1和2处的字段依次是另一个结构数组，其第三个元素是一个简单的标量结构。终端标量结构有一个包含矩阵值的字段。

请注意，可以通过以下方式获得与上述示例相同的结果:

sa.foo = 1;

sa(2).bar(3).baz(1,4) = 5

**参见:**getfield, rmfield, orderfields, isfield, fieldnames, isstruct, struct。

: *val =* **getfield** *(s, field)*

: *val =* **getfield** *(s, sidx1, field1, fidx1, …)*

从结构或嵌套结构中获取名为field的字段的值。

如果s是一个结构数组，则sidx选择结构数组中的一个元素，field指定所选元素的字段名，fidx选择字段中的哪个元素(对于数组或单元格数组)。有关该语法的更完整描述，参见setfield。

**参见:**setfield, rmfield, orderfields, isfield, fieldnames, isstruct, struct。

: *sout =* **rmfield** *(s, "f")*

: *sout =* **rmfield** *(s, f)*

返回移除了字段f的结构(数组)s的副本。

如果f是字符串或字符数组的单元格数组，则删除每个命名字段。

**参见:**orderfields, fieldnames, isfield。

: *sout =* **orderfields** *(s1)*

: *sout =* **orderfields** *(s1, s2)*

: *sout =* **orderfields** *(s1, {cellstr})*

: *sout =* **orderfields** *(s1, p)*

: *[sout, p] =* **orderfields** *(…)*

返回s1的副本，其中字段按字母顺序排列，或按第二个输入指定的顺序排列。

给定一个输入结构体s1，按字母顺序排列字段名。

如果给出了第二个结构体参数，则按照s2中的字段名排列s1中的字段名。第二个参数还可以指定字符串单元格数组cellstr中的顺序。第二个参数也可以是一个置换向量。

可选的第二个输出参数p是将原始名称顺序转换为新名称顺序的排列向量。

例子:

s = struct ("d", 4, "b", 2, "a", 1, "c", 3);

t1 = orderfields (s)

⇒ t1 =

scalar structure containing the fields:

a = 1

b = 2

c = 3

d = 4

t = struct ("d", {}, "c", {}, "b", {}, "a", {});

t2 = orderfields (s, t)

⇒ t2 =

scalar structure containing the fields:

d = 4

c = 3

b = 2

a = 1

t3 = orderfields (s, [3, 2, 4, 1])

⇒ t3 =

scalar structure containing the fields:

a = 1

b = 2

c = 3

d = 4

[t4, p] = orderfields (s, {"d", "c", "b", "a"})

⇒ t4 =

scalar structure containing the fields:

d = 4

c = 3

b = 2

a = 1

p =

1

4

2

3

**参见:**fieldnames, getfield, setfield, rmfield, isfield, isstruct, struct。

: *s =* **substruct** *(type, subs, …)*

创建用于subsref或subsassign的下标结构。

例如:

idx = substruct ("()", {3, ":"})

⇒ idx =

scalar structure containing the fields:

type = ()

subs =

{

[1,1] = 3

[1,2] = :

}

x = [1, 2, 3;

4, 5, 6;

7, 8, 9];

subsref (x, idx)

⇒ 7 8 9

注意:关键字 end 不能在 subsref 或 subsasgn 中用于索引分配。

**参见:**subsref, subsassign。

**6.1.5处理结构中的数据**

在结构中处理数据的最简单方法是在for循环中(参见在结构元素上循环)。使用structfun函数可以实现类似的效果，其中用户定义的函数应用于结构的每个字段。参见structfun。

或者，为了处理结构中的数据，可以在处理之前将结构转换为另一种类型的容器。

: *c =* **struct2cell** *(s)*

从struct对象中存储的对象创建一个新的单元格数组。

如果f是结构中的字段数，则生成的单元格数组将具有对应于[f size(s)]的维度向量。例如:

s = struct ("name", {"Peter", "Hannah", "Robert"},

"age", {23, 16, 3});

c = struct2cell (s)

⇒ c = {2x1x3 Cell Array}

c(1,1,:)(:)

⇒

{

[1,1] = Peter

[2,1] = Hannah

[3,1] = Robert

}

c(2,1,:)(:)

⇒

{

[1,1] = 23

[2,1] = 16

[3,1] = 3

}

**参见:**cell2struct, namedargs2cell, fieldnames。

: *c =* **namedargs2cell** *(s)*

从标量结构创建字段名称/值对的单元格数组。

例子:

s.Name = "Peter";

s.Height = 185;

s.Age = 42;

c = namedargs2cell (s)

⇒ { "Name", "Peter", "Height", 185, "Age", 42 }

**参见:**struct2cell。

**6.2 containers.Map**

: *m =* **containers.Map** *()*

: *m =* **containers.Map** *(keys, vals)*

: *m =* **containers.Map** *(keys, vals, "UniformValues", is\_uniform)*

: *m =* **containers.Map** *("KeyType", kt, "ValueType", vt)*

创建容器对象。存储键/值对列表的映射类。

Keys是映射的唯一键数组。键可以是数字标量或字符串。数字键的类型可以是double、single、int32、uint32、int64或uint64。其他数字或逻辑键将被转换为“double”。可以按原样输入单个字符串键。多个字符串键作为字符串的单元格数组输入。

Vals是映射的值数组，元素数量与键数相同。

在没有输入参数的情况下调用时，将创建一个默认映射，以字符串作为键类型，并以“any”作为值类型。

UniformValues选项指定映射的值是否必须严格为相同类型。如果is\_uniform为true，则首先验证要添加到映射中的任何值，以确保它们具有正确的类型。

当使用“KeyType”和“ValueType”参数调用时，使用指定的类型创建一个空映射。输入kt和vt分别是映射的键和值的类型。kt取值为char, double, single, int32, uint32, int64, uint64。vt取值为any, char, double, single, int32, uint32, int64, uint64, logical。

返回值m是容器的对象。映射类。

**参见:**struct。

**6.3单元格阵列**

在一个变量中存储不同大小或类型的多个变量既必要又方便。单元格数组是一个容器类，能够做到这一点。一般来说，单元格数组的工作方式与n维数组类似，只是使用“{”和“}”作为分配和索引操作符。

**6.3.1单元格阵列的基本用法**

作为示例，下面的代码创建了一个包含字符串和2 × 2随机矩阵的单元格数组

c = {"a string", rand(2, 2)};

要访问单元格数组的元素，可以使用{和}操作符对其进行索引。因此，在前面的例子中创建的变量可以像这样索引:

c{1}

⇒ ans = a string

与数值数组一样，单元格数组的几个元素可以通过索引向量的索引来提取

c{1:2}

⇒ ans = a string

⇒ ans =

0.593993 0.627732

0.377037 0.033643

索引操作符还可用于插入或覆盖单元格数组的元素。下面的代码将标量3插入到前面创建的单元格数组的第三位

c{3} = 3

⇒ c =

{

[1,1] = a string

[1,2] =

0.593993 0.627732

0.377037 0.033643

[1,3] = 3

}

有关索引单元格数组的详细信息，请参见索引单元格数组。

一般情况下，嵌套单元格数组按照前面的示例分层显示。在某些情况下，通过索引引用它们是有意义的，这可以通过celldisp函数来执行。

: **celldisp** *(c)*

: **celldisp** *(c, name)*

递归地显示单元格数组的内容。

默认情况下，这些值显示为变量c的名称。但是，该名称可以替换为变量名称。例如:

c = {1, 2, {31, 32}};

celldisp (c, "b")

⇒

b{1} =

1

b{2} =

2

b{3}{1} =

31

b{3}{2} =

32

**参见:**disp。

要测试一个对象是否是一个单元数组，请使用iscell函数。例如:

iscell (c)

⇒ ans = 1

iscell (3)

⇒ ans = 0

: *tf =* **iscell** *(x)*

如果x是一个单元格数组对象，则返回true。

**参见:**ismatrix, isstruct, iscellstr, isa。

**6.3.2创建单元格阵列**

介绍性示例(参见单元格数组的基本用法)展示了如何创建包含当前可用变量的单元格数组。但是，在许多情况下，创建单元格数组然后用数据填充它是有用的。

cell函数返回一个给定大小的单元格数组，其中包含空矩阵。此函数类似于用于创建新数值数组的zero函数。下面的示例创建一个包含空矩阵的2 × 2单元格数组

c = cell (2,2)

⇒ c =

{

[1,1] = [](0x0)

[2,1] = [](0x0)

[1,2] = [](0x0)

[2,2] = [](0x0)

}

就像数值数组一样，单元格数组也可以是多维的。cell函数接受任意数量的正整数来描述返回的cell数组的大小。也可以通过一个正整数向量来设置单元数组的大小。在下面的示例中，创建两个大小相等的单元格数组，并显示第一个单元格数组的大小

c1 = cell (3, 4, 5);

c2 = cell ( [3, 4, 5] );

size (c1)

⇒ ans =

3 4 5

可以看到，size函数也适用于单元格数组。其他描述对象大小的函数(如长度、数字、行和列)也是如此。

: *C =* **cell** *(n)*

: *C =* **cell** *(m, n)*

: *C =* **cell** *(m, n, k, …)*

: *C =* **cell** *([m n …])*

创建一个新的单元格数组对象。

如果使用单个标量整数参数调用，则返回方形NxN单元数组。如果用两个或多个标量整型参数或整型值的向量调用，则返回具有给定维度的数组。

**参见:**cellstr, mat2cell, num2cell, struct2cell。

作为创建空单元格数组然后填充它们的替代方法，可以使用num2cell、mat2cell和cellslices函数将数值数组转换为单元格数组。

: *C =* **num2cell** *(A)*

: *C =* **num2cell** *(A, dim)*

将数字矩阵A转换为单元格数组。

当不指定dim时，A的每个元素都成为输出C中的1x1元素。

如果定义了dim，则C的各个元素包含A在指定维度上的所有元素。Dim也可以是应用了相同规则的维度向量。

例如:

x = [1,2;3,4]

⇒

1 2

3 4

## each element of A becomes a 1x1 element of C

num2cell (x)

⇒

{

[1,1] = 1

[2,1] = 3

[1,2] = 2

[2,2] = 4

}

## all rows (dim 1) of A appear in each element of C

num2cell (x, 1)

⇒

{

[1,1] =

1

3

[1,2] =

2

4

}

## all columns (dim 2) of A appear in each element of C

num2cell (x, 2)

⇒

{

[1,1] =

1 2

[2,1] =

3 4

}

## all rows and cols appear in each element of C

## (hence, only 1 output)

num2cell (x, [1, 2])

⇒

{

[1,1] =

1 2

3 4

}

**参见:**mat2cell。

: *C =* **mat2cell** *(A, dim1, dim2, …, dimi, …, dimn)*

: *C =* **mat2cell** *(A, rowdim)*

将矩阵A转换为单元数组C。

每个维度参数(dim1, dim2等)是一个整数向量，它指定如何在输出C中的新元素中划分该维度的元素。第i个维度中的元素数量是size (a, i)。因为a中的所有元素都必须分区，所以要求sum (dimi) == size (a, i)。输出单元C的大小是nummel (dim1) x nummel (dim2) x…x nummel (dimn)。

给定一个单维参数rowdim，输出将按照指定划分为行。所有其他维度都没有被划分，因此所有列(dim 2)、页面(dim 3)等都出现在每个输出元素中。

例子

x = reshape (1:12, [3, 4])'

⇒

1 2 3

4 5 6

7 8 9

10 11 12

## The 4 rows (dim1) are divided in to two cell elements

## with 2 rows each.

## The 3 cols (dim2) are divided in to three cell elements

## with 1 col each.

mat2cell (x, [2,2], [1,1,1])

⇒

{

[1,1] =

1

4

[2,1] =

7

10

[1,2] =

2

5

[2,2] =

8

11

[1,3] =

3

6

[2,3] =

9

12

}

## The 4 rows (dim1) are divided in to two cell elements

## with a 3/1 split.

## All columns appear in each output element.

mat2cell (x, [3,1])

⇒

{

[1,1] =

1 2 3

4 5 6

7 8 9

[2,1] =

10 11 12

}

**参见:**num2cell, cell2mat。

: *sl =* **cellslices** *(x, lb, ub, dim)*

给定一个数组x，该函数从由下界和上界的索引向量lb、ub确定的数组中生成一个切片的单元格数组。

换句话说，它相当于下面的代码:

n = length (lb);

sl = cell (1, n);

for i = 1:length (lb)

sl{i} = x(:,…,lb(i):ub(i),…,:);

endfor

索引的位置由dim决定。如果未指定，则沿着第一个非单元素维度进行切片。

**参见:**cell2mat, cellindexmat, cellfun。

**6.3.3索引单元格数组**

如单元格数组的基本用法所示，可以使用“{”和“}”操作符从单元格数组中提取元素。如果您想提取或访问仍然是单元格数组的子数组，则需要使用'('和')'操作符。下面的例子说明了两者的区别:

c = {"1", "2", "3"; "x", "y", "z"; "4", "5", "6"};

c{2,3}

⇒ ans = z

c(2,3)

⇒ ans =

{

[1,1] = z

}

因此，使用'{}'可以访问单元格数组的元素，而使用'()'可以访问单元格数组的子数组。

使用'('和')'操作符，索引适用于单元格数组，就像多维数组一样。例如，可以将单元格数组的第一列和第三列的所有行设置为0，命令如下:

c(:, [1, 3]) = {0}

⇒ =

{

[1,1] = 0

[2,1] = 0

[3,1] = 0

[1,2] = 2

[2,2] = y

[3,2] = 5

[1,3] = 0

[2,3] = 0

[3,3] = 0

}

注意，以上也可以这样实现:

c(:, [1, 3]) = 0;

在这里，标量' 0 '被自动提升为单元格数组'{0}'，然后分配给c的子数组。

再举一个用'()'索引单元格数组的例子，你可以用下面的命令交换单元格数组的第一行和第二行:

c = {1, 2, 3; 4, 5, 6};

c([1, 2], :) = c([2, 1], :)

⇒ =

{

[1,1] = 4

[2,1] = 1

[1,2] = 5

[2,2] = 2

[1,3] = 6

[2,3] = 3

}

使用'{'和'}'操作符访问单元格数组的多个元素将产生一个包含所有请求元素的逗号分隔列表(参见逗号分隔列表)。使用'{'和'}'操作符，上面例子中的前两行可以像这样交换回来:

[c{[1,2], :}] = deal (c{[2, 1], :})

⇒ =

{

[1,1] = 1

[2,1] = 4

[1,2] = 2

[2,2] = 5

[1,3] = 3

[2,3] = 6

}

对于结构数组和数值数组，空矩阵'[]'可用于从单元格数组中删除元素:

x = {"1", "2"; "3", "4"};

x(1, :) = []

⇒ x =

{

[1,1] = 3

[1,2] = 4

}

下面的例子展示了如何删除单元格数组元素的内容，但不删除它们的空格:

x = {"1", "2"; "3", "4"};

x(1, :) = {[]}

⇒ x =

{

[1,1] = [](0x0)

[2,1] = 3

[1,2] = [](0x0)

[2,2] = 4

}

索引操作对单元格数组进行操作，而不是对单元格数组中的对象进行操作。相反，cellindexmat对每个单元格数组条目中的对象应用矩阵索引，并返回所请求的值。

: *y =* **cellindexmat** *(x, varargin)*

对单元格数组中的矩阵执行索引。

给定矩阵x的单元格数组，这个函数计算

Y = cell (size (X));

for i = 1:numel (X)

Y{i} = X{i}(varargin{1}, varargin{2}, …, varargin{N});

endfor

索引参数可以是标量(2)、数组([1,3])、范围(1:3)或冒号操作符(":")。但是，索引关键字端不可用。

**参见:**cellslices, cellfun。

**6.3.4字符串的单元格数组**

单元格数组的一个常见用途是在同一个变量中存储多个字符串。通过让每一行都是字符串，也可以在字符矩阵中存储多个字符串。然而，这引入了一个问题，即所有字符串的长度必须相等。因此，建议使用单元格数组来存储多个字符串。对于需要字符矩阵表示进行操作的情况，有几个函数可以将字符串的单元格数组转换为字符数组并再转换回来。char和strvcat将单元格数组转换为字符数组(参见串联字符串)，而cellstr函数将字符数组转换为字符串的单元格数组:

a = ["hello"; "world"];

c = cellstr (a)

⇒ c =

{

[1,1] = hello

[2,1] = world

}

: *cstr =* **cellstr** *(strmat)*

从字符串数组strmat的元素创建一个新的单元格数组对象。

格式的每一行都成为cstr的一个元素。在转换之前，删除行中任何尾随空格。

要从单元格字符串转换回字符数组，请使用char。

**参见:**cell, char。

使用单元格数组存储多个字符串的另一个优点是，Octave中包含的大多数字符串操作函数都支持这种表示。例如，可以使用strcmp函数将一个字符串与许多其他字符串进行比较。如果此函数的参数之一是字符串，另一个是字符串的单元格数组，则单元格数组的每个元素将与字符串参数进行比较:

c = {"hello", "world"};

strcmp ("hello", c)

⇒ ans =

1 0

以下字符串函数支持字符串的单元格数组:char、strvcat、strcat(参见串联字符串)、strcmp、strncmp、strcmpi、strncmpi(参见在字符串中搜索)、str2double、deblank、strtrim、strtrunc、strfind、strmatch、regexp、regexpi(参见字符串操作)和str2double(参见转换字符串)。

函数iscellstr可用于测试对象是否是字符串的单元格数组。

: *tf =* **iscellstr** *(cell)*

如果单元格数组的每个元素都是字符串，则返回true。

**参见:**ischar, isstring。

**6.3.5处理单元格数据**

存储在单元数组中的数据可以根据实际数据以几种方式进行处理。处理该数据的最简单方法是使用一个或多个for循环对其进行迭代。通过使用cellfun函数可以更容易地实现相同的思想，该函数对单元格数组的所有元素调用用户指定的函数。参见cellfun。

另一种方法是将数据转换为不同的容器，例如矩阵或数据结构。根据数据的不同，可以使用cell2mat和cell2struct函数。

: *m =* **cell2mat** *(c)*

通过将单元格数组c的所有元素连接成一个超矩形，将单元格数组c转换为一个矩阵。

c的元素必须是数字、逻辑或字符矩阵;或者单元阵列;或结构;猫必须能把它们连在一起。

参见:mat2cell, num2cell。

: *S =* **cell2struct** *(cell, fields)*

: *S =* **cell2struct** *(cell, fields, dim)*

将单元格转换为结构。

字段中的字段数必须与单元格中沿维度dim的元素数匹配，即numel (fields) == size (cell, dim)。如果省略dim，则假定值为1。

S = cell2struct ({"Peter", "Hannah", "Robert";

185, 170, 168},

{"Name","Height"}, 1);

S(1)

⇒

{

Name = Peter

Height = 185

}

**参见:**struct2cell, cell2mat, struct。

**6.4逗号分隔列表**

逗号分隔的列表2是所有Octave函数的基本参数类型——包括输入参数和返回参数。在这个例子中

max (a, b)

' a, b '是一个逗号分隔的列表。逗号分隔的列表可以同时出现在赋值的右侧和左侧。例如

x = [1 0 1 0 0 1 1; 0 0 0 0 0 0 7];

[i, j] = find (x, 2, "last");

这里，' x, 2， ' last ' '是一个逗号分隔的列表，它构成find的输入参数。Find返回以逗号分隔的输出参数列表，该列表逐个元素分配给以逗号分隔的列表' i, j '。

另一个使用逗号分隔列表的例子是用[]创建一个新数组(参见矩阵)或用{}创建一个单元格数组(参见单元格数组的基本用法)。在表达式中

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

c = {4, 5, 6, 7};

' 1,2,3,4 '和' 4,5,6,7 '都是逗号分隔的列表。

逗号分隔的列表不能由用户直接操作。但是，结构数组和单元格数组都可以转换为逗号分隔的列表，从而用于代替显式编写的逗号分隔列表。这个特性在很多方面都很有用，下面的小节将介绍。

**6.4.1单元格数组生成的逗号分隔列表**

如上所述(参见索引单元格数组)，可以使用{和}操作符将单元格数组的元素提取到逗号分隔的列表中。通过用[和]包围这个列表，它可以连接成一个数组。例如:

a = {1, [2, 3], 4, 5, 6};

b = [a{1:4}]

⇒ b =

1 2 3 4 5

类似地，可以创建一个新的单元格数组，其中包含用{}选择的单元格元素。通过用'{'和'}'包围列表，将创建一个新的单元格数组，如下面的示例所示:

a = {1, rand(2, 2), "three"};

b = { a{ [1, 3] } }

⇒ b =

{

[1,1] = 1

[1,2] = three

}

此外，单元格元素(通过{}访问)可以直接传递给函数。单元格数组中的元素列表将作为参数列表传递给给定函数，就像将元素作为单独的参数调用一样。下面的例子中对printf的两个调用是相同的，但后者更简单，可以处理任意大小的单元数组:

c = {"GNU", "Octave", "is", "Free", "Software"};

printf ("%s ", c{1}, c{2}, c{3}, c{4}, c{5});

-| GNU Octave is Free Software

printf ("%s ", c{:});

-| GNU Octave is Free Software

如果在赋值的左侧使用，则可以将由{}生成的逗号分隔列表赋值给。一个例子是

in{1} = [10, 20, 30];

in{2} = inf;

in{3} = "last";

in{4} = "first";

out = cell (4, 1);

[out{1:3}] = in{1 : 3};

[out{4:6}] = in{[1, 2, 4]})

⇒ out =

{

[1,1] =

10 20 30

[2,1] = Inf

[3,1] = last

[4,1] =

10 20 30

[5,1] = Inf

[6,1] = first

}

**6.4.2从结构数组生成逗号分隔列表**

结构数组同样可以用于创建以逗号分隔的列表。这是通过寻址结构数组的一个字段来实现的。例如:

x = ceil (randn (10, 1));

in = struct ("call1", {x, 3, "last"},

"call2", {x, inf, "first"});

out = struct ("call1", cell (2, 1), "call2", cell (2, 1));

[out.call1] = find (in.call1);

[out.call2] = find (in.call2);

**7 变量**

变量允许您为值命名并在以后引用它们。您已经在许多示例中看到了变量。变量名必须由字母、数字和下划线组成，但不能以数字开头。Octave对变量名的长度没有强制限制，但是变量名长度超过30个字符很少有用。以下都是有效的变量名

x

x15

\_\_foo\_bar\_baz\_\_

fucnrdthsucngtagdjb

然而，像\_\_foo\_bar\_baz\_\_这样以两个下划线开头和结尾的名字被理解为保留给Octave内部使用。你不应该在你写的代码中使用它们，除了访问Octave文档中的内部变量和内置符号常量。

大小写在变量名中很重要。符号a和a是不同的变量。

变量名本身就是一个有效的表达式。它表示变量的当前值。通过赋值操作符和自增操作符为变量赋新值。参见赋值表达式。

有一个自动创建的具有特殊含义的变量。ans变量总是包含最后一次计算的结果，其中的输出没有分配给任何变量。代码a = cos (pi)会将值-1赋给变量a，但不会改变ans的值。然而，代码cos (pi)会将ans的值设置为-1。

Octave中的变量没有固定类型，因此可以先将数值存储在变量中，然后在同一程序中使用相同的名称来保存字符串值。变量在被赋值之前不能使用。这样做会导致错误。

自动变量:ans

没有显式赋值给变量的最近计算结果。

例如，在表达式之后

3^2 + 4^2

，则ans返回的值为25。

: *tf =* **isvarname** *(name)*

如果name是有效的变量名，则返回true。

有效的变量名由字母、数字和下划线(\_)组成，且第一个字符不能是数字。

**参见:**iskeyword, exist, who。

: *varname =* **matlab.lang.makeValidName** *(str)*

: *varname =* **matlab.lang.makeValidName** *(…, "ReplacementStyle", rs)*

: *varname =* **matlab.lang.makeValidName** *(…, "Prefix", pfx)*

: *[varname, ismodified] =* **matlab.lang.makeValidName** *(…)*

从str创建有效的变量名varname。

输入str必须是字符串或字符串的单元格数组。输出的varname将是相同的类型。

有效的变量名是不以数字开头的字母、数字和下划线的序列。

“ReplacementStyle”选项指定如何处理无效字符。可接受的值是

"underscore" (默认)

用下划线(“\_”)替换所有无效字符。

"delete"

删除任何无效字符。

"hex"

用十六进制表示替换所有无效字符。

空白字符总是在应用“ReplacementStyle”之前被删除。空格后的小写字母将变为大写字母。

“Prefix”选项指定要添加的字符串pfx作为输入的前缀，如果它以数字开头。PFX本身必须是一个有效的变量名。默认前缀为“x”。

可选输出ismodified是一个逻辑数组，指示str中相应的元素是否为有效名称。

**参见:**iskeyword, isvarname, matlab.lang.makeUniqueStrings。

: *uniqstr =* **matlab.lang.makeUniqueStrings** *(str)*

: *uniqstr =* **matlab.lang.makeUniqueStrings** *(str, ex)*

: *uniqstr =* **matlab.lang.makeUniqueStrings** *(str, ex, maxlength)*

: *[uniqstr, ismodified] =* **matlab.lang.makeUniqueStrings** *(…)*

从字符串列表中构造一个唯一字符串列表。

输入str必须是字符串或字符串的单元格数组。输出的uniqstr将具有相同的类型。

该算法通过向第二个字符串添加下划线(“\_”)和数字计数使两个字符串唯一。

如果ex是字符串或字符串的单元格数组，uniqstr将包含它们之间以及相对于ex唯一的元素。

如果ex是str的索引数组或逻辑数组，那么它将选择唯一的str子集。未选中的元素不会被修改。

可选输入maxlength指定uniqstr中任意字符串的最大长度。如果输入字符串不能在不超过maxlength的情况下唯一，则会引发错误。

可选输出ismodified是一个逻辑数组，指示str中的每个元素是否被修改以使其唯一。

**参见:**unique, matlab.lang.makeValidName。

: *n =* **namelengthmax** *()*

返回MATLAB兼容的最大变量名长度。

Octave能够存储长度为2^{31}- 1的字符串。但是，为了与MATLAB兼容，所有变量、函数和结构字段的名称都应短于namelengthmax返回的长度。特别是，存储在MATLAB文件格式(\*.mat)中的变量的名称将被截断为这个长度。

**7.1全局变量**

关键词:global

全局变量是可以在Octave中的任何地方访问的变量。这与局部变量相反，局部变量只有在显式传递时才能在当前上下文之外访问，例如在调用函数(fcn (local\_var1, local\_var2))时将其作为参数包含。

使用全局声明语句将变量声明为全局。下面的语句都是全局声明。

global a

global a b

global c = 2

global d = 3 e f = 5

请注意，全局限定符仅扩展到下一个语句结束指示符，该指示符可以是逗号('，')、分号(';')或换行符(" \n ")。例如，下面的代码声明了一个全局变量a和一个局部变量b，赋值为1。

global a, b = 1

全局变量只能在全局语句中初始化一次。例如，在执行以下代码之后

global gvar = 1

global gvar = 2

全局变量gvar的值是1，而不是2。发出' clear gvar '命令不会改变上述行为，但' clear all '会。

为了访问一个通用变量，有必要在函数体内声明一个变量为全局变量。例如,

global x

function f ()

x = 1;

endfunction

f ()

不将全局变量x的值设置为1。相反，将创建一个名为x的局部变量，并将其赋值为1。为了更改全局变量x的值，还必须在函数体内将其声明为全局变量，如下所示

function f ()

global x;

x = 1;

endfunction

在函数参数列表中传递全局变量将生成一个局部副本，而不会修改全局值。例如，给定一个函数

function f (x)

x = 0

endfunction

把x定义为顶层的全局变量，

global x = 13

表达

f (x)

将函数内部的x值显示为0，但顶层的x值保持不变，因为函数使用其参数的副本。

编程注意:虽然全局变量有时是解决编码问题的正确方法，但现代最佳实践不鼓励使用它们。依赖于全局变量的代码在不同用户之间的行为可能不可预测，并且难以调试。这是因为全局变量可能会带来系统性的变化，因此很难将bug定位到特定函数或函数中的特定循环中。

: *tf =* **isglobal** *(name)*

如果name是全局可见的变量，则返回true。

例如:

global x

isglobal ("x")

⇒ 1

**参见:**isvarname, exist。

**7.2持久变量**

关键词: persistent

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。

下面的示例使用持久变量创建一个函数，该函数打印调用该函数的次数。

function count\_calls ()

persistent calls = 0;

printf ("'count\_calls' has been called %d times\n",

++calls);

endfunction

for i = 1:3

count\_calls ();

endfor

-| 'count\_calls' has been called 1 times

-| 'count\_calls' has been called 2 times

-| 'count\_calls' has been called 3 times

如示例所示，可以使用持久化声明语句将变量声明为持久化。下面的语句都是持久化声明。

persistent a

persistent a b

persistent c = 2

persistent d = 3 e f = 5

持久变量的行为等同于C语言中静态变量的行为。

持久变量的一个限制是，函数的输入参数和输出参数都不能是持久的:

function y = foo ()

persistent y = 0; # Not allowed!

endfunction

foo ()

-| error: can't make function parameter y persistent

与全局变量一样，持久变量只能初始化一次。例如，在执行以下代码之后

persistent pvar = 1

persistent pvar = 2

持久性变量pvar的值是1，而不是2。

如果声明了持久变量，但没有初始化为特定的值，则它将包含一个空矩阵。因此，也可以通过检查持久变量是否为空来初始化它，如下面的示例所示。

function count\_calls ()

persistent calls;

if (isempty (calls))

calls = 0;

endif

printf ("'count\_calls' has been called %d times\n",

++calls);

endfunction

这个实现的行为与count\_calls的前一个实现完全相同。

持久变量的值一直保存在内存中，直到显式清除为止。假设count\_calls的实现保存在磁盘上，我们会得到以下行为。

for i = 1:2

count\_calls ();

endfor

-| 'count\_calls' has been called 1 times

-| 'count\_calls' has been called 2 times

clear

for i = 1:2

count\_calls ();

endfor

-| 'count\_calls' has been called 3 times

-| 'count\_calls' has been called 4 times

clear all

for i = 1:2

count\_calls ();

endfor

-| 'count\_calls' has been called 1 times

-| 'count\_calls' has been called 2 times

clear count\_calls

for i = 1:2

count\_calls ();

endfor

-| 'count\_calls' has been called 1 times

-| 'count\_calls' has been called 2 times

也就是说，持久变量只有在包含该变量的函数被删除时才会从内存中删除。注意，如果函数定义直接输入到Octave提示符中，那么持久性变量将被一个简单的clear命令清除，因为整个函数定义将从内存中删除。如果您不希望即使函数被清除也从内存中删除持久变量，则应该使用mlock函数(参见函数锁定)。

**7.3变量状态**

在创建简单的一次性程序时，在提示符处查看哪些变量可用是非常方便的。函数who及其兄弟函数whos和whos\_line\_format将显示关于内存中的内容的不同信息，如下所示。

str = "A random string";

who

-| Variables in the current scope:

-|

-| ans str

: **who**

: **who** *pattern …*

: **who** *option pattern …*

: *C =* **who** *(…)*

列出当前定义的与给定模式匹配的变量。

有效的模式语法与clear命令所描述的相同。如果没有提供模式，则列出所有变量。

默认情况下，只显示局部范围内可见的变量。

以下是有效的选项，但不能组合使用。

global

列出全局作用域中的变量，而不是当前作用域中的变量。

-regexp

在匹配要显示的变量时，模式被认为是正则表达式。使用regexp函数接受的相同模式语法。

-file

下一个参数被视为文件名。列出在指定文件中找到的所有变量。从文件中读取变量时不接受任何模式。

如果作为函数调用，则返回与给定模式匹配的已定义变量名的单元格数组。

**参见:**whos, isglobal, isvarname, exist, regexp。

: **whos**

: **whos** *pattern …*

: **whos** *option pattern …*

: *S =* **whos** *("pattern", …)*

提供与给定模式匹配的当前定义变量的详细信息。

选项和模式语法与who命令相同。

关于每个变量的扩展信息汇总在一个表中，其中包含以下默认项。

Attr

所列变量的属性。可能的属性有:

blank

局部作用域内的变量

c

复杂类型变量。

f

形式形参(函数参数)。

g

变量具有全局作用域。

p

持续的变量。

Name

变量的名称。

Size

变量的逻辑大小。标量是1x1，向量是1xN或Nx1，二维矩阵是MxN。

Bytes

当前用于存储变量的内存量。

Class

变量的类。例子包括double, single, char, uint16, cell和struct。

可以通过whos\_line\_format函数对表进行定制，以显示更多或更少的信息。

如果whos作为函数调用，则返回与给定模式匹配的已定义变量名的结构数组。结构中描述每个变量的字段是:名称、大小、字节、类、全局、稀疏、复杂、嵌套、持久。

参见:who, whos\_line\_format。

: *val =* **whos\_line\_format** *()*

: *old\_val =* **whos\_line\_format** *(new\_val)*

: *old\_val =* **whos\_line\_format** *(new\_val, "local")*

查询或设置whos命令使用的格式字符串。

完整的格式字符串是:

%[modifier]<command>[:width[:left-min[:balance]]];

可用的命令序列如下:

%a

打印变量的属性(c=complex, s=sparse, f=formal parameter, g=global, p=persistent)。

%b

打印变量占用的字节数。

%c

打印变量的类名。

%e

打印变量保存的元素。

%n

打印变量名。

%s

打印变量的尺寸。

%t

打印变量的类型名称。

每个命令也可以有对齐修饰符:

l

左对齐

r

右对齐(默认)。

c

列对齐(仅适用于命令%s)。

width参数是一个正整数，指定用于打印的最小列数。不需要最大值，因为该字段将根据需要自动扩展。

参数left-min和balance仅在与命令“%s”一起使用列对齐修饰符时可用。Balance指定字段宽度内的列号，它将在条目之间对齐。编号从0开始，表示最左边的一列。left-min指定指定的余额列左边的最小字段宽度。

默认格式为:

" %la:5; %ln:6; %cs:16:6:1; %rb:12; %lc:-1;\n"

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:**whos。

可以确定给定变量是否可用，而不是显示内存中有哪些变量。这样就可以根据变量的存在来改变程序的行为。下面的示例说明了这一点。

if (! exist ("meaning", "var"))

disp ("The program has no 'meaning'");

endif

: *c =* **exist** *(name)*

: *c =* **exist** *(name, type)*

检查name是否作为变量、函数、文件、目录或类存在。

返回码c是其中之一

1

Name是一个变量。

2

name是一个绝对文件名，Octave路径中的普通文件，或者(在附加' .m '之后)Octave路径中的函数文件。

3

name是’.oct’ or ‘.mex’在Octave的path中。

5

Name是一个内置函数。

7

Name为目录。

8

Name是一个classdef类。

103

Name是一个与文件无关的函数(在命令行中输入)。

0

名称不存在。

如果提供了可选参数类型，则只检查指定类型的符号。有效的类型有

"var"

只检查变量。

"builtin"

只检查内置函数。

"dir"

只检查目录。

"file"

只检查文件和目录。

"class"

只检查classdef类。

如果没有给出类型，并且name有多个可能的匹配项，exist将根据以下优先级列表返回一个代码:变量、内置函数、oct-file、目录、文件、类。

如果在Octave的搜索路径中存在一个名为name的常规文件，exist返回2。对于不在搜索路径上的其他类型文件的信息，请使用函数file\_in\_path和stat的某种组合。

编程注意:If name是由一个有bug的。oct/实现的。调用exist可能会导致Octave崩溃。为了保持高性能，Octave信任.oct/。Mex文件，而不是对它们进行沙箱处理。

**参见:**file\_in\_loadpath, file\_in\_path, dir\_in\_loadpath, stat。

通常Octave会管理内存，但有时手动从内存中删除变量也是可行的。当处理占用大量内存的大变量时，通常需要这样做。在使用IEEE浮点格式的计算机上，下面的程序分配一个需要大约128 MB内存的矩阵。

large\_matrix = zeros (4000, 4000);

由于将这个变量放在内存中可能会降低其他计算的速度，因此有必要手动从内存中删除它。clear或clearvars函数可以做到这一点。

: **clear**

: **clear** *pattern …*

:**clear** *options pattern …*

删除与给定模式匹配的名称，从而释放内存。

模式可以包含以下特殊字符:

?

匹配任何单个字符

\*

匹配零个或多个字符。

[ list ]

匹配 list 指定的字符列表。如果第一个字符是 ！或 ^，匹配除 list 指定的字符之外的所有字符。例如，模式 [a-zA-Z] 将匹配所有小写和大写字母字符。在 Windows 上，方括号是字面匹配的，不用于对字符进行分组。

例如:命令

clear foo b\*r

清除名称foo和所有以字母“b”开头、以字母“r”结尾的名称。

如果clear在没有任何参数的情况下被调用，所有用户定义的变量都会从当前工作区中被清除(例如，局部变量)。出现的任何全局变量在当前工作空间中将不再可见，但它们将继续存在于全局工作空间中。函数不受这种形式的clear的影响。

以下选项有长格式和短格式

all, -all, -a

清除符号表中的所有本地和全局用户定义变量以及所有函数。

-exclusive, -x

清除与以下模式不匹配的变量。

functions, -functions, -f

从函数符号表中清除函数名。持久变量将被重新初始化为其默认值，除非该函数已被mlock锁定在内存中。

global, -global, -g

清除全局变量名。

variables, -variables, -v

清除局部变量名。

classes, -classes, -c

清除类结构表和所有对象。

-regexp, -r

模式参数被视为正则表达式，所有匹配项都将被清除。

除了-exclusive和-regexp，所有长选项都可以不使用破折号。注意，除了-exclusive之外，只能出现一个其他选项。所有选项必须出现在任何模式之前。

编程注意事项:命令clear name仅在当前定义了变量和名为name的(阴影)函数时清除变量名。例如，假设您定义了一个函数foo，然后通过执行赋值foo = 2来隐藏它。执行一次clear foo命令将清除变量定义并恢复foo作为函数的定义。第二次执行clear foo将清空函数定义。

清除链接到全局变量的局部变量名时，仅删除该变量的局部副本。全局副本不受影响，可以使用全局global\_varname进行恢复。相反，clear -g global\_varname将同时删除本地和全局变量。

**参见:**clearvars, who, whos, exist, mlock。

: **clearvars**

: **clearvars** *pattern …*

: **clearvars** *-regexp pattern …*

: **clearvars** *… -except pattern …*

: **clearvars** *… -except -regexp pattern …*

: **clearvars** *-global …*

从内存中删除与给定模式匹配的变量。

模式可以包含以下特殊字符:

?

匹配任何单个字符

\*

匹配零个或多个字符。

[ list ]

匹配list指定的字符列表。如果第一个字符是!或^，匹配除list指定字符外的所有字符。例如，模式[a-zA-Z]将匹配所有小写和大写字母字符。

如果给出-regexp选项，则后续模式将被视为正则表达式，并且将清除任何匹配项。

如果给出了-except选项，则后续模式将选择不被清除的变量。

如果给出-global选项，则所有模式将应用于全局变量而不是局部变量。

当不带参数调用clearvars时，它会删除所有的局部变量。

示例代码:

清除所有以“x”开头的变量和特定的变量“foobar”

clearvars x\* foobar

清除特定变量“foobar”，并使用正则表达式清除所有以“x”或“y”开头的变量。

clearvars foobar -regexp ^x ^y

清除除“foobar”以外的所有变量

clearvars -except foobar

清除所有以"foo"开头的变量，以"bar"结尾的除外

clearvars foo\* -except -regexp bar$

**参见:**clear, who, whos, exist。

: **pack** *()*

在MATLAB中巩固工作空间内存。

这个函数是为了兼容性而提供的，但在Octave中不做任何事情。

**参见:**clear。

关于函数或变量的信息，比如它在文件系统中的位置，也可以从Octave中获得。这通常只在程序开发期间有用，而不是在程序中有用。

: **type** *name …*

: **type** *-q name …*

: *text =* **type** *("name", …)*

显示name的内容，可以是文件、函数(m-file)、变量、操作符或关键字。

Type通常在标题行前加上描述名称类别的标题行，如函数或变量;-q选项抑制这种行为。

如果不使用输出变量，内容将显示在屏幕上。否则，返回字符串的单元格数组，其中每个元素对应于每个请求函数的内容。

:**which** *name …*

*: [str, …] =* **which** *('name', …)*

显示每个名称的类型。

如果从函数文件中定义了name，则还会显示该文件的全名。

**参见:**帮助，寻找。

: **what**

:**what** *dir*

: *w =* **what** *(dir)*

列出目录dir中特定于Octave的文件。

如果未指定dir，则使用当前目录。

如果请求返回参数，找到的文件将在结构w中返回。该结构包含以下字段:

path

目录dir的全路径

m

m文件的单元格数组

mat

mat文件的单元数组

mex

mex文件的单元格数组

oct

oct文件的单元数组

mdl

mdl文件的单元数组

slx

slx文件的单元数组

p

p文件的单元数组

classes

类目录的单元数组(@classname/)

packages

包目录的单元数组(+pkgname/)

兼容性注意:Octave不支持mdl、slx和p文件。什么将总是返回这些类别的空列表。

**参见:**which, ls, exist。

**8表达式**

表达式是Octave中语句的基本构建块。表达式的计算结果是一个值，您可以打印、测试、存储在变量中、传递给函数，或者使用赋值操作符将新值赋给变量。

表达式本身可以作为语句使用。大多数其他类型的语句包含一个或多个表达式，这些表达式指定要操作的数据。与其他语言一样，Octave中的表达式包括变量、数组引用、常量和函数调用，以及它们与各种操作符的组合。

**8.1索引表达式**

索引表达式允许您引用或提取矢量、矩阵 （2-D） 或高维数组的选定元素。数组可以通过以下三种方式之一进行索引：组件索引、线性索引和逻辑索引。

**组件索引**

分量索引可以是标量、矢量、范围或特殊运算符“：”，用于选择整行、整列或更高维的切片。

组件索引表达式由一组括号组成，括起来用逗号分隔的 M 表达式。每个单独的索引值或组件都用于应用它的对象的相应维度。换句话说，第一个索引组件用于对象的第一个维度（行），第二个索引组件用于对象的第二个维度（列），依此类推。索引分量 M 的数量定义了索引表达式的维数。具有两个分量的索引将称为二维索引，因为它具有两个维度。

在最简单的情况下，1） 所有分量都是标量，并且 2） 索引表达式 M 的维数等于它所应用对象的维数。例如：

A = reshape (1:8, 2, 2, 2) # Create 3-D array

A =

ans(:,:,1) =

1 3

2 4

ans(:,:,2) =

5 7

6 8

A(2, 1, 2) # second row, first column of second slice

# in third dimension: ans = 6

返回对象在特定维度上的大小等于索引表达式对应组件中的元素数量。当所有组件都是标量时，结果是单个输出值。但是，如果任何组件是矢量或范围，则返回值是各自维度中索引的笛卡尔积。例如:

A([1, 2], 1, 2) ≡ [A(1,1,2); A(2,1,2)]

⇒

ans =

5

6

返回值的总数是为每个索引组件返回的元素数量的乘积。在上面的例子中，总数是2\*1\*1 = 2个元素。

注意，在给定维度中返回的对象的大小等于该维度的索引表达式中的元素数。在上面的代码中，第一个索引组件([1,2])被指定为行向量，但它的形状并不重要。重要的事实是，组件指定了两个值，因此结果在第一维中必须具有2的大小;因为第一维对应于行，所以整体结果是一个列向量。

A(1, [2, 1, 1], 1) # result is a row vector: ans = [3, 1, 1]

A(ones (2, 2), 1, 1) # result is a column vector: ans = [1; 1; 1; 1]

第一行再次演示了给定维度中的输出大小等于相应索引组件中的元素数量。在本例中，输出在第二维中有三个元素(对应于列)，因此结果是一个行向量。该示例还展示了如何使用索引表达式中的重复条目来复制输出中的元素。最后一个例子进一步证明了索引组件的形状无关紧要，重要的只是元素的数量(2x2 = 4)。

只要索引表达式的维数大于1 (M > 1)，上述规则就适用。但是，对于一维索引表达式，则适用特殊的规则，并且输出的形状由索引组件的形状决定。例如:

A([1, 2]) # result is a row vector: ans = [1, 2]

A([1; 2]) # result is a column vector: ans = [1; 2]

A(P)的形状规则为:

* 当A或P中至少有一个具有二维或二维以上维度时，则A(P)具有P的形状。当至少一个变量是二维矩阵或N-D数组时，就会发生这种情况。
* 当A和P都是一维向量时，那么A(P)就是A本身的形状。特别地，当A是一个行向量时，那么A(P)也是一个行向量，与P的形状无关。当A是列向量的情况是类似的。

冒号(':')可以用作索引组件，以选择指定维度中的所有元素。给定矩阵，

A = [1, 2; 3, 4]

下面所有的表达式都是等价的，选择矩阵的第一行。

A(1, [1, 2]) # row 1, columns 1 and 2

A(1, 1:2) # row 1, columns in range 1-2

A(1, :) # row 1, all columns

当在一维索引的特殊情况下使用冒号时，结果总是一个列向量。使用冒号索引创建列向量是一种非常常见的代码习惯用法，在这种情况下，它通常比调用重塑更快、更清晰。

A(:) # result is column vector: ans = [1; 2; 3; 4]

A(:)' # result is row vector: ans = [1, 2, 3, 4]

在索引表达式中，关键字end自动引用特定维度的最后一个条目。这个神奇的索引也可以用于范围，并且通常不需要在索引之前调用size或length来收集数组边界。例如:

A(1:end/2) # first half of A => [1, 2]

A(end + 1) = 5; # append element

A(end) = []; # delete element

A(1:2:end) # odd elements of A => [1, 3]

A(2:2:end) # even elements of A => [2, 4]

A(end:-1:1) # reversal of A => [4, 3, 2, 1]

有关更多信息,请参阅"end"关键字。

**线性索引**

允许将一维索引与多维对象一起使用。这也称为线性索引。在这种情况下，多维数组的元素按列优先顺序获取，就像在 Fortran 中一样。也就是说，将数组的列想象成彼此堆叠以形成一个列矢量，然后将单个线性索引应用于该矢量。

A = [1, 2, 3; 4, 5, 6; 7, 8, 9];

A(4) # linear index of 4th element in 2-D array: ans = 2

A(3:5) # result has shape of index component: ans = [7, 2, 5]

A([1, 2, 2, 1]) # result includes repeated elements: ans = [1, 4, 4, 1]

**逻辑索引**

逻辑值还可用于索引矩阵和元胞数组。使用逻辑数组进行索引时，结果将是一个矢量，其中包含与逻辑数组的真实部分相对应的值。以下示例对此进行了说明。

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

idx = [true, false; false true];

data(idx)

⇒ ans = [ 1; 4 ]

idx = (data <= 2);

data(idx)

⇒ ans = [ 1; 2 ]

在上面的代码中，可以将 data（idx） 替换为 data（ data <= 2 ），而不是创建 idx 数组。

虽然逻辑索引表达式的大小通常与要索引的数组的大小相同，但这不是必要条件。如果逻辑索引的大小与数组不同，则数组中的元素将根据其线性顺序与逻辑索引中的元素进行匹配，就像线性索引一样。

data = [ 1, 2, 3; 4, 5, 6 ];

idx = [ true, false, false, true ];

data(idx)

⇒ ans = [ 1 5 ] # idx selected the 1st and 4th position elements

如果逻辑索引大于 then 数组，则如果任何 true 值尝试选择大于数组中元素数的线性位置，则会发生越界错误。

idx = [ true, true, false; false, true, false; true; false; false ];

data(idx)

⇒ ans = [ 1; 2; 5; 3 ] # returns positions 1, 3, 4, 5 in a column

idx = [ true, true, false; false, true, false; true; false; true ];

data(idx)

⇒ error: a(9): out of bound 6 (dimensions are 2x3)

超出数组大小的逻辑索引的 false 元素将被忽略，但是当逻辑索引的第 9 个元素为 true 时，它会在尝试选择数组中不存在的第 9 个元素时触发错误。

**8.1.1高级索引**

**链式索引**

Octave 允许使用重复（链式）索引表达式在单个命令中提取数组的子集，而无需使用中间变量。这样可以更轻松地编写具有复杂索引操作或使用多种索引方法的代码。以下示例演示了两个等效的索引提取操作：

A = reshape (1:16, 4, 4);

B = A(2:4, 2:3);

C = B(3:5);

D = C( [ true, false, true ] )

⇒ D = [ 8, 11 ]

D = A(2:4, 2:3)(3:5)([ true, false, true ])

⇒ D = [ 8, 11 ]

链式索引必然比生成相同结果的单个索引表达式慢，但通常比使用中间变量赋值执行多个离散索引操作的计算效率更高。

请注意，链式索引仅与右侧表达式兼容，不能用于赋值操作的左侧。

**分量到线性指数的转换**

当需要从索引不能写为组件的笛卡尔乘积的数组中提取条目的子集时，可以使用线性索引和函数 sub2ind。例如：

A = reshape (1:8, 2, 2, 2) # Create 3-D array

A =

ans(:,:,1) =

1 3

2 4

ans(:,:,2) =

5 7

6 8

A(sub2ind (size (A), [1, 2, 1], [1, 1, 2], [1, 2, 1]))

⇒ ans = [A(1, 1, 1), A(2, 1, 2), A(1, 2, 1)]

: *ind =* **sub2ind** *(dims, i, j)*

: *ind =* **sub2ind** *(dims, s1, s2, …, sN)*

将下标转换为线性索引。

输入 dims 是一个维度矢量，其中每个元素都是相应维度中数组的大小（请参阅大小）。其余输入是要转换的下标的标量或矢量。

输出矢量 ind 包含转换后的线性指数。

背景：数组元素可以由线性索引指定，该索引从 1 开始并贯穿数组中的元素数，也可以使用行、列、页等的下标指定它们。函数 ind2sub 和 sub2ind 在两种形式之间相互转换。

线性索引遍历维度 1（行），然后是维度 2（列），然后是维度 3（页面），依此类推，直到它对所有元素进行编号。请考虑以下 3×3 矩阵：

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

[(2,1), (2,2), (2,3)] ==> [2, 5, 8]

[(3,1), (3,2), (3,3)] [3, 6, 9]

左矩阵包含每个矩阵元素的下标元组。右矩阵显示同一矩阵的线性索引。

下面的示例演示如何通过对 sub2ind 的单个调用将 3×3 矩阵的二维索引 （2,1） 和 （2,3） 转换为线性索引。

s1 = [2, 2];

s2 = [1, 3];

ind = sub2ind ([3, 3], s1, s2)

⇒ ind = 2 8

**参见:** ind2sub, size.

: *[s1, s2, …, sN] =* **ind2sub** *(dims, ind)*

将线性索引转换为下标。

输入 dims 是一个维度矢量，其中每个元素都是相应维度中数组的大小（请参阅大小）。第二个输入 ind 包含要转换的线性索引。

输出 s1、...、sN 包含转换后的下标。

背景：数组元素可以由线性索引指定，该索引从 1 开始并贯穿数组中的元素数，也可以使用行、列、页等的下标指定它们。函数 ind2sub 和 sub2ind 在两种形式之间相互转换。

线性索引遍历维度 1（行），然后是维度 2（列），然后是维度 3（页面），依此类推，直到它对所有元素进行编号。请考虑以下 3×3 矩阵：

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

[2, 5, 8] ==> [(2,1), (2,2), (2,3)]

[3, 6, 9] [(3,1), (3,2), (3,3)]

左侧矩阵包含每个矩阵元素的线性索引。右矩阵显示同一矩阵的下标元组。

下面的示例演示如何将线性索引 2 和 8 转换为 3×3 矩阵的相应下标。

ind = [2, 8];

[r, c] = ind2sub ([3, 3], ind)

⇒ r = 2 2

⇒ c = 1 3

如果输出下标数超过维数，则超出的维数设置为1。另一方面，如果提供的下标少于维度，则超出的维度将合并到最终请求的维度中。为清楚起见，请考虑以下示例：

ind = [2, 8];

dims = [3, 3];

## same as dims = [3, 3, 1]

[r, c, s] = ind2sub (dims, ind)

⇒ r = 2 2

⇒ c = 1 3

⇒ s = 1 1

## same as dims = [9]

r = ind2sub (dims, ind)

⇒ r = 2 8

**参见:**sub2ind,size。

: *tf =* **isindex** *(ind)*

: *tf =* **isindex** *(ind, n)*

如果 ind 是有效的索引，则返回 true。

有效索引可以是正整数（尽管可能是实数据类型）或逻辑数组。

如果存在，则 n 指定要编制索引的维度的最大范围。如果可能，将缓存内部结果，以便后续使用 ind 的索引不会再次执行检查。

实现说明：在检查有效索引之前，首先将字符串转换为双精度值。除非字符串包含 NULL 字符“\0”，否则它将始终是有效的索引。

**组件数量不等于维度**

具有“nd”维度的数组可以由具有 1 到 'nd' 分量的索引表达式进行索引。对于普通和最常见的情况，组件数“M”与维度数“nd”匹配。在这种情况下，适用普通的索引规则，每个组件对应于数组的相应维度。

但是，如果索引分量的数量超过维度数 （M > nd），则多余的分量必须全部是单例 （1）。此外，如果 M < nd，则该行为等效于重塑输入对象，以便将尾随的 nd M 维度合并到最后一个索引维度 M 中。因此，结果将具有索引表达式的维数，而不是原始对象的维数。当索引的维数大于 1 （M > 1） 时，就会出现这种情况，因此不会应用线性索引的特殊规则。举个例子，这是最容易理解的：

A = reshape (1:8, 2, 2, 2) # Create 3-D array

A =

ans(:,:,1) =

1 3

2 4

ans(:,:,2) =

5 7

6 8

## 2-D indexing causes third dimension to be merged into second dimension.

## Equivalent array for indexing, Atmp, is now 2x4.

Atmp = reshape (A, 2, 4)

Atmp =

1 3 5 7

2 4 6 8

A(2,1) # Reshape to 2x4 matrix, second entry of first column: ans = 2

A(2,4) # Reshape to 2x4 matrix, second entry of fourth column: ans = 8

A(:,:) # Reshape to 2x4 matrix, select all rows & columns, ans = Atmp

请注意，这里优雅地使用双冒号来替换对 reshape 函数的调用。

**阵列复制**

线性索引的另一个高级用途是创建填充单个值的数组。这可以通过在标量值上使用 1 的索引来完成。结果是一个对象，其索引表达式的维度和每个元素都等于原始标量。例如，以下语句

a = 13;

a(ones (1, 4))

生成一个行矢量，其四个元素都等于 13。

同样，通过用两个 1 矢量对标量进行索引，可以创建一个矩阵。以下声明

a = 13;

a(ones (1, 2), ones (1, 3))

创建一个 2x3 矩阵，所有元素都等于 13。这也可以写成

13(ones (2, 3))

使用索引而不是代码构造标量 \* （M， N， ...） 更有效，因为它避免了不必要的乘法运算。此外，不能为要复制的对象定义乘法，而始终定义索引数组。以下代码演示如何从基本单元创建 2x3 元胞阵列，该单元本身不是标量。

{"Hello"}(ones (2, 3))

应该注意的是，1 （1， n）（1 的行矢量）会产生一个范围对象（增量为零）。范围在内部存储为起始值、增量、结束值和值总数;因此，当元素数量大于 4 时，它比矢量或矩阵的存储效率更高。特别是，当 'r' 是行矢量时，表达式

r(ones (1, n), :)

r(ones (n, 1), :)

将产生相同的结果，但第一个结果会快得多，至少对于“r”和“n”足够大。在第一种情况下，索引以压缩形式保存为一个范围，这允许 Octave 选择更有效的算法来处理表达式。

对于不熟悉这些技术的用户，一般建议使用函数 repmat 将较小的数组复制到较大的数组中，该函数使用此类技巧。

**用于增强性能的索引**

索引的第二个用途是加快代码速度。索引是一种快速操作，明智地使用它可以减少对单个数组元素进行循环的要求，这是一种缓慢的操作。

考虑以下示例，该示例创建一个包含值 a（i） = sqrt （i） 的 10 元素行矢量 a。

for i = 1:10

a(i) = sqrt (i);

endfor

使用这样的循环创建矢量是相当低效的。在这种情况下，使用表达式的效率会高得多

a = sqrt (1:10);

这完全避免了循环。

在无法避免循环的情况下，或者必须将多个值组合在一起以形成更大的矩阵，通常先设置矩阵的大小（预先分配存储），然后使用索引命令插入元素会更快。例如，给定一个矩阵 a，

[nr, nc] = size (a);

x = zeros (nr, n \* nc);

for i = 1:n

x(:,(i-1)\*nc+1:i\*nc) = a;

endfor

速度明显快于

x = a;

for i = 1:n-1

x = [x, a];

endfor

因为 Octave 不必反复调整中间结果的大小。

有关更多性能改进建议，请参阅矢量化和更快的代码执行。

**8.2调用功能**

函数是特定计算的名称。因为它有一个名字，你可以在程序的任何时候通过名字来请求它。例如，函数sqrt计算一个数的平方根。

一组固定的函数是内置的，这意味着它们可以在每个Octave程序中使用。根号函数就是其中之一。此外，您还可以定义自己的函数。有关如何做到这一点的信息，请参阅函数和脚本。

使用函数的方法是使用函数调用表达式，该表达式由函数名和括号内的参数列表组成。参数是表达式，它为函数将要进行的计算提供了原始材料。当有多个参数时，它们之间用逗号分隔。如果没有参数，可以省略圆括号，但包含圆括号是一个好主意，以便清楚地表明要进行函数调用。下面是一些例子:

sqrt (x^2 + y^2) # One argument

ones (n, m) # Two arguments

rand () # No arguments

每个函数都需要特定数量的参数。例如，调用sqrt函数时必须带一个参数，即要取平方根的数字:

sqrt (argument)

一些内置函数的参数数量是可变的，这取决于特定的用法，它们的行为取决于所提供的参数的数量。

与其他表达式一样，函数调用也有一个值，该值由函数根据您提供的参数计算得出。在这个例子中，sqrt (argument)的值是参数的平方根。函数也可能有副作用，例如为某些变量赋值或进行输入或输出操作。

与大多数语言不同，Octave中的函数可以返回多个值。例如，以下语句

[u, s, v] = svd (a)

计算矩阵a的奇异值分解，并将三个结果矩阵赋值给u、s和v。

多重赋值表达式的左侧本身是一个表达式列表，即可能由索引表达式限定的变量名列表。另请参见索引表达式和赋值表达式。

**8.2.1按值调用**

在Octave中，与Fortran不同，函数参数是按值传递的，这意味着函数调用中的每个参数在传递给函数之前都会被求值并分配给内存中的临时位置。目前还没有办法指定应该通过引用而不是通过值传递函数参数。这意味着不可能在调用函数中直接改变函数参数的值。它只能更改函数体中的本地副本。例如，函数

function f (x, n)

while (n-- > 0)

disp (x);

endwhile

endfunction

显示第一个参数的值n次。在这个函数中，变量n被用作临时变量，而不必担心它的值在调用函数中也会改变。按值调用也很有用，因为总是可以为任何函数参数传递常量，而不必首先确定函数不会尝试修改参数。

调用者可以使用变量作为参数的表达式，但被调用的函数并不知道这一点:它只知道参数的值。例如，给定一个名为as的函数

foo = "bar";

fcn (foo)

您不应该将参数视为“变量foo”。相反，可以将参数视为字符串值“bar”。

尽管Octave对函数参数使用按值传递语义，但值不会被不必要地复制。例如,

x = rand (1000);

f (x);

实际上并不强制存在两个1000 × 1000元素矩阵，除非函数f修改了其参数的值。然后Octave必须创建一个副本，以避免在函数f的作用域之外更改值，或者尝试(并且可能失败!)修改常量的值或临时结果的值。

**8.2.2递归**

在一些限制条件下，允许递归函数调用。递归函数是直接或间接调用自身的函数。例如，下面是一种计算给定整数阶乘的低效方法:

function retval = fact (n)

if (n > 0)

retval = n \* fact (n-1);

else

retval = 1;

endif

endfunction

这个函数是递归的，因为它直接调用自己。它最终会终止，因为每次调用它自己时，它使用的参数比上次调用时少1。一旦参数不再大于0，它就不再调用自己，递归结束。

函数max\_recursion\_depth可用于指定递归深度的限制，并防止Octave无限递归。类似地，函数max\_stack\_depth可用于指定函数调用的深度限制，无论是否递归。这些限制有助于防止Octave正在运行的计算机上的堆栈溢出，因此解释器将抛出错误并返回命令提示符，而不是使用信号退出。

: *val =* **max\_recursion\_depth** *()*

: *old\_val =* **max\_recursion\_depth** *(new\_val)*

: *old\_val =* **max\_recursion\_depth** *(new\_val, "local")*

查询或设置函数可递归调用的次数的内部限制。

如果超出限制，则打印一条错误消息，并将控制返回到顶层。

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:**max\_stack\_depth。

: *val =* **max\_stack\_depth** *()*

: *old\_val =* **max\_stack\_depth** *(new\_val)*

: *old\_val =* **max\_stack\_depth** *(new\_val, "local")*

查询或设置函数可递归调用的次数的内部限制。

如果超出限制，则打印一条错误消息，并将控制返回到顶层。

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:**max\_recursion\_depth。

**8.2.3通过句柄访问**

可以通过使用特殊操作符“@”获得的函数句柄对函数进行抽象和引用。例如,

f = @plus;

f (2, 2)

⇒ 4

等价于直接调用+ (2,2)除了一般编程的抽象之外，函数句柄还可以通过向属性添加侦听器或分配预先存在的操作，在图形和图形的回调方法中使用，例如以下示例:

function mydeletefcn (h, ~, msg)

printf (msg);

endfunction

sombrero;

set (gcf, "deletefcn", {@mydeletefcn, "Bye!\n"});

close;

以上将在图形关闭(删除)时向终端打印“Bye!”有许多图形属性动作可以分配回调函数，包括，buttondownfcn, windowscrollwheelfcn, createfcn, deletefcn, keypressfcn等。

请注意，' @ '字符在定义类函数(即方法)时也起作用，但不是作为语法元素。相反，它以一个目录名开头，其中包含共享目录名的类的方法，但没有“@”字符。参见面向对象编程。

**8.3算术运算符**

以下算术运算符可用，可用于标量和矩阵。逐个元素的操作符和函数广播(参见广播)。

x + y

加法(总是一个元素一个元素地工作)。如果两个操作数都是矩阵，则行数和列数必须一致，或者它们必须可以广播到相同的形状。

x - y

减法(总是一个元素一个元素地工作)。如果两个操作数都是矩阵，则它们的行数和列数必须一致，或者它们必须可以广播为相同的形状。

x \* y

矩阵乘法。x的列数必须与y的行数一致。

x .\* y

中的元素的乘法。如果两个操作数都是矩阵，则行数和列数必须一致，或者它们必须可以广播到相同的形状。

x / y

正确的部门。这在概念上等同于表达式

(inv (y') \* x')'，但它是在不形成y'逆的情况下计算的。

如果系统不是平方的，或者系数矩阵是奇异的，则计算最小范数解。

x ./ y

逐个元素的右除法。

x \ y

左部。这在概念上等同于表达式

inv (x) \* y

但是它的计算没有形成x的逆。

如果系统不是平方的，或者系数矩阵是奇异的，则计算最小范数解。

x .\ y

逐个元素左除法。y的每个元素除以x对应的每个元素。

x ^ y

电力运营商。如果x和y都是标量，则此运算符返回x的y次方。如果x是标量，y是方阵，则使用特征值展开计算结果。如果x是一个方阵，如果y是整数，则通过重复乘法计算结果，如果y不是整数，则通过特征值展开计算结果。如果x和y都是矩阵，则会产生错误。

这个操作符的实现需要改进。

x .^ y

逐元素幂运算符。如果两个操作数都是矩阵，则行数和列数必须一致，或者它们必须可以广播到相同的形状。如果可能有多个复杂结果，则取非负参数(角度)最小的结果。该规则可能返回复根，即使实根也是可能的。如果需要实际结果，请使用realpow、realsqrt、cbrt或nroot。

-x

无效

+x

加号。该操作符对操作数没有影响。

x’

复共轭转置。对于实参数，这个运算符与转置运算符相同。对于复杂参数，此操作符等价于表达式

conj (x.')

x.’

转置。

注意，因为Octave的逐元素运算符以'开头。，这样的语句可能存在歧义

1./m

因为周期既可以解释为常数的一部分，也可以解释为运算符的一部分。为了解决此冲突，Octave将该表达式视为您键入的

(1) ./ m

而不是

(1.) / m

尽管这与Octave的词法分析器的正常行为不一致，后者通常倾向于通过在任何给定点上选择最长可能的匹配来将输入分解为令牌，但在这种情况下它更有用。

还要注意，二进制操作符和空格的某些组合可能会在调用函数的Command Syntax形式中产生明显的歧义。有关Octave如何处理该语法的描述，请参阅命令语法和函数语法。

: *B =* **ctranspose** *(A)*

返回A的共轭转置复数。

这个函数和A'是等价的。

**参见:** transpose。

: *y =* **pagectranspose** *(A)*

返回n维数组A的逐页复共轭转置。

这相当于每个页面k对应A(:，:， k)'。

**参见:**页转置，页转置，排列。

: *C =* **ldivide** *(A, B)*

返回A和B逐个元素的左除法。

这个函数和A .\ B是等价的。

**参见:**rdivide, mldivide, times, plus。

: *C =* **minus** *(A, B)*

这个函数和A - B是等价的。

**参见:**plus, uminus。

: *C =* **mldivide** *(A, B)*

返回矩阵左除A和B。

这个函数和A \ B是等价的。

如果系统不是平方的，或者系数矩阵是奇异的，则计算最小范数解。

**参见:**mrdivide, ldivide, rdivide, linsolve。

: *C =* **mpower** *(A, B)*

返回A的B次方的矩阵幂运算。

这个函数和A ^ B是等价的。

**参见:**power, mtimes, plus, minus。

: *C =* **mrdivide** *(A, B)*

返回矩阵A和B的右除法。

这个函数和A / B是等价的。

如果系统不是平方的，或者系数矩阵是奇异的，则计算最小范数解。

**参见:**mldivide, rdivide, plus, minus。

: *C =* **mtimes** *(A, B)*

: *C =* **mtimes** *(A1, A2, …)*

返回输入的矩阵乘法乘积。

这个函数和A \* B是等价的。如果给出更多参数，则从左到右累积应用乘法:

(…((A1 \* A2) \* A3) \* …)

**参见:**times, plus, minus, rdivide, mrdivide, mldivide, mpower。

: *C =* **plus** *(A, B)*

: *C =* **plus** *(A1, A2, …)*

这个函数和A + B是等价的。

如果给出了更多的参数，则从左到右累计应用总和:

(…((A1 + A2) + A3) + …)

**参见:**minus, uplus。

: *C =* **power** *(A, B)*

返回A的B次方的逐元素运算。

这个函数和A ^ B是等价的。

如果可能有多个复杂结果，则返回具有最小非负参数(角度)的结果。如果需要实际结果，请使用realpow、realsqrt、cbrt或nroot。

**参见:**mpower, realpower, realsqrt, cbrt, nroot。

: *C =* **rdivide** *(A, B)*

返回A和B逐个元素的右除法。

这个函数和A / B是等价的。

**参见:**ldivide, mrdivide, times, plus。

: *C =* **times** *(A, B)*

: *C =* **times** *(A1, A2, …)*

返回输入的每个元素的乘法乘积。

这个函数和A .\* B是等价的。如果给出更多参数，则从左到右累积应用乘法:

(…((A1 .\* A2) .\* A3) .\* …)

**参见:**mtimes, rdivide。

: *B =* **transpose** *(A)*

返回A的转置。

这个函数和A。’是等价的。

**参见:** ctranspose。

: *B =* **pagetranspose** *(A)*

返回n维数组A的逐页转置。

这等价于A(:，:， k)。每一页k。

**参见:**pagectranspose, transpose, permute。

: *B =* **uminus** *(A)*

这个函数和- A是等价的。

**参见:** uplus, minus。

: *B =* **uplus** *(A)*

这个函数和+ A是等价的。

**参见:**uminus, plus。

**8.4比较运算符**

比较运算符比较数值之间的关系，如相等关系。它们是使用关系操作符编写的。

如果比较为真，Octave的所有比较运算符都返回值1，如果比较为假，则返回值0。对于矩阵值，它们都是在逐个元素的基础上工作的。广播规则适用。参见广播。例如:

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

⇒ 1 0

0 1

根据广播规则，如果一个操作数是标量，另一个操作数是矩阵，则该标量依次与矩阵的每个元素进行比较，结果与矩阵的大小相同。

x < y

当x小于y时成立。

x <= y

当x小于等于y时成立。

x == y

当x = y时成立。

x >= y

当x大于等于y时成立。

x > y

当x大于y时成立。

x != y

x ~= y

当x不等于y时成立。

对于复数，定义了以下排序:z1 < z2当且仅当

abs (z1) < abs (z2)

|| (abs (z1) == abs (z2) && arg (z1) < arg (z2))

这与max, min和sort使用的顺序一致，但与MATLAB不一致，MATLAB只比较实部。

字符串比较也可以用strcmp函数来执行，而不是用上面列出的比较操作符。参见字符串。

: *TF =* **eq** *(A, B)*

如果两个输入相等，则返回true。

这个函数等价于A == B。

**参见:**ne, isequal, le, ge, gt, ne, lt。

: *TF =* **ge** *(A, B)*

这个函数等价于A >= B。

**参见:**le, eq, gt, ne, lt。

: *TF =* **gt** *(A, B)*

这个函数等价于A > B。

**参见:**le, eq, ge, ne, lt。

: *tf =* **isequal** *(x1, x2, …)*

如果所有x1, x2，…都相等，则返回true。

**参见:**isequal。

: *tf =* **isequaln** *(x1, x2, …)*

如果在NaN == NaN的附加假设下，所有x1, x2，…都相等(不比较数据集中的NaN占位符)，则返回true。

**参见:**isequal。

: ***TF = le (A, B)***

这个函数等价于A <= B。

**参见:**eq, ge, gt, ne, lt。

: *TF =* **lt** *(A, B)*

这个函数等价于A < B。

**参见:**le, eq, ge, gt, ne。

: *TF =* **ne** *(A, B)*

如果两个输入不相等，则返回true。

这个函数等价于A != B。

**参见:**eq, isequal, le, ge, lt。

**8.5.1逐元素布尔运算符**

逐个元素的布尔表达式是使用布尔运算符“or”(“|”)、“and”(“&”)和“not”(“!”)的比较表达式的组合，以及用于控制嵌套的括号。布尔表达式的真值是通过组合组件表达式中相应元素的真值来计算的。如果一个值为零，则认为它为假，否则为真。

任何可以使用比较表达式的地方都可以使用逐个元素的布尔表达式。它们可以在if和while语句中使用。但是，在if或while语句中用作条件的矩阵值只有在其所有元素都非零时才为真。

与比较操作一样，逐元素布尔表达式的每个元素也有一个数值(如果为真则为1，如果为假则为0)，如果布尔表达式的结果存储在变量中或用于算术运算，则该值将发挥作用。

下面是三个逐元素布尔运算符的描述。

boolean1 & boolean2

如果boolean1和boolean2对应的元素都为真，则结果的元素为真。

boolean1 | boolean2

如果对应的boolean1或boolean2的任何一个元素为真，结果的元素为真。

! boolean

~ boolean

如果boolean的对应元素为假，则结果的每个元素都为真。

这些运算符在逐个元素的基础上工作。例如，表达式

[1, 0; 0, 1] & [1, 0; 2, 3]

返回一个2 × 2的单位矩阵。

对于二进制操作符，应用广播规则。参见广播。特别是，如果其中一个操作数是标量而另一个是矩阵，则该操作符将应用于标量和矩阵的每个元素。

对于二元逐个元素的布尔运算符，在计算结果之前先对子表达式布尔1和布尔2求值。当表达式有副作用时，这可能会产生影响。例如，在表达式

a & b++

即使变量a为零，变量b的值也会递增。

这种行为对于布尔运算符按照对矩阵值操作数的描述工作是必要的。

: *TF =* **and** *(x, y)*

: *TF =* **and** *(x1, x2, …)*

返回x和y的逻辑与。

此函数相当于运算符语法x & y。如果给出了两个以上的参数，则从左到右累计应用逻辑与运算:

(…((x1 & x2) & x3) & …)

**参见:**or, not, xor。

: *z =* **not** *(x)*

返回x的逻辑NOT。

这个函数等价于操作符语法!x。

**参见:**and, or, xor。

: *TF =* **or** *(x, y)*

: *TF =* **or** *(x1, x2, …)*

返回x和y的逻辑或。

此函数相当于运算符语法x | y。如果给出了两个以上的参数，则从左到右累计应用逻辑或:

(…((x1 | x2) | x3) | …)

**参见:**and, not, xor。

**8.5.2短路布尔运算符**

结合if和while条件中对标量值的隐式转换，Octave的逐元素布尔运算符通常足以执行大多数逻辑运算。然而，有时需要在确定了整个真值后立即停止对布尔表达式的求值。Octave的短路布尔运算符就是这样工作的。

boolean1 && boolean2

表达式boolean1被求值，并使用等价于all (boolean(:))的操作将其转换为标量。如果boolean1不是逻辑值，如果它的值非零，则认为它为真，如果它的值为零，则认为它为假。如果boolean是一个数组，则只有当它非空且所有元素都非零时才认为它为真。如果boolean的值为false，则整个表达式的结果为false。如果为真，则表达式boolean2的求值方式与boolean1相同。如果为真，则整个表达式的结果为真。否则，整个表达式的结果为假。

**警告:**计算all (boolean(:))的等价性的一个例外是当boolean为空数组时。为了与MATLAB兼容，空数组的真值始终为假，因此即使all([])为真，[]&& true的计算结果也为假。

boolean1 || boolean2

表达式boolean1被求值，并使用等价于all (boolean(:))的操作将其转换为标量。如果boolean1不是逻辑值，如果它的值非零，则认为它为真，如果它的值为零，则认为它为假。如果boolean是一个数组，则只有当它非空且所有元素都非零时才认为它为真。如果boolean的值为true，则整个表达式的结果为true。如果为false，则表达式boolean2的求值方式与boolean1相同。如果为真，则整个表达式的结果为真。否则，整个表达式的结果为假。

**警告:**空矩阵的真值总是false，详细信息请参见前面的列表项。

在确定表达式的整体真值之前，两个操作数可能不会被求值，这一点很重要。例如，在表达式

a && b++

只有当变量a非零时，变量b的值才递增。

这可以用来编写更简洁的代码。例如，可以写

function f (a, b, c)

if (nargin > 2 && ischar (c))

…

而不是使用两个if语句来避免尝试计算一个不存在的参数。例如，如果没有短路特性，则必须编写

function f (a, b, c)

if (nargin > 2)

if (ischar (c))

…

写作

function f (a, b, c)

if (nargin > 2 & ischar (c))

…

如果使用一个或两个参数调用f将导致错误，因为Octave将被迫尝试计算操作符' & '的两个操作数。

MATLAB有特殊的行为，允许运算符' & '和' | '在if和while语句的真值表达式中使用时短路。Octave的行为与兼容性相同，但是，强烈不鼓励以这种方式使用' & '和' | '操作符，并将发出警告。相反，您应该使用总是具有短路行为的' && '和' || '操作符。

最后，在Octave中不支持三元操作符(?:)。如果短路不重要，可以用ifelse函数代替。

: *M =* **merge** *(mask, tval, fval)*

: *M =* **ifelse** *(mask, tval, fval)*

根据mask的值合并true\_val和false\_val的元素。

如果mask是逻辑标量，则其他两个参数可以是任意值。否则，mask必须为逻辑数组，tval、fval应为匹配类的数组或单元格数组。在标量掩码的情况下，如果mask为真则返回tval，否则返回fval。

在数组掩码的情况下，tval和fval必须是标量或维度等于mask的数组。结果构造如下:

result(mask) = tval(mask);

result(! mask) = fval(! mask);

掩码也可以是任意数字类型，在这种情况下，它首先被转换为逻辑类型。

编程说明：ifelse 是 merge 的别名，可以互换使用。

**参见:**logical, diff。

**8.6赋值表达式**

赋值是将新值存储到变量中的表达式。例如，下面的表达式将值1赋给变量z:

z = 1

该表达式执行后，变量z的值为1。赋值之前z的旧值将被遗忘。“=”号称为赋值操作符。

赋值也可以存储字符串值。例如，下面的表达式将在变量message中存储值"this food is good":

thing = "food"

predicate = "good"

message = [ "this " , thing , " is " , predicate ]

⇒ "this food is good"

(这也说明了字符串的连接。)

大多数操作符(加法、连接等)除了计算一个值之外没有任何作用。如果忽略该值，不如不使用该操作符。赋值操作符则不同。它确实产生了一个值，但是即使忽略了这个值，这个赋值仍然会通过变量的改变让人感觉到。我们称之为副作用。

赋值操作的左操作数不一定是变量(参见变量)。它也可以是矩阵的一个元素(参见索引表达式)或返回值列表(参见调用函数)。这些都被称为左值，这意味着它们可以出现在赋值操作符的左侧。右操作数可以是任何表达式。它产生新值，赋值函数将新值存储在指定的变量、矩阵元素或返回值列表中。

重要的是要注意变量没有永久类型。变量的类型就是它当前所保存的值的类型。在下面的程序片段中，变量foo首先是一个数值，然后是一个字符串值:

>> foo = 1

foo = 1

>> foo = "bar"

foo = bar

当第二次赋值给foo一个字符串值时，它之前有一个数值的事实被忘记了。

将标量赋值给索引矩阵，将由索引引用的所有元素设置为该标量值。例如，如果a是一个至少有两列的矩阵，

a(:, 2) = 5

将a的第二列中的所有元素设置为5。

当赋值将矢量、矩阵或数组元素的值设置在该变量当前大小之外的位置或维度时，数组大小将增加以适应新值：

>> a = [1, 2, 3]

a = 1 2 3

>> a(4) = 4

a = 1 2 3 4

>> a(2, :) = [5, 6, 7, 8]

a =

1 2 3 4

5 6 7 8

尝试增加数组的大小，使所需的输出大小不明确将导致错误：

>> a(9) = 10

-| error: Invalid resizing operation or ambiguous assignment to an

out-of-bounds array element

这是因为添加第 9 个元素会在值 10 的所需数组位置产生歧义，每种可能性都需要不同的数组大小扩展来适应赋值。

只要分配是明确的，就可以使用比填充新扩展数组所需的更少的指定元素进行分配。在这些情况下，数组将自动填充 null 值：

>> a = [1, 2]

a = 1 2

>> a(4) = 5

a = 1 2 0 5

>> a(3, :) = [6, 7, 8, 9]

a =

1 2 0 5

0 0 0 0

6 7 8 9

>> a(4, 5) = 10

a =

1 2 0 5 0

0 0 0 0 0

6 7 8 9 0

0 0 0 0 10

对于所有内置类型，null 值将适用于该对象类型。

数值数组：

>> a = int32 ([1, 2])

a = 1, 2

>> a(4) = 5

a = 1 2 0 5

逻辑数组:

>> a = [true, false, true]

a = 1 0 1

>> d(5) = true

d = 1 0 1 0 1

字符数组

>> a = "abc"

a = abc

>> a(5) = "d"

a = abcd

>> double (a)

ans = 97 98 99 0 100

单元阵列：

>> e = {1, "foo", [3, 4]};

>> e(5) = "bar"

e =

{

[1,1] = 1

[1,2] = foo

[1,3] =

3 4

[1,4] = [](0x0)

[1,5] = bar

}

结构数组:

>> a = struct("foo",1,"bar",2);

>> a(3) = struct("foo",3,"bar",9)

a =

1x3 struct array containing the fields:

foo

bar

>> a.foo

ans = 1

ans = [](0x0)

ans = 3

>> a.bar

ans = 2

ans = [](0x0)

ans = 9

请注意，Octave 目前无法将任意对象类型连接到数组中。必须在对象类中显式定义此类行为，否则尝试串联将导致错误。请参阅面向对象编程

在大多数情况下，分配一个空矩阵“[]”可以删除矩阵和矢量的行或列。请参阅空矩阵。例如，给定一个 4 x 5 矩阵 A，赋值

A (3, :) = []

删除 A 的第三行,以及赋值

A (:, 1:2:5) = []

删除第一列、第三列和第五列。

删除数组对象的一部分必然会调整对象的大小。当删除允许在整个维度（例如，矢量的一个元素或矩阵的一行或一列）中一致地减小大小时，沿该维度的大小将减小，同时保留维度。但是，如果无法保持维度，则对象将按照按列元素排序重新塑造为矢量：

>> a = [1, 2, 3, 4; 5, 6, 7, 8]

a =

1 2 3 4

5 6 7 8

>> a(:, 3) = []

a =

1 2 4

5 6 8

>> a(4) = []

a = 1 5 2 4 8

赋值是一个表达式，因此它有一个值。因此，z = 1 作为表达式的值为 1。这样做的一个结果是，您可以一起编写多个作业：

x = y = z = 0

在所有三个变量中存储值 0。这样做是因为 z = 0 的值（即 0）存储到 y 中，然后 y = z = 0（即 0）的值存储到 x 中。

对值列表的赋值也是如此，因此以下是一个有效的表达式

[a, b, c] = [u, s, v] = svd (a)

这完全等同于

[u, s, v] = svd (a)

a = u

b = s

c = v

在这样的表达式中，表达式每个部分的值数不需要匹配。例如，表达式

[a, b] = [u, s, v] = svd (a)

相当于

[u, s, v] = svd (a)

a = u

b = s

但是，表达式左侧的值数不能超过右侧的值数。例如，以下情况将产生错误。

[a, b, c, d] = [u, s, v] = svd (a);

-| error: element number 4 undefined in return list

符号 ~ 可以用作左值列表中的占位符，表示应忽略相应的返回值，不要存储在任何地方：

[~, s, v] = svd (a);

这比使用虚拟变量更简洁、更节省内存。右侧表达式的 nargout 值不受影响。如果将赋值用作表达式，则返回值是一个逗号分隔的列表，其中删除了忽略的值。

一个非常常见的编程模式是用给定的值递增现有变量，如下所示

a = a + 2;

这可以使用 += 运算符以更清晰、更简洁的形式编写

a += 2;

减法 （-=）、乘法 （\*=） 和除法 （/=） 也存在类似的运算符。表单的表达式

expr1 op= expr2

被评价为

expr1 = (expr1) op (expr2)

其中 op 可以是 +、-、\* 或 /，只要 expr2 是没有副作用的简单表达式。如果 expr2 还包含赋值运算符，则此表达式的计算结果为

temp = expr2

expr1 = (expr1) op temp

其中 temp 是一个占位符临时值，用于存储计算 expr2 的计算结果。所以，表达式

a \*= b+1

被评价为

a = a \* (b+1)

而且不

a = a \* b + 1

您可以在需要表达式的任何位置使用赋值。例如，写入 x ！= （y = 1） 将 y 设置为 1，然后测试 x 是否等于 1 是有效的。但这种风格往往会使进程难以阅读。除了一次性进程之外，您应该重写它以摆脱这种赋值嵌套。这从来都不是很难。

**8.7自增操作符**

自增运算符将变量的值增加或减少1。对变量进行自增的运算符写成' ++ '。它可用于在取值之前或之后对变量进行递增。

例如，要预先增加变量x，您可以编写++x。这将给x加1，然后返回x的新值作为表达式的结果。它和表达式x = x + 1完全一样。

要对变量x进行后增量，可以编写x++。这将变量x加1，但返回x加1之前的值。例如，如果x等于2，则表达式x++的结果为2，而x的新值为3。

对于矩阵和向量实参，自增和自减操作符作用于操作数的每个元素。

自增和自减操作符必须“紧抱”对应的变量。这意味着，在这些操作符和它们所影响的变量之间不允许有空格。

下面是所有自增和自减表达式的列表。

++x

该表达式将变量x加1，该表达式的值为x的新值，相当于表达式x = x + 1。

--x

该表达式对变量x进行减量，该表达式的值为x的新值，相当于表达式x = x - 1。

x++

这个表达式导致变量x递增。表达式的值是x的旧值。

x--

这个表达式导致变量x递减。表达式的值是x的旧值。

**8.8操作符优先级**

当不同的操作符在一个表达式中相邻出现时，操作符优先级决定如何对操作符进行分组。例如，' \* '的优先级高于' + '。因此，表达式a + b \* c意味着将b与c相乘，然后将a加到乘积上(即a + (b \* c))。

可以使用圆括号来否决运算符的优先级。你可以把优先级规则理解为如果你自己不写括号，那么括号是假定的。事实上，当您有一个不寻常的操作符组合时，使用括号是明智的，因为阅读程序的其他人可能不记得在这种情况下的优先级是什么。你可能也会忘记，然后你也会犯错误。显式括号将有助于防止任何此类错误。

当优先级相等的操作符一起使用时，最左边的操作符首先分组，除了赋值操作符，其分组顺序相反。因此，表达式a - b + c归为(a - b) + c，而表达式a = b = c归为a = (b = c)。

当操作数后面跟着另一个操作符时，前缀一元操作符的优先级很重要。例如，-x^2表示-(x^2)，因为' - '的优先级低于' ^ '。

下面是一个Octave中运算符的表，按优先级递减的顺序排列。除非特别说明，所有操作员从左到右分组。

function call and array indexing, cell array indexing, and structure element indexing

‘()’ ‘{}’ ‘.’

postfix increment, and postfix decrement

‘++’ ‘--’

这些算子从右向左分组。

transpose and exponentiation

‘'’ ‘.'’ ‘^’ ‘.^’

unary plus, unary minus, prefix increment, prefix decrement, and logical "not"

‘+’ ‘-’ ‘++’ ‘--’ ‘~’ ‘!’

multiply and divide

‘\*’ ‘/’ ‘\’ ‘.\’ ‘.\*’ ‘./’

add, subtract

‘+’ ‘-’

colon

‘:’

relational

‘<’ ‘<=’ ‘==’ ‘>=’ ‘>’ ‘!=’ ‘~=’

element-wise "and"

‘&’

element-wise "or"

‘|’

logical "and"

‘&&’

logical "or"

‘||’

assignment

‘=’ ‘+=’ ‘-=’ ‘\*=’ ‘/=’ ‘\=’ ‘^=’ ‘.\*=’ ‘./=’ ‘.\=’ ‘.^=’ ‘|=’ ‘&=’

这些算子从右向左分组。

**9评估**

通常，只需在Octave提示符下输入表达式，或者让Octave解释保存在文件中的命令，就可以对表达式求值。

有时，您可能会发现有必要对已经计算并存储在字符串中的表达式求值，这正是eval函数允许您做的。

: **eval** *(try)*

: **eval** *(try, catch)*

解析字符串，并将其作为一个Octave程序进行计算。

如果执行失败，计算可选的字符串catch。

字符串try在当前上下文中求值，因此在eval返回后任何结果仍然可用。

下面的示例在当前工作空间中创建变量A，其近似值为3.1416。

eval ("A = acos(-1);");

如果在对try求值期间发生错误，则对catch字符串求值，如下例所示:

eval ('error ("This is a bad example");',

'printf ("This error occurred:\n%s\n", lasterr ());');

-| This error occurred:

This is a bad example

编程注意:如果你只是将eval用作错误捕获机制，而不是用于执行任意代码串，请考虑使用try/catch块或unwind\_protect/unwind\_protect\_cleanup块。这些技术具有更高的性能，并且不像评估任意代码那样引入安全考虑。

**参见:**evalin, evalc, assignin, feval。

evalc函数还捕获由求值表达式产生的任何控制台输出。

: *s =* **evalc** *(try)*

: *s =* **evalc** *(try, catch)*

将字符串try当作一个Octave程序来解析和求值，同时将输出捕获到返回变量s中。

如果执行失败，计算可选的字符串catch。

这个函数的行为类似于eval，但是通常会写入控制台的任何输出或警告消息都会被捕获并在字符串s中返回。

在执行此函数期间，日志将被禁用。当使用system时，外部程序产生的任何输出都不会被捕获，除非它们的输出被系统函数本身捕获。

s = evalc ("t = 42"), t

⇒ s = t = 42

⇒ t = 42

**参见:**eval, diary。

**9.1按函数名调用函数**

feval函数允许您从包含其名称的字符串调用函数。这在编写需要调用用户提供的函数的函数时非常有用。feval函数以要调用的函数的名称作为其第一个参数，其余的参数都给函数。

下面的示例是一个使用feval的简单函数，它使用牛顿方法找到用户提供的一个变量函数的根。

function result = newtroot (fname, x)

# usage: newtroot (fname, x)

#

# fname : a string naming a function f(x).

# x : initial guess

delta = tol = sqrt (eps);

maxit = 200;

fx = feval (fname, x);

for i = 1:maxit

if (abs (fx) < tol)

result = x;

return;

else

fx\_new = feval (fname, x + delta);

deriv = (fx\_new - fx) / delta;

x = x - fx / deriv;

fx = fx\_new;

endif

endfor

result = x;

endfunction

请注意，这只是调用用户提供的函数的一个例子，不应该太认真。除了使用更健壮的算法之外，任何严肃的代码都会检查所有参数的数量和类型，确保提供的函数确实是一个函数，等等。有关数字对象的谓词列表，请参见数字对象的谓词，有关exist函数的描述，请参见变量的状态。

: *retval =* **feval** *(name, …)*

求名为name的函数的值。

第一个参数之后的任何参数都作为输入传递给命名函数。例如,

feval ("acos", -1)

⇒ 3.1416

使用参数' -1 '调用函数。

函数feval也可以与任何类型的函数句柄一起使用(参见函数句柄)。历史上，feval是调用用户提供的字符串函数的唯一方法，但函数句柄现在更受欢迎，因为它们提供了更清晰的语法。例如,

f = @exp;

feval (f, 1)

⇒ 2.7183

f (1)

⇒ 2.7183

是调用f所引用的函数的等效方法。如果无法事先预测f是函数句柄、字符串中的函数名还是内联函数，则可以使用feval代替。

存在一个类似的函数run，用于调用用户脚本文件，这些文件不一定位于用户路径上

: **run** *script*

: **run** *("script")*

在当前工作区中运行脚本。

在Octave的加载路径中指定的目录中，并且以.m扩展名结尾的脚本，可以通过简单地键入它们的名称来运行。对于不在加载路径上的脚本，使用run。

文件名脚本可以是裸文件名、完全限定文件名或相对文件名，并且可以带或不带文件扩展名。如果没有指定扩展名，Octave将首先搜索扩展名为.m的脚本，然后再返回到没有扩展名的脚本名称。

实现注意:如果脚本包含一个path组件，那么run首先将工作目录更改为找到脚本的目录。接下来，执行脚本。最后，run返回到原始工作目录，除非脚本特别更改了目录。

参见:path, addpath, source。

**9.2不同环境下的评估**

在计算表达式之前，需要替换表达式中使用的变量的值。这些都存储在符号表中。每当解释器启动一个新函数时，它都会保存当前的符号表并创建一个新函数，并使用函数参数列表和一对预定义变量(如nargin)对其进行初始化。函数内的表达式使用新的符号表。

有时你想写一个函数，这样当你调用它的时候，它就会在你自己的上下文中修改变量。这允许您使用按名称传递样式的函数，这类似于在C等编程语言中使用指针。

考虑如何将save和load写入m文件。例如:

function create\_data

x = linspace (0, 10, 10);

y = sin (x);

save mydata x y

endfunction

使用evalin，你可以这样写save:

function save (file, name1, name2)

f = open\_save\_file (file);

save\_var (f, name1, evalin ("caller", name1));

save\_var (f, name2, evalin ("caller", name2));

endfunction

这里，' caller '是create\_data函数，name1是字符串"x"，它的计算结果只是x的值。

稍后，你想在不同的上下文中从mydata加载值:

function process\_data

load mydata

… do work …

endfunction

通过赋值，你可以这样写load:

function load (file)

f = open\_load\_file (file);

[name, val] = load\_var (f);

assignin ("caller", name, val);

[name, val] = load\_var (f);

assignin ("caller", name, val);

endfunction

这里，' caller '是process\_data函数。

您可以在命令提示符下使用上下文' base '而不是' caller '来设置和使用变量。

这些函数在实际中很少使用。一个例子是fail (' code '， ' pattern ')函数，它在调用者的上下文中计算' code '，并检查它产生的错误消息是否与给定的模式匹配。其他例子，如保存和加载是用c++编写的，其中所有Octave变量都在' caller '上下文中，不需要evalin。

: **evalin** *(context, try)*

: **evalin** *(context, try, catch)*

与eval类似，不同之处在于表达式是在上下文context中求值的，上下文可以是"caller"或"base"。

**参见:**eval, assignin。

:**assignin** *(context, varname, value)*

给context context中的varname赋值，它可以是"base"或"caller"。

**参见:**evalin。

**10语句**

语句可以是简单的常量表达式，也可以是嵌套循环和条件语句的复杂列表。

控制语句如if, while等控制着Octave程序的执行流程。所有控制语句都以if和while等特殊关键字开头，以区别于简单表达式。许多控制语句包含其他语句;例如，if语句包含另一个语句，该语句可能被执行，也可能不被执行。

每个控制语句都有一个相应的结束语句，它标志着控制语句的结束。例如，关键字endif标记if语句的结束，而endwhile标记while语句的结束。您可以在任何需要更具体的结束关键字的地方使用关键字end，但首选使用更具体的关键字，因为如果使用它们，Octave能够为不匹配或缺失的结束令牌提供更好的诊断。

在if或while等关键字和相应的结束语句之间包含的语句列表称为控制语句的主体。

**10.1 if语句**

if语句是Octave的决策语句。if语句有三种基本形式。最简单的形式是这样的:

if (condition)

then-body

endif

Condition是一个表达式，它控制语句的其余部分将执行什么操作。只有当condition为真时才执行then-body。

if语句中的条件如果其值非零则认为为真，如果其值为零则认为为假。如果If语句中的条件表达式的值是一个向量或矩阵，则只有当它非空且所有元素都非零时，才认为它为真。条件为矩阵时的概念等效代码如下所示。

if (matrix) ≡ if (all (matrix(:)))

if语句的第二种形式是这样的:

if (condition)

then-body

else

else-body

endif

如果condition为真，则执行then-body;否则，else-body被执行。

下面是一个例子:

if (rem (x, 2) == 0)

printf ("x is even\n");

else

printf ("x is odd\n");

endif

在本例中，如果表达式rem (x, 2) == 0为真(即x的值可以被2整除)，则计算第一个printf语句，否则计算第二个printf语句。

第三种也是最常见的if语句形式允许将多个决策合并到一个语句中。它是这样的:

if (condition)

then-body

elseif (condition)

elseif-body

else

else-body

endif

可以出现任意数量的elseif子句。依次测试每个条件，如果发现其中一个条件为真，则执行其相应的主体。如果所有条件都不为真，并且存在else子句，则执行其子句体。只能出现一个else子句，并且必须是语句的最后一部分。

在下面的示例中，如果第一个条件为真(即x的值可以被2整除)，则执行第一个printf语句。如果它为假，则测试第二个条件，如果它为真(即x的值可以被3整除)，则执行第二个printf语句。否则，执行第三条printf语句。

if (rem (x, 2) == 0)

printf ("x is even\n");

elseif (rem (x, 3) == 0)

printf ("x is odd and divisible by 3\n");

else

printf ("x is odd\n");

endif

注意，elseif关键字不能拼写为elseif，这在Fortran中是允许的。如果是，else和If之间的空格将告诉Octave将其视为另一个If语句的else子句中的新If语句。例如，如果你写

if (c1)

body-1

else if (c2)

body-2

endif

Octave需要额外的输入来完成第一个if语句。如果您正在交互式地使用Octave，它将继续提示您进行额外输入。如果Octave从文件中读取这个输入，它可能会抱怨缺少或不匹配的结束语句，或者，如果你没有使用更具体的结束语句(endif, endfor等)，它可能会简单地产生不正确的结果，而不会产生任何警告消息。

如果我们像这样重写上面的语句，就更容易看到错误，

if (c1)

body-1

else

if (c2)

body-2

endif

使用缩进来显示Octave如何对语句进行分组。参见函数和脚本。

**10.2 switch语句**

根据一个变量的值采取不同的操作是很常见的。这可以通过以下方式使用if语句实现

if (X == 1)

do\_something ();

elseif (X == 2)

do\_something\_else ();

else

do\_something\_completely\_different ();

endif

然而，这种代码的编写和维护都非常麻烦。为了克服这个问题，Octave支持switch语句。使用这个语句，上面的例子变成

switch (X)

case 1

do\_something ();

case 2

do\_something\_else ();

otherwise

do\_something\_completely\_different ();

endswitch

这段代码使问题的重复结构更加明确，使代码更容易阅读，从而更容易维护。此外，如果变量X应该更改其名称，则只需更改一行，而使用if语句时则需要更改一行。

switch语句的一般形式是

switch (expression)

case label

command\_list

case label

command\_list

…

otherwise

command\_list

endswitch

其中label可以是任何表达式。但是，不会检测到重复的标签值，并且只执行与第一个匹配对应的command\_list。要使switch语句有意义，必须存在至少一个case label command\_list子句，否则command\_list子句是可选的。

如果label是一个单元格数组，则如果单元格数组中的任何元素匹配表达式，则执行相应的command\_list。作为一个例子，下面的程序将输出' Variable is要么6要么7 '。

A = 7;

switch (A)

case { 6, 7 }

printf ("variable is either 6 or 7\n");

otherwise

printf ("variable is neither 6 nor 7\n");

endswitch

与所有其他特定的结束关键字一样，endswitch可以被end替换，但是如果使用特定的形式，可以获得更好的诊断。

与使用if语句相比，使用switch语句的一个优点是标签可以是字符串。如果使用If语句，则不可能编写

if (X == "a string") # This is NOT valid

因为X和字符串之间将进行字符对字符的比较，而不是计算字符串是否相等。这种特殊情况由switch语句处理，并且可以编写像这样的程序

switch (X)

case "a string"

do\_something

…

endswitch

**10.2.1 C程序员注意事项**

在广泛使用的C编程语言中也可以使用switch语句。然而，在八度音阶和C中的语句之间存在一些差异

用例是排他的，所以它们不会像C语言的switch语句中的用例那样“失败”。

command\_list元素不是可选的。将列表设为可选意味着需要在标签和命令列表之间使用分隔符。否则，像

switch (foo)

case (1) -2

…

会产生惊人的结果吗

switch (foo)

case (1)

case (2)

doit ();

…

特别是对于C程序员。如果doit()应该在foo为1或2时执行，上面的代码应该用这样的单元格数组编写

switch (foo)

case { 1, 2 }

doit ();

…

**10.3 while语句**

在编程中，循环是指程序的一部分被(或至少可以)连续执行两次或两次以上。

while语句是Octave中最简单的循环语句。只要条件为真，它就重复执行语句。与if语句中的条件一样，如果while语句中的条件的值为非零，则认为其为真，如果其值为零，则认为其为假。如果while语句中的条件表达式的值是一个向量或矩阵，则只有当它非空且所有元素都非零时，才认为它为真。

Octave的while语句是这样的:

while (condition)

body

endwhile

这里的body是一个语句或语句列表，我们称之为循环体，condition是一个表达式，控制循环持续运行多久。

while语句做的第一件事是测试条件。如果condition为真，则执行语句体。body执行后，再次测试condition，如果它仍然为真，则再次执行body。这个过程不断重复，直到condition不再为真。如果condition初始值为false，则循环体永远不会执行。

这个例子创建了一个变量fib，其中包含斐波那契数列的前十个元素。

fib = ones (1, 10);

i = 3;

while (i <= 10)

fib (i) = fib (i-1) + fib (i-2);

i++;

endwhile

这里循环体包含两个语句。

循环是这样工作的:首先，将i的值设置为3。然后，while测试i是否小于或等于10。这是当i等于3时的情况，因此fib的第i个元素的值被设置为序列中前两个值的和。然后i++增加i的值并重复循环。当i达到11时，循环终止。

条件和语句体之间不需要换行;但是使用它可以使程序更清晰，除非主体非常简单。

**10.4 do-until语句**

do-until语句类似于while语句，不同之处在于它重复执行语句，直到条件为真，并且在循环结束时对条件进行测试，因此循环体总是至少执行一次。与if语句中的条件一样，如果do-until语句的值非零，则认为其条件为真，如果其值为零，则认为其条件为假。如果do-until语句中的条件表达式的值是一个向量或矩阵，则只有当它非空且所有元素都非零时才认为它为真。

Octave的do-until语句是这样的:

do

body

until (condition)

这里的body是一个语句或语句列表，我们称之为循环体，condition是一个表达式，控制循环持续运行多久。

这个例子创建了一个变量fib，其中包含斐波那契数列的前十个元素。

fib = ones (1, 10);

i = 2;

do

i++;

fib (i) = fib (i-1) + fib (i-2);

until (i == 10)

在do关键字和正文之间不需要换行符;但是使用它可以使程序更清晰，除非主体非常简单。

**10.5 for语句**

for语句可以更方便地计算循环的迭代次数。for语句的一般形式是这样的:

for var = expression

body

endfor

其中body代表任何语句或语句列表，expression是任何有效表达式，var可以有多种形式。通常它是一个简单的变量名或索引变量。如果表达式的值是一个结构体，var也可以是一个包含两个元素的vector。请参阅下面的结构元素循环。

for语句中的赋值表达式的工作方式与Octave的普通赋值语句略有不同。它不是将表达式的完整结果赋值，而是将表达式的每一列依次赋值给var。如果expression是一个范围、行向量或标量，则每次执行循环体时var的值将是一个标量。如果var是列向量或矩阵，则每次循环体执行时，var都将是列向量。

下面的例子展示了另一种创建包含斐波那契数列前十个元素的向量的方法，这次使用了for语句:

fib = ones (1, 10);

for i = 3:10

fib(i) = fib(i-1) + fib(i-2);

endfor

这段代码首先对表达式3:10求值，生成一个范围从3到10的值。然后将变量i赋值为范围的第一个元素，循环体执行一次。当到达循环体的末尾时，将范围内的下一个值赋给变量i，并再次执行循环体。这个过程一直持续到没有更多的元素可以分配为止。

在Octave中，也可以使用for语句迭代矩阵或单元数组。例如，考虑

disp ("Loop over a matrix")

for i = [1,3;2,4]

i

endfor

disp ("Loop over a cell array")

for i = {1,"two";"three",4}

i

endfor

在这种情况下，变量i取矩阵或单元矩阵列的值。因此，第一个循环迭代两次，产生两个列向量[1;2]，然后是[3;4]，同样地，循环遍历单元格数组。这可以扩展到多维数组上的循环。例如:

a = [1,3;2,4]; c = cat (3, a, 2\*a);

for i = c

i

endfor

在上述情况下，多维矩阵c被重塑为二维矩阵，如重塑(c, rows (c)， prod (size (c)(2:end)))，然后产生与二维矩阵上的循环相同的行为。

虽然可以将所有的for循环重写为while循环，但Octave语言具有这两种语句，因为for循环通常既不需要输入，又更容易理解。计数迭代次数在循环中非常常见，将此计数视为循环的一部分而不是在循环中执行的操作可能更容易。

**10.5.1遍历结构元素**

for语句的一种特殊形式允许你循环遍历结构体的所有元素:

for [ val, key ] = expression

body

endfor

在这种形式的for语句中，表达式的值必须是一个结构。如果是，则依次将key和val设置为元素的名称和相应的值，直到没有其他元素为止。例如:

x.a = 1

x.b = [1, 2; 3, 4]

x.c = "string"

for [val, key] = x

key

val

endfor

-| key = a

-| val = 1

-| key = b

-| val =

-|

-| 1 2

-| 3 4

-|

-| key = c

-| val = string

元素的访问没有任何特定的顺序。如果您需要以特定的方式循环遍历列表，则必须使用函数字段名并自己对列表进行排序。

**10.6 break语句**

break语句跳出最内层的while、do-until或for循环。break语句只能在循环体中使用。下面的例子找到给定整数的最小除数，并确定质数:

num = 103;

div = 2;

while (div\*div <= num)

if (rem (num, div) == 0)

break;

endif

div++;

endwhile

if (rem (num, div) == 0)

printf ("Smallest divisor of %d is %d\n", num, div)

else

printf ("%d is prime\n", num);

endif

当第一个while语句的余数为零时，Octave立即跳出循环。这意味着Octave立即进入循环后的语句并继续处理。(这与exit语句非常不同，exit语句会停止整个Octave程序。)

这是另一个与前一个程序等价的程序。它演示了如何将while语句的条件替换为if语句中的break:

num = 103;

div = 2;

while (1)

if (rem (num, div) == 0)

printf ("Smallest divisor of %d is %d\n", num, div);

break;

endif

div++;

if (div\*div > num)

printf ("%d is prime\n", num);

break;

endif

endwhile

**10.7 continue语句**

continue语句和break语句一样，只在while、do-until或for循环中使用。它跳过循环体的其余部分，导致循环周围的下一个循环立即开始。与此形成对比的是break，它完全跳出循环。下面是一个例子:

# print elements of a vector of random

# integers that are even.

# first, create a row vector of 10 random

# integers with values between 0 and 100:

vec = round (rand (1, 10) \* 100);

# print what we're interested in:

for x = vec

if (rem (x, 2) != 0)

continue;

endif

printf ("%d\n", x);

endfor

如果vec的一个元素是奇数，则本例跳过该元素的print语句，并继续返回循环中的第一个语句。

这不是continue语句的一个实际示例，但它应该让您清楚地了解它是如何工作的。通常，人们可能会这样写循环:

for x = vec

if (rem (x, 2) == 0)

printf ("%d\n", x);

endif

endfor

**10.8 unwind\_protect语句**

Octave支持一种有限形式的异常处理，模仿Lisp的unwind-protect形式。

unwind\_protect块的一般形式是这样的:

unwind\_protect

body

unwind\_protect\_cleanup

cleanup

end\_unwind\_protect

其中body和cleanup都是可选的，可以包含任何Octave表达式或命令。无论控制如何退出主体，清理中的语句都保证执行。

这对于防止对全局变量的临时更改出现可能的错误很有用。例如，下面的代码将始终恢复全局变量frobnosticate的原始值，即使在unwind\_protect块的第一部分发生错误。

save\_frobnosticate = frobnosticate;

unwind\_protect

frobnosticate = true;

…

unwind\_protect\_cleanup

frobnosticate = save\_frobnosticate;

end\_unwind\_protect

如果没有unwind\_protect，如果在计算unwind\_protect块的第一部分时发生错误，frobnosticate的值将不会被恢复，因为计算将在错误点停止，并且不会执行恢复值的语句。

除了unwind\_protect之外，Octave还支持另一种形式的异常处理，即try块。

**10.9 try语句**

try块的原始形式是这样的:

try

body

catch

cleanup

end\_try\_catch

其中body和cleanup都是可选的，可以包含任何Octave表达式或命令。清理中的语句只在body中出现错误时执行。

执行body时不打印任何警告或错误消息。如果在body执行过程中确实发生了错误，那么清理可以使用函数lasterr或lasterror来访问将要打印的消息文本及其标识符。另一种形式是，

try

body

catch err

cleanup

end\_try\_catch

将自动将lastror的输出存储在err结构中。有关lasterr和lasterror函数的详细信息，请参阅错误和警告。

**10.10延续线**

在Octave语言中，大多数语句以换行符结束，您必须告诉Octave忽略换行符，以便从一行继续到下一行。以字符结尾的行…在被Octave的解析器划分为token之前，将它们与下面的行连接。例如，线条

x = long\_variable\_name ...

+ longer\_variable\_name ...

- 42

形成一个单独的陈述。

在延续标记和换行符之间的任何文本都将被忽略。例如，语句

x = long\_variable\_name ... # comment one

+ longer\_variable\_name ...comment two

- 42 # last comment

等价于上面所示的。

在双引号字符串常量中，字符\必须用作延续标记。\必须出现在换行符之前的行尾:

s = "This text starts in the first line \

and is continued in the second line."

括号内的输入可以继续到下一行，而不必使用延续标记。例如，可以编写这样的语句

if (fine\_dining\_destination == on\_a\_boat

|| fine\_dining\_destination == on\_a\_train)

seuss (i, will, not, eat, them, sam, i, am, i,

will, not, eat, green, eggs, and, ham);

endif

而不必使用延续标记来增加混乱。

**11功能和脚本**

复杂的Octave程序通常可以通过定义函数来简化。函数可以在交互式Octave会话期间直接在命令行上定义，也可以在外部文件中定义，并且可以像调用内置函数一样调用。

**11.1函数和脚本文件简介**

本节涉及7个不同的内容。

1. 在命令提示符处输入一个函数。
2. 将一组命令存储在一个称为脚本文件的文件中。
3. 将函数存储在文件中，称为函数文件。
4. 函数文件中的子函数。
5. 在一个脚本文件中包含多个函数。
6. 私有函数。
7. 嵌套函数。

函数文件和脚本文件都以。m的扩展名结尾，以便与MATLAB兼容。如果你想在一个文件中有多个独立的函数，它必须是一个脚本文件(参见脚本文件)，并且要使用这些函数，你必须在使用脚本文件中的函数之前执行脚本文件。

**11.2定义函数**

在其最简单的形式中，名为name的函数的定义如下:

function name

body

endfunction

一个有效的函数名就像一个有效的变量名:由字母、数字和下划线组成的序列，而不是以数字开头。函数与变量共享相同的名称池。

函数体由Octave语句组成。它是定义中最重要的部分，因为它说明了函数应该做什么。

例如，这里有一个函数，当它被执行时，会在你的终端上敲响铃铛(假设它可以这样做):

function wakeup

printf ("\a");

endfunction

printf语句(参见输入和输出)只是告诉Octave打印字符串"\a"。特殊字符' \a '表示警告字符(ASCII 7)。参见字符串。

一旦定义了这个函数，您就可以通过输入函数名来要求Octave对其求值。

通常，您需要将一些信息传递给您定义的函数。在Octave中向函数传递参数的语法是

function name (arg-list)

body

endfunction

其中arg-list是一个以逗号分隔的函数参数列表。当函数被调用时，参数名被用来保存调用中给出的参数值。参数列表可以为空，在这种情况下，这种形式相当于上面所示的形式。

要在敲钟的同时打印一条消息，你可以这样修改唤醒:

function wakeup (message)

printf ("\a%s\n", message);

endfunction

使用如下语句调用这个函数

wakeup ("Rise and shine!");

将导致Octave敲响您的终端的铃声并打印消息“Rise and shine!”'，后面跟着一个换行字符(printf语句第一个参数中的' \n ')。

在大多数情况下，您还需要从定义的函数中获取一些信息。下面是编写返回单个值的函数的语法:

function ret-var = name (arg-list)

body

endfunction

符号ret-var是保存函数返回值的变量名。这个变量必须在函数体结束之前定义，以便函数返回值。

函数体中使用的变量是函数的局部变量。在arg-list和ret-var中命名的变量也是函数的局部变量。有关如何在函数中访问全局变量的信息，请参阅全局变量。

例如，下面是一个计算向量元素平均值的函数:

function retval = avg (v)

retval = sum (v) / length (v);

endfunction

如果我们把avg写成这样，

function retval = avg (v)

if (isvector (v))

retval = sum (v) / length (v);

endif

endfunction

然后用一个矩阵而不是一个向量作为参数调用这个函数，Octave就会打印出这样的错误信息:

error: value on right hand side of assignment is undefined

因为if语句的主体从未执行过，检索也从未定义过。为了防止此类模糊的错误，最好始终确保返回变量始终具有值，并在遇到问题时生成有意义的错误消息。例如，avg可以这样写:

function retval = avg (v)

retval = 0;

if (isvector (v))

retval = sum (v) / length (v);

else

error ("avg: expecting vector argument");

endif

endfunction

这个函数还有一个问题。如果调用它时没有参数呢?如果没有额外的错误检查，Octave可能会打印一条错误消息，这并不能真正帮助您跟踪错误的来源。为了允许您捕获此类错误，Octave为每个函数提供了一个名为nargin的自动变量。每次调用函数时，都会自动将nargin初始化为实际传递给函数的参数数量。例如，我们可以这样重写avg函数:

function retval = avg (v)

retval = 0;

if (nargin != 1)

usage ("avg (vector)");

endif

if (isvector (v))

retval = sum (v) / length (v);

else

error ("avg: expecting vector argument");

endif

endfunction

对于用.m文件代码编写的函数，如果调用时带有比预期更多的参数，Octave会自动报告错误。如果用太少的参数调用函数，Octave不会自动报告错误，因为函数通常可能有默认参数，但任何尝试使用未给定值的变量都会导致错误。函数可以检查调用它们时使用的参数，以避免此类问题，并提供更多特定于上下文的错误消息。

: *n =* **nargin** *()*

: *n =* **nargin** *(fcn)*

报告函数输入参数的数量。

从函数内部调用，返回传递给函数的参数个数。在顶层，返回传递给Octave的命令行参数的数量。

如果使用可选参数fcn(函数名或句柄)调用，则返回声明的函数可以接受的参数数量。

如果fcn的最后一个参数为varargin，则返回值为负值。例如，set的函数union声明为

function [y, ia, ib] = union (a, b, varargin)

and

nargin ("union")

⇒ -3

编程注意:对编译后的函数(.)Oct文件)，如内置或动态加载的函数。

**参见:**nargout, narginchk, varargin, inputname。

: *namestr =* **inputname** *(n)*

: *namestr =* **inputname** *(n, ids\_only)*

将第n个参数的名称返回给调用函数。

如果参数不是一个简单的变量名，则返回一个空字符串。返回""的例子包括数字(5.1)、表达式(y/2)以及单元格或结构索引(c{1}或s.field)。

Inputname仅在函数中有用。在命令行或脚本中使用时，它总是返回一个空字符串。

默认情况下，如果第n个参数不是有效的变量名，则返回空字符串。如果可选参数ids\_only为false，则返回该参数的文本，即使它不是一个有效的变量名。这是一个Octave扩展，它允许程序员准确地查看函数是如何被调用的，即使输入是复杂的表达式。

**参见:**nargin, narginchk。

: *val =* **silent\_functions** *()*

: *old\_val =* **silent\_functions** *(new\_val)*

: *old\_val =* **silent\_functions** *(new\_val, "local")*

查询或设置控制是否抑制函数的内部输出的内部变量。

如果禁用此选项，Octave将显示通过计算函数体中不以分号结束的表达式产生的结果。

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。

**11.3从函数返回**

用户定义函数的函数体可以包含一条返回语句。该语句将控制权返回给Octave程序的其余部分。它是这样的:

return

与C语言中的return语句不同，Octave的return语句不能用于从函数返回值。相反，您必须将值赋给作为函数语句一部分的返回变量列表。return语句只是使从深度嵌套循环或条件语句中退出函数变得更容易。

下面是一个函数的例子，它检查向量中是否有非零元素。

function retval = any\_nonzero (v)

retval = 0;

for i = 1:length (v)

if (v (i) != 0)

retval = 1;

return;

endif

endfor

printf ("no nonzero elements found\n");

endfunction

注意，如果没有添加额外的逻辑来避免在向量确实包含非零元素时打印消息，则无法使用break语句在找到非零值时退出循环来编写此函数。

: return

当Octave在函数或脚本中遇到关键字return时，它会立即将控制权返回给调用者。在顶层，return语句被忽略。在每个函数定义的末尾都假定有一条返回语句。

**11.4多个返回值**

与许多其他计算机语言不同，Octave允许您定义返回多个值的函数。定义返回多个值的函数的语法为

function [ret-list] = name (arg-list)

body

endfunction

其中name、arg-list和body具有与之前相同的含义，ret-list是一个以逗号分隔的变量名列表，用于保存函数返回的值。返回值列表必须至少包含一个元素。如果ret-list只有一个元素，则这种形式的函数语句等价于前一节中描述的形式。

下面是一个函数的示例，它返回两个值，一个是向量的最大元素，另一个是它在向量中第一次出现的索引。

function [max, idx] = vmax (v)

idx = 1;

max = v (idx);

for i = 2:length (v)

if (v (i) > max)

max = v (i);

idx = i;

endif

endfor

endfunction

在这种特殊情况下，这两个值可以作为单个数组的元素返回，但这并不总是可能或方便的。要返回的值可能没有兼容的维度，通常需要为单个返回值指定不同的名称。

可以使用nthargout函数在单元格数组中一次只获取一些返回值或几个返回值。参见单元格数组对象。

: *arg =* **nthargout** *(n, fcn, …)*

: *arg =* **nthargout** *(n, ntot, fcn, …)*

返回函数句柄或字符串fcn指定的函数的第n个输出参数。

任何附加参数都直接传递给fcn。调用fcn的参数总数可以传入not;默认情况下，ntot为n。输入n也可以是输出的索引向量，在这种情况下，输出将是请求的输出参数的单元格数组。

nthargout的预期用途是避免使用中间变量。例如，当找到一个矩阵的最大条目的索引时，下面的两个组合

m = magic (5);

cell2mat (nthargout ([1, 2], @ind2sub, size (m),

nthargout (2, @max, m(:))))

⇒ 5 3

完全等同于以下几行:

m = magic (5);

[~, idx] = max (M(:));

[i, j] = ind2sub (size (m), idx);

[i, j]

⇒ 5 3

将所有输出参数收集到单个单元数组中也很有帮助，如下面的代码所示:

USV = nthargout ([1:3], @svd, hilb (5));

**参见:**nargin, nargout, varargin, varargout, isargout。

除了每次调用函数时设置nargout之外，Octave还自动将nargout初始化为期望返回的值的数量。这允许您根据函数用户请求的值的数量编写行为不同的函数。对内置变量ans的隐式赋值不会计算在输出参数的计数中，因此nargout的值可能为零。

svd和hist函数是内置函数的示例，它们根据nargout的值表现不同。例如，hist在没有输出变量的情况下调用时将绘制一个直方图，但如果调用时有输出，它将返回频率计数和/或bin中心，而不创建一个图。

可以编写只设置某些返回值的函数。例如，调用函数

function [x, y, z] = f ()

x = 1;

z = 2;

endfunction

像

[a, b, c] = f ()

产物

a = 1

b = [](0x0)

c = 2

还有一个警告。

: *n =* **nargout** *()*

: *n =* **nargout** *(fcn)*

报告函数输出参数的数量。

从函数内部调用，返回调用者期望接收的值的数目。在顶层，没有参数的nargout是未定义的，会产生错误。

如果使用可选参数fcn(函数名或句柄)调用，则返回该函数可以产生的已声明输出值的个数。

如果最终输出参数为varargout，则返回值为负。

例如,

f ()

会导致nargout在函数f和内部返回0吗

[s, t] = f ()

将导致nargout在函数f内返回2。

在第二个用法中，

nargout (@histc) # or nargout ("histc") using a string input

将返回2，因为histc有两个输出，而

nargout (@imread)

将返回-2，因为imread有两个输出，第二个是varargout。

编程注意。Nargout不适用于内置函数，并对所有匿名函数返回-1。

参见:nargin, varargout, isargout, nthargout。

**11.5变长返回列表**

使用类似于使用特殊变分参数名的语法，可以从函数返回可变数量的输出参数。为了让函数返回可变数量的输出参数，使用了特殊的输出参数名称varargout。与varargin一样，varargout是一个单元格数组，它将包含请求的输出参数。

例如，下面的函数将第一个输出参数设置为1，第二个参数设置为2，以此类推。

function varargout = one\_to\_n ()

for i = 1:nargout

varargout{i} = i;

endfor

endfunction

当被调用时，这个函数返回如下的值

[a, b, c] = one\_to\_n ()

⇒ a = 1

⇒ b = 2

⇒ c = 3

如果varargin (varargout)没有作为输入(输出)参数列表的最后一个元素出现，那么它就不是特殊的，处理方式与任何其他参数名称相同。

: *[r1, r2, …, rn] =* **deal** *(a)*

: *[r1, r2, …, rn] =* **deal** *(a1, a2, …, an)*

将输入参数复制到相应的输出参数中。

如果只提供了一个输入参数，则将其值复制到每个输出。

例如,

[a, b, c] = deal (x, y, z);

等同于

a = x;

b = y;

c = z;

和

[a, b, c] = deal (x);

等同于

a = b = c = x;

编程注意:deal通常用于从单元数组或结构派生的逗号分隔列表。这是不必要的，因为解释器可以在没有函数调用开销的情况下执行相同的操作。例如:

c = {[1 2], "Three", 4};

[x, y, z] = c{:}

⇒

x =

1 2

y = Three

z = 4

**参见:**cell2struct, struct2cell, repmat。

**11.6变长参数列表**

有时在定义函数时不知道输入参数的数量。举个例子，假设一个函数返回所有输入参数中最小的一个。例如:

a = smallest (1, 2, 3);

b = smallest (1, 2, 3, 4);

在这个例子中，a和b都是1。最小函数的一种写法是

function val = smallest (arg1, arg2, arg3, arg4, arg5)

body

endfunction

然后使用margin的值来确定应该考虑哪个输入参数。这种方法的问题在于它只能处理有限数量的输入参数。

如果特殊参数名称varargin出现在函数参数列表的末尾，则表明该函数接受可变数量的输入参数。使用varargin函数是这样的

function val = smallest (varargin)

body

endfunction

在函数体中，可以通过变量varargin访问输入参数。这个变量是一个包含所有输入参数的单元格数组。有关使用单元格数组的详细信息，请参阅单元格数组。最小的函数现在可以这样定义

function val = smallest (varargin)

val = min ([varargin{:}]);

endfunction

这个实现可以处理任意数量的输入参数，但它也是这个问题的一个非常简单的解决方案。

varargin的一个稍微复杂一点的例子是print\_arguments函数，它打印所有输入参数。这样的函数可以这样定义

function print\_arguments (varargin)

for i = 1:length (varargin)

printf ("Input argument %d: ", i);

disp (varargin{i});

endfor

endfunction

这个函数产生如下的输出

print\_arguments (1, "two", 3);

-| Input argument 1: 1

-| Input argument 2: two

-| Input argument 3: 3

: *[reg, prop] =* **parseparams** *(params)*

: *[reg, var1, …] =* **parseparams** *(params, name1, default1, …)*

在reg中返回param中第一个字符串元素之前的单元格元素，在prop中返回从第一个字符串元素开始的所有剩余元素。

例如:

[reg, prop] = parseparams ({1, 2, "linewidth", 10})

reg =

{

[1,1] = 1

[1,2] = 2

}

prop =

{

[1,1] = linewidth

[1,2] = 10

}

parseparams函数可用于将常规数值参数与作为可变单元数组的属性/值对给出的附加参数分开。

在调用的第二种形式中，直接指定可用选项，并将其默认值作为名称-值对给出。如果参数没有形成名称-值对，或者出现一个与任何可用选项不匹配的选项，则会发生错误。

当从m-file函数调用时，错误会以调用函数的名称作为前缀。

选项的匹配不区分大小写。

**参见:**varargin, inputParser

**11.7忽略参数**

在正式参数列表中，可以使用虚拟占位符~代替名称。这表明相应的参数值应该被忽略，并且不存储到任何变量中。

function val = pick2nd (~, arg2)

val = arg2;

endfunction

使用此声明不会影响margin的值。

也可以使用相同的语法忽略返回参数。例如，sort函数既返回已排序的值，又返回原始输入的索引向量，这将导致已排序的输出。忽略第二个输出很简单—不要请求多个输出。但是忽略第一个输出，只计算第二个输出，需要使用~占位符。

x = [2, 3, 1];

[s, i] = sort (x)

⇒

s =

1 2 3

i =

3 1 2

[~, i] = sort (x)

⇒

i =

3 1 2

当使用~占位符时，必须使用逗号(而不是空白)来分隔输出参数。否则，解释器将把~视为逻辑非操作符。

[~ i] = sort (x)

parse error:

invalid left hand side of assignment

函数可以利用忽略的输出来减少执行的计算次数。为此，使用isargout函数查询是否需要输出参数。例如:

function [out1, out2] = long\_function (x, y, z)

if (isargout (1))

## Long calculation

…

out1 = result;

endif

…

endfunction

: *tf =* **isargout** *(k)*

在函数中，返回一个逻辑值，该值指示是否将参数k赋值给输出时的变量。

如果结果为false，则通过使用波浪(~)特殊输出参数在函数调用期间忽略该参数。函数可以使用isargout来避免对不需要的输出执行不必要的计算。

如果k超出范围1:max (nargout)，则该函数返回false。K也可以是一个数组，在这种情况下，函数逐个元素地工作，并返回一个逻辑数组。在顶层，isargout返回一个错误。

**参见:**nargout, varargout, nthargout。

**11.8默认参数**

由于Octave支持可变数量的输入参数，因此为一些输入参数分配默认值非常有用。当在参数列表中声明输入参数时，可以像这样为参数分配默认值

function name (arg1 = val1, …)

body

endfunction

如果用户没有给arg1赋值，则它的值为val1。

作为一个例子，下面的函数实现了经典的“Hello, World”程序的一个变体。

function hello (who = "World")

printf ("Hello, %s!\n", who);

endfunction

在没有输入参数的情况下调用该函数时，将打印以下内容

hello ();

-| Hello, World!

当它带输入参数被调用时，它打印以下内容

hello ("Beautiful World of Free Software");

-| Hello, Beautiful World of Free Software!

有时显式地告诉Octave使用输入参数的默认值是有用的。这可以在调用函数时写一个':'作为输入参数的值。

hello (:);

-| Hello, World!

**11.9验证参数**

Octave是一种弱类型编程语言。因此，有可能调用带有参数的函数，这可能会导致错误或可能产生不希望看到的副作用。例如，调用具有巨大稀疏矩阵的字符串处理函数。

在函数的开头验证它是否被正确调用是一个很好的做法。为此，Octave提供了几个函数。

**11.9.1验证参数的数量**

在Octave中，下面的习语经常出现在函数定义的开头:

if (nargin < min\_#\_inputs || nargin > max\_#\_inputs)

print\_usage ();

endif

它会停止函数的执行，并在输入数量错误时打印一条关于调用函数的正确方法的消息。

类似的错误检查由narginchk和nargoutchk提供。

: **narginchk** *(minargs, maxargs)*

检查输入参数的正确数目。

如果调用函数中的参数数量超出minargs和maxargs范围，则生成错误消息。否则，什么都不做。

minargs和maxargs都必须是标量数值。0、Inf和负值都是允许的，minargs和maxargs可以相等。

注意，这个函数计算调用者的边际值。

**参见:**nargoutchk, error, nargout, nargin。

: **nargoutchk** *(minargs, maxargs)*

: *msgstr =* **nargoutchk** *(minargs, maxargs, nargs)*

: *msgstr =* **nargoutchk** *(minargs, maxargs, nargs, "string")*

: *msgstruct =* **nargoutchk** *(minargs, maxargs, nargs, "struct")*

检查输出参数的正确数量。

在第一种形式中，如果参数的数量不在minargs和maxargs之间，则返回错误。否则，什么都不做。请注意，此函数对调用者计算nargout的值，因此其值必须未被篡改。

minargs和maxargs都必须是数值标量。0, Inf和-都是有效的，它们可以有相同的值。

为了向后兼容，如果请求的输出数量无效，其他形式将返回适当的错误消息字符串(或结构)。

这对于检查提供给函数的输出参数的数量是否在可接受的范围内非常有用。

参见:narginchk, error, nargout, nargin。

**11.9.2验证实参的类型**

除了参数的数量之外，还可以检查输入的各种属性。Validatestring用于字符串参数，validateattributes用于数值参数。

: *validstr =* **validatestring** *(str, strarray)*

: *validstr =* **validatestring** *(str, strarray, funcname)*

: *validstr =* **validatestring** *(str, strarray, funcname, varname)*

: *validstr =* **validatestring** *(…, position)*

验证str是strarray中的元素或元素的子字符串。

当str是要测试的字符串，而strarray是包含有效值的字符串的单元格数组时，validstr将是str的验证形式，其中验证定义为str是validstr的成员或子字符串。这对于验证和扩展短选项(如“r”)到它们的较长形式(如“red”)都很有用。如果str是validstr的子字符串，并且存在多个匹配项，则如果所有匹配项都是彼此的子字符串，则返回最短的匹配项。否则，将引发错误，因为str的展开是二义性的。所有比较都不区分大小写。

附加的输入funcname、varname和position是可选的，它们将使任何生成的验证错误消息更加具体。

例子:

validatestring ("r", {"red", "green", "blue"})

⇒ "red"

validatestring ("b", {"red", "green", "blue", "black"})

⇒ error: validatestring: multiple unique matches were found for 'b':

blue, black

**参见:**strcmp, strcmpi, validateattributes, inputParser。

:**validateattributes** *(A, classes, attributes)*

:**validateattributes** *(A, classes, attributes, arg\_idx)*

:**validateattributes** *(A, classes, attributes, func\_name)*

: **validateattributes** *(A, classes, attributes, func\_name, arg\_name)*

: **validateattributes** *(A, classes, attributes, func\_name, arg\_name, arg\_idx)*

检查输入参数的有效性。

通过属于一个类并持有所有属性来确认参数A是有效的。如果没有，则抛出错误，并相应地格式化消息。通过函数名fun\_name、参数名arg\_name及其在输入arg\_idx中的位置，可以使错误消息更加完整。

类必须是带有类名的字符串单元格数组(允许为空单元格数组)(记住，类名是区分大小写的)。除了类名之外，以下类别名也是有效的:

"float"

由“双精度”和“单精度”类组成的浮点值。

"integer"

整数值，包含(u)int8， (u)int16， (u)int32， (u)int64类。

"numeric"

由浮点数或整数值组成的数值。

属性必须是包含a检查名称的单元格数组，其中一些需要在名称之后提供附加值(请参阅下面的每个详细信息)。

"<="

所有值都小于或等于属性中的以下值。

"<"

所有值都小于属性中的以下值。

">="

所有值都大于或等于属性中的以下值。

">"

所有值都大于属性中的以下值。

"2d"

一个二维矩阵。注意，向量和空矩阵有两个维度，其中一个维度的长度为1，或者两个维度的长度都为0。

"3d"

不超过3个维度。二维矩阵是三维矩阵，其三维长度为1。

"binary"

所有值要么为1，要么为0。

"column"

值排列在单个列中。

"decreasing"

没有任何值是NaN，并且每个值都小于前一个值。

"diag"

值是一个对角矩阵。

"even"

所有的值都是偶数。

"finite"

所有的值都是有限的。

"increasing"

没有一个值是NaN，每个值都大于前一个值。

"integer"

所有值均为整型。这与使用isinteger不同，后者只检查其是否是整数类型。这将检查A中的每个值是否为整数值，也就是说，它没有小数部分。

"ncols"

具有与属性中的下一个值完全相同的列。

"ndims"

具有与属性中的下一个值完全相同的维度。

"nondecreasing"

任何值都不是NaN，每个值都大于等于前一个值。

"nonempty"

它不是空的。

"nonincreasing"

没有任何值是NaN，并且每个值都小于或等于前一个值。

"nonnan"

没有值是NaN。

"nonnegative"

所有值都是非负的。

"nonsparse"

它不是一个稀疏矩阵。

"nonzero"

没有值是零。

"nrows"

具有与属性中的下一个值完全相同的行数。

"numel"

具有与属性中的下一个值相同数量的元素。

"odd"

所有值都是奇数。

"positive"

所有值都是正的。

"real"

它是一个非复矩阵。

"row"

值排列在单行中。

"scalar"

它是一个标量。

"size"

它的长度等于下一个属性的值。下一个值必须是包含每个维度长度的数组。如果要忽略某个维度的检查，可以使用NaN的值。

"square"

是一个方阵。

"vector"

值排列在单个向量(列或向量)中。

**参见:**isa, validatestring, inputParser。

作为validateattributes的替代方案，有几个更短的方便函数用于检查单个属性。

:**mustBeFinite** *(x)*

要求输入x是有限的。

如果输入x的任何元素不是由isfinite (x)确定的有限元素，则引发错误。

**参见:**必能南，是有限的。

: **mustBeGreaterThan** *(x, c)*

要求输入x大于c。

如果输入x的任何元素不大于c，由x > c确定，则引发错误。

**参见:**mustbegreaterthanoreequal, mustBeLessThan, gt。

:**mustBeGreaterThanOrEqual** *(x, c)*

要求输入x大于等于c。

如果输入x的任何元素不大于或等于c，由x >= c确定，则引发错误。

**参见:**必须大于等于，必须小于等于，等。

: **mustBeInteger** *(x)*

要求输入x为整数值(但不一定是整型)。

如果输入x的任何元素不是由各种检查确定的有限、实数、整数值，则引发错误。

**参见:**mustbennumeric。

: **mustBeLessThan** *(x, c)*

要求输入x小于c。

如果输入x的任何元素不小于c，由x < c确定，则引发错误。

**参见:**必须小于等于，必须大于等于，大于等于。

: **mustBeLessThanOrEqual** *(x, c)*

要求输入小于或等于给定值。

如果输入x的任何元素不小于或等于c，由x <= c确定，则引发错误。

参见:必须小于等于，必须大于等于。

:**mustBeMember** *(x, valid)*

要求输入x是给定有效值集合的成员。

如果输入x的任何元素不是由ismember (x)确定的有效集合的成员，则引发错误。

编程注意:在调用ismember时，由于字符和单元字符串之间的相互作用，字符输入可能表现得很奇怪。但如果你只是自然地使用它，它可能会“做你想做的事”。为了保证操作，使用cellstr将所有字符数组转换为单元格字符串。

**参见:**mustBeNonempty, ismember.

: **mustBeNegative** *(x)*

要求输入x为负。

如果输入x的任何元素不为负，则由x < 0确定，则引发错误。

**参见:**mustBeNonnegative。

: **mustBeNonempty** *(x)*

要求输入x非空。

如果输入x为空，由isempty (x)确定，则引发错误。

**参见:**mustBeMember, mustBeNonzero, isempty。

: **mustBeNonNan** *(x)*

要求输入x是非nan的。

如果输入x的任何元素是NaN(由isnan (x)确定)，则引发错误。

**参见:**必须是有限的，必须是空的，是空的。

: **mustBeNonnegative** *(x)*

要求输入x不为负。

如果输入x的任何元素为负，则引发错误，由x >= 0确定。

**参见:**必须非零，必须积极。

: **mustBeNonpositive** *(x)*

要求输入x不为正。

如果输入x的任何元素为正数，由x <= 0确定，则引发错误。

**参见:**mustnegative, mustBeNonzero。

: **mustBeNonsparse** *(x)*

要求输入x不是稀疏的。

如果输入x是稀疏的，由issparse (x)确定，则引发错误。

**参见:**issparse。

: **mustBeNonzero** *(x)*

要求输入x不为零。

如果输入x的任何元素为零，由x == 0确定，则引发错误。

**参见:**mustBeNonnegative, mustBePositive。

:**mustBeNumeric** *(x)*

要求输入x是数字。

如果输入x不是由isnumeric (x)确定的数字，则引发错误。

**参见:**mustBeNumericOrLogical, isnumeric。

:**mustBeNumericOrLogical** *(x)*

要求输入x是数字或逻辑的。

如果输入x不是数字或逻辑的，由isnumeric (x) || islogical (x)确定，则引发错误。

**参见:**mustbennumeric, isnumeric, islogical。

: **mustBePositive** *(x)*

要求输入x为正。

如果输入x的任何元素不为正，则由x > 0确定，则引发错误。

**参见:**mustBeNonnegative, mustBeNonzero。

: **mustBeReal** *(x)*

要求输入x是实数。

如果输入x不是实数，则引发错误，由isreal (x)确定。

**参见:**必必有限，必不南，以色列。

**11.9.3解析参数 重做重做重做重做重做重做重做重做重做重做重做重做**

如果前面的验证函数都不够用，那么还有一个类inputParser，它可以为函数执行极其复杂的输入检查。

: *p =* **inputParser** *()*

创建inputParser类的对象p。

这个类的设计目的是方便地解析函数参数。类支持四种类型的参数:

1. mandatory (see addRequired);
2. optional (see addOptional);
3. named (see addParameter);
4. switch (see addSwitch).

使用这些方法定义函数 API 后，可以使用 parse 方法分析提供的参数，并使用 Results 访问器访问结果。

:**inputParser.Parameters**

返回已定义的参数名列表。

: **inputParser.Results**

返回结构，参数名作为字段名和相应的值。

: **inputParser.Unmatched**

返回结构类似于Results，但用于不匹配的参数。请参阅KeepUnmatched属性。

: **inputParser.UsingDefaults**

返回带有使用默认值的参数名称的单元格数组。

: **inputParser.CaseSensitive** = boolean

设置参数名称的匹配是否区分大小写。默认为false。

: **inputParser.FunctionName** = name

设置在错误消息中使用的函数名;默认为空字符串。

: **inputParser.KeepUnmatched** = boolean

设置是否对未定义的参数给出错误。默认为false。如果设置为true，则可以在parse方法之后通过Unmatched访问额外的参数。请注意，由于Switch和Parameter参数可以混合使用，因此不可能知道不匹配的类型。如果发现argument不匹配，则假定它属于Parameter类型，并期望后跟一个值。

: **inputParser.StructExpand** = boolean

设置是否可以将结构传递给函数，而不是参数/值对。默认为true。

下面的例子展示了如何使用这个类:

function check (varargin)

p = inputParser (); # create object

p.FunctionName = "check"; # set function name

p.addRequired ("pack", @ischar); # mandatory argument

p.addOptional ("path", pwd(), @ischar); # optional argument

## create a function handle to anonymous functions for validators

val\_mat = @(x) isvector (x) && all (x <= 1) && all (x >= 0);

p.addOptional ("mat", [0 0], val\_mat);

## create two arguments of type "Parameter"

val\_type = @(x) any (strcmp (x, {"linear", "quadratic"}));

p.addParameter ("type", "linear", val\_type);

val\_verb = @(x) any (strcmp (x, {"low", "medium", "high"}));

p.addParameter ("tolerance", "low", val\_verb);

## create a switch type of argument

p.addSwitch ("verbose");

p.parse (varargin{:}); # Run created parser on inputs

## the rest of the function can access inputs by using p.Results.

## for example, get the tolerance input with p.Results.tolerance

endfunction

check ("mech"); # valid, use defaults for other arguments

check (); # error, one argument is mandatory

check (1); # error, since ! ischar

check ("mech", "~/dev"); # valid, use defaults for other arguments

check ("mech", "~/dev", [0 1 0 0], "type", "linear"); # valid

## following is also valid. Note how the Switch argument type can

## be mixed into or before the Parameter argument type (but it

## must still appear after any Optional argument).

check ("mech", "~/dev", [0 1 0 0], "verbose", "tolerance", "high");

## following returns an error since not all optional arguments,

## 'path' and 'mat', were given before the named argument 'type'.

check ("mech", "~/dev", "type", "linear");

注1:一个函数可以混合使用这四种API类型，但它们必须以特定的顺序出现。必需参数必须放在首位，并且可以后跟任何可选参数。只有Parameter和Switch参数可以混合在一起，并且它们必须出现在末尾。

注意2:如果可选参数和参数参数在函数API中混合使用，那么一旦字符串可选参数没有通过验证，它将被视为可选参数的结束。剩下的参数将与任何Parameter或Switch参数进行比较。

**参见:**nargin, validateattributes, validatestring, varargin。

**11.10功能文件**

除了简单的一次性程序外，每次需要时都必须定义所需的所有函数是不实际的。相反，您通常希望将它们保存在一个文件中，以便您可以轻松地编辑它们，并保存它们以供以后使用。

Octave不要求您在使用文件之前从文件中加载函数定义。您只需要将函数定义放在Octave可以找到它们的地方。

当Octave遇到未定义的标识符时，它首先查找已经编译并当前列在其符号表中的变量或函数。如果在那里找不到定义，它就在目录列表(路径)中搜索以.m结尾的文件，这些文件与未定义的标识符具有相同的基名一旦Octave找到一个文件名匹配的文件，就读取该文件的内容。如果它定义了单个函数，则编译并执行该函数。有关如何在单个文件中定义多个函数的更多信息，请参阅脚本文件。

当Octave从函数文件定义一个函数时，它会保存它读取的文件的全名和文件上的时间戳。如果文件上的时间戳发生了变化，Octave可能会重新加载文件。当Octave以交互方式运行时，每次Octave打印提示符时，时间戳检查通常最多发生一次。如果当前工作目录发生变化，也会搜索新的函数定义。

检查时间戳允许您在Octave运行时编辑函数的定义，并自动使用新的函数定义，而无需重新启动Octave会话。

为了避免检查不太可能更改的函数的时间戳而不必要地降低性能，Octave假设Octave -home/share/ Octave /version/m目录树中的函数文件不会更改，因此它不必在每次使用这些文件中定义的函数时检查它们的时间戳。这通常是一个非常好的假设，并且为与Octave一起分发的函数文件提供了显著的性能改进。

如果你知道你自己的函数文件在运行Octave时不会改变，你可以通过调用ignore\_function\_time\_stamp ("all")来提高性能，这样Octave就会忽略所有函数文件的时间戳。将“system”传递给此函数将重置默认行为。

: **edit** *name*

: **edit** *field value*

: *value =* **edit** *("get", field)*

: *value =* **edit** *("get", "all")*

编辑指定函数，或更改编辑器设置。

如果以文件或函数的名称作为参数调用edit，它将在editor定义的文本编辑器中打开。

* 如果函数名在路径上的文件中可用，那么它将在编辑器中打开。如果没有找到文件，则使用m-file变体，以“”结尾。，将予以考虑。如果仍然没有找到文件，那么以“@”开头的变体，然后以“@”开头和以“”结尾。M”将被考虑。
* 如果name是命令行函数的名称，那么将创建一个m文件来包含该函数及其当前定义。
* 如果名字。如果指定了Cc，那么它将搜索name。Cc中的路径，并在编辑器中打开它。如果没有找到该文件，那么将创建一个新的.cc文件。如果name恰好是一个m文件或命令行函数，那么该函数的文本将作为注释插入到.cc文件中。
* 如果名字。Ext在您的路径上，那么它将被编辑，否则编辑器将以名称开始。Ext在当前目录中作为文件名。

警告:在新定义可用之前，您可能需要清除name。如果您正在编辑.cc文件，则需要执行mkoctfile name。Cc之前的定义将可用。

如果使用字段和值变量调用edit，则控制字段字段的值将设置为value。

如果请求输出参数并且获得第一个输入参数，则edit将返回控制字段字段的值。如果控件字段不存在，edit将返回一个包含所有字段和值的结构。因此，edit ("get"， "all")返回一个完整的控制结构。

使用了以下控制字段:

‘author’

这个名称放在新函数的“## Author:”字段之后。默认情况下，它从密码数据库的gecos字段进行猜测。

‘email’

这是要在作者字段的名称之后列出的电子邮件地址。默认情况下，它会猜测<$LOGNAME@$HOSTNAME>，如果$HOSTNAME没有定义，它会使用uname -n。你可能想要重写这个。一定要使用user@host格式。

‘license’

‘gpl’

GNU通用公共许可证(默认)。

‘bsd’

无广告条款的bsd风格许可证。

‘pd’

公共领域

‘"text"’

您自己的默认版权和许可。

除非您指定“pd”，否则edit将在版权声明前加上“copyright (C) YYYY Author”。

‘mode’

该值决定编辑器应该以异步模式启动(编辑器在后台启动，Octave继续)还是同步模式启动(Octave等待直到编辑器退出)。将其设置为“sync”以同步模式启动编辑器。默认值是“async”(参见system)。

‘editinplace’

确定文件是否应该就地编辑，而不考虑它们是否可修改。默认为true。将其设置为false以使只读函数文件在编辑时自动复制到' home '(如果存在)。

‘home’

该值指示在编辑器中打开系统文件之前应该将其复制到的目录。这样做的目的是，这个目录也在路径中，这样系统功能文件的编辑副本就会遮蔽原来的目录。此设置仅在' editinplace '设置为false时有效。默认值是空矩阵([])，这意味着不使用它。以前版本的Octave的默认值是~/ Octave。

**参见:**编辑器、路径。

: **mfilename** *()*

:**mfilename** *("fullpath")*

: **mfilename** *("fullpathext")*

返回当前正在执行的文件的名称。

返回当前正在执行的脚本或函数的基本名称，不带任何扩展名。如果从m文件外部(如命令行)调用，则返回空字符串。

给定参数"fullpath"，包括文件名的目录部分，但不包括扩展名。

给定参数" fullpatheext "，包括文件名的目录部分和扩展名。

**参见:**inputname, dbstack。

: *val =* **ignore\_function\_time\_stamp** *()*

: *old\_val =* **ignore\_function\_time\_stamp** *(new\_val)*

查询或设置内部变量，该变量控制Octave在每次查找函数文件中定义的函数时是否检查文件上的时间戳。

如果内部变量设置为"system"， Octave不会自动重新编译Octave -home/share/version/m子目录下的函数文件，如果它们自上次编译以来发生了变化，但如果它们在搜索路径中发生了变化，则会重新编译其他函数文件。

如果设置为"all"， Octave将不会重新编译任何函数文件，除非它们的定义被clear删除。

如果设置为"none"， Octave将始终检查文件上的时间戳，以确定函数文件中定义的函数是否需要重新编译。

**11.10.1操作加载路径**

当调用函数时，Octave在目录列表中搜索包含函数声明的文件。这个目录列表称为加载路径。默认情况下，加载路径包含由Octave分发的目录列表以及当前工作目录。要查看当前的加载路径，请调用path函数，不带任何输入或输出参数。

可以使用addpath和rmpath在加载路径中添加或删除目录。例如，下面的代码将' ~/Octave '添加到加载路径中。

addpath ("~/Octave")

在此之后，目录' ~/Octave '将被搜索函数。

: **addpath** *(dir1, …)*

:**addpath** *(dir1, …, option)*

: *oldpath =* **addpath** *(…)*

将命名目录添加到函数搜索路径。

如果option为"-begin"或0(默认值)，则在当前路径前加上目录名。如果option为"-end"或1，则将目录名附加到当前路径后。添加到路径中的目录必须存在。

除了接受单个目录参数外，还接受以pathsep分隔的目录名列表。例如:

addpath ("dir1:/dir2:~/dir3")

新添加的路径出现在加载路径中的顺序与它们出现在addpath参数中的顺序相同。当将加载路径扩展到前面时，首先添加参数列表中的最后一个路径。当将加载路径扩展到末尾时，首先添加参数列表中的第一个路径。

对于添加的每个目录，如果它不在路径中，addpath检查是否存在一个名为PKG\_ADD的文件(注意没有.m扩展名)，如果它存在就运行它。

**参见:**path, rmpath, genpath, pathdef, savepath, pathsep。

: *pathstr =* **genpath** *(dir)*

: *pathstr =* **genpath** *(dir, skipdir1, …)*

返回由dir及其所有子目录构造的路径。

该路径不包括包目录(以' + '开头)、旧式类目录(以' @ '开头)、私有目录或这些类型的任何子目录。

如果给出了额外的字符串参数，结果路径将排除具有这些名称的目录。

**参见:**path, addpath。

: rmpath (dir1, …)

: oldpath = rmpath (dir1, …)

从当前函数搜索路径中删除dir1，…

除了接受单个目录参数外，还接受以pathsep分隔的目录名列表。例如:

rmpath(“dir1: / dir2: ~ / dir3”)

对于删除的每个目录，rmpath检查是否存在一个名为PKG\_DEL的文件(注意没有.m扩展名)，如果存在则运行它。

参见:path, addpath, genpath, pathdef, savepath, pathsep。

: path ()

: str = path ()

: str = path (path1, …)

修改或显示Octave的加载路径。

如果margin和nargout为零，则以易于阅读的格式显示Octave加载路径的元素。

如果nargout为零且nargout大于零，则返回当前加载路径。

如果margin大于零，将参数连接起来，用pathsep分隔它们。设置结果的内部搜索路径并返回它。

不检查重复的元素。

参见:addpath, rmpath, genpath, pathdef, savepath, pathsep。

: val = pathdef ()

返回Octave的默认路径。

路径信息从四个来源之一提取。可能的来源，按优先顺序排列如下:

1. .octaverc
2. ~/.octaverc
3. <OCTAVE\_HOME>/…/<version>/m/startup/octaverc
4. Octave’s path prior to changes by any octaverc file.

参见:path, addpath, rmpath, genpath, savepath。

: val = pathsep ()

查询路径中用于分隔目录的字符。

参见:filesep。

: rehash ()

重新初始化Octave的加载路径目录缓存。

: fname = file\_in\_loadpath (file)

: fname = file\_in\_loadpath (file, "all")

如果可以在path指定的目录列表中找到该文件，则返回该文件的绝对名称。

如果没有找到文件，则返回一个空字符串。

当file已经是一个绝对名称时，将根据文件系统而不是Octave的loadpath检查该名称。在这种情况下，如果file存在，它将以fname返回，否则返回一个空字符串。

如果第一个参数是字符串的单元格数组，则在loadpath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。

如果提供了第二个可选参数"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\_path, dir\_in\_loadpath, path。

: pathstr = restoredefaultpath ()

将Octave的路径恢复到启动时的初始状态。

重新初始化的路径作为输出返回。

参见:path, addpath, rmpath, genpath, pathdef, savepath, pathsep。

: pathstr = command\_line\_path ()

返回解释器启动时命令行中给Octave的path参数(——path arg)。

参见:path, addpath, rmpath, genpath, pathdef, savepath, pathsep。

: dirname = dir\_in\_loadpath (dir)

: dirname = dir\_in\_loadpath (dir, "all")

如果在path指定的目录列表中可以找到与dir匹配的loadpath元素的绝对名称，则返回该元素的绝对名称。

如果没有找到匹配项，则返回一个空字符串。

匹配在每个路径元素的末尾执行。例如，如果dir是"foo/bar"，它匹配路径元素"/some/dir/foo/bar"，但不匹配"/some/dir/foo/bar/baz" "/some/dir/allfoo/bar"。当dir是一个绝对名称，而不仅仅是一个路径片段时，它将根据文件系统而不是Octave的loadpath进行匹配。在这种情况下，如果dir存在，它将以dirname的形式返回，否则返回一个空字符串。

如果提供了可选的第二个参数，则返回一个包含所有名称匹配的单元格数组，而不仅仅是第一个。

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

: current\_encoding = dir\_encoding (dir)

: dir\_encoding (dir, new\_encoding)

: dir\_encoding (dir, "delete")

: old\_encoding = dir\_encoding (dir, new\_encoding)

查询或设置dir中读取m文件时使用的编码。

每个目录编码覆盖(全局设置的)m文件编码。

字符串DIR必须与目录在加载路径中的显示方式相匹配。

new\_encoding输入必须是有效的编码标识符或“delete”。在后一种情况下，将删除任何逐目录编码，并将对给定的dir使用(全局设置的)m-file编码。

只有在请求输出参数时才返回当前或以前使用的编码。

当向加载路径添加新路径时(例如使用addpath)，将自动从.oct-config文件读取目录编码。要为同一文件夹中的所有文件设置编码，该文件必须包含以“encoding=”开头的行，后面跟着编码标识符。

例如，要将同一文件夹下所有文件的编码设置为ISO 8859-1 (Latin-1)，则创建一个文件。oct-config，其内容如下:

encoding=iso8859-1

如果在文件已经被解析之后更改了文件编码，则必须再次解析文件以使更改生效。这可以用clear all命令来完成。

参见:addpath, path。

**11.10.2子功能**

函数文件可能包含称为子函数的次要函数。这些辅助函数仅对同一函数文件中的其他函数可见。例如，文件f.m包含

function f ()

printf ("in f, calling g\n");

g ()

endfunction

function g ()

printf ("in g, calling h\n");

h ()

endfunction

function h ()

printf ("in h\n")

endfunction

定义一个主函数f和两个子函数。子函数g和h只能从主函数f或从其他子函数调用，但不能从文件外部调用。

: subfcn\_list = localfunctions ()

返回当前文件中所有局部函数(即子函数)的列表。

返回值是一个列单元数组，其中包含从调用localfunctions的函数可访问的所有本地函数的函数句柄。嵌套函数不包括在列表中。

如果调用来自命令行、匿名函数或脚本，则返回值为空单元格数组。

参见:函数。

**11.10.3私有功能**

在许多情况下，一个函数需要访问一个或多个辅助函数。如果helper函数被限制在单个函数的范围内，那么可以使用上面讨论的子函数。但是，如果一个辅助函数被多个函数使用，那么这就不再可能了。在这种情况下，辅助函数可能被放置在需要访问该辅助函数的函数所在目录的子目录中，名为“private”。

举一个简单的例子，考虑一个函数func1，它调用辅助函数func2来完成大部分工作。例如:

function y = func1 (x)

y = func2 (x);

endfunction

然后如果到func1的路径是<directory>/func1。M，如果在目录<directory>/private/func2中找到func2。M，那么func2只能用于在<directory>中找到的函数，如func1。

**11.10.4嵌套函数**

嵌套函数类似于子函数，因为只有主函数在文件外部可见。但是，它们也允许子函数访问父函数中的局部变量。这种共享访问模拟了使用全局变量来共享信息——但是全局变量对Octave的其余部分不可见。作为一种编程策略，以这种方式共享数据可能会创建难以维护的代码。建议尽可能使用子函数来代替嵌套函数。

举一个简单的例子，考虑一个父函数foo，它调用嵌套的子函数bar，并使用一个共享变量x。

function y = foo ()

x = 10;

bar ();

y = x;

function bar ()

x = 20;

endfunction

endfunction

foo ()

⇒ 20

注意，没有用于共享x的特殊语法。这可能会导致父函数与其子函数之间意外共享变量的问题。虽然变量通常是继承的，但子函数的参数和返回值对子函数来说是局部的。

现在考虑使用变量x和y的函数foobar, foobar调用一个嵌套函数foo，该函数以x作为参数并返回y。foo然后调用bat进行一些计算。

function z = foobar ()

x = 0;

y = 0;

z = foo (5);

z += x + y;

function y = foo (x)

y = x + bat ();

function z = bat ()

z = x;

endfunction

endfunction

endfunction

foobar ()

⇒ 10

重要的是要注意，foobar中的x和y保持为零，因为在foo中它们分别是返回值和参数。bat中的x是指foo中的x。

变量继承会导致eval和脚本出现问题。如果在父函数中创建了一个新变量，那么在嵌套的子函数中应该发生什么就不清楚了。例如，考虑一个父函数foo和嵌套的子函数bar:

function y = foo (to\_eval)

bar ();

eval (to\_eval);

function bar ()

eval ("x = 100;");

eval ("y = x;");

endfunction

endfunction

foo ("x = 5;")

⇒ error: can not add variable "x" to a static workspace

foo ("y = 10;")

⇒ 10

foo ("")

⇒ 100

父函数foo无法创建一个新的变量x，但是子函数bar成功了。此外，即使在eval语句中，bar中的y与其父函数foo中的y也是相同的。最好避免将eval与嵌套函数结合使用。

与子函数一样，只有文件中的第一个嵌套函数可以从外部调用。在函数内部，规则更为复杂。一般来说，嵌套函数可以调用:

1. Globally visible functions
2. Any function that the nested function’s parent can call
3. Sibling functions (functions that have the same parents)
4. Direct children

作为一个复杂的例子，考虑一个父函数ex\_top和两个子函数ex\_a和ex\_b。此外，ex\_a还有两个子函数ex\_aa和ex\_ab。例如:

function ex\_top ()

## Can call: ex\_top, ex\_a, and ex\_b

## Can NOT call: ex\_aa and ex\_ab

function ex\_a ()

## Can call everything

function ex\_aa ()

## Can call everything

endfunction

function ex\_ab ()

## Can call everything

endfunction

endfunction

function ex\_b ()

## Can call: ex\_top, ex\_a, and ex\_b

## Can NOT call: ex\_aa and ex\_ab

endfunction

endfunction

**11.10.5重载和自动加载**

函数可以重载以使用不同的输入参数。例如，操作符' + '在Octave中被重载，可以处理single, double, uint8, int32和许多其他参数。重载函数的首选方法是通过类和面向对象编程(参见函数重载)。但是，有时需要撤销用户重载并调用与特定类型关联的默认函数。内置函数就是为此目的而存在的。

: […] = builtin (f, …)

调用基函数f，即使f被重载为给定类型签名的另一个函数。

在进行面向对象编程时，这通常是有用的，并且需要调用Octave的一个基函数，而不是一个新类的重载函数。

一个简单的例子，将sin函数重新定义为cos函数，展示了内置函数是如何工作的。

sin (0)

⇒ 0

function y = sin (x), y = cos (x); endfunction

sin (0)

⇒ 1

builtin ("sin", 0)

⇒ 0

一个动态链接的文件可以定义几个函数。然而，当Octave根据函数文件名搜索函数时，Octave需要一种方式来查找动态链接文件中的每个函数。在支持符号链接的操作系统上，可以为它包含的每个函数创建一个指向原始文件的符号链接。

然而，至少有一种众所周知的操作系统不支持符号链接。为每个函数复制原始文件是不可取的，因为这会增加Octave使用的磁盘空间量。相反，Octave提供了自动加载函数，该函数允许用户定义在哪个文件中找到某个函数。

: autoload\_map = autoload ()

: autoload (function, file)

: autoload (…, "remove")

定义要从文件自动加载的函数。

第二个参数file应该是一个绝对文件名，或者是与运行autolload命令的函数或脚本位于同一目录下的文件名。文件不应该依赖于Octave加载路径。

通常，对自动加载的调用出现在PKG\_ADD脚本文件中，当将目录添加到Octave的加载路径时，对该脚本文件进行评估。为了避免在file中硬编码目录名，如果file与PKG\_ADD脚本在同一目录中，则

autoload ("foo", "bar.oct");

将从bar.oct文件中加载函数foo。以上用法当杠。Oct不在同一个目录中，或者使用

autoload ("foo", file\_in\_loadpath ("bar.oct"))

强烈反对，因为他们的行为可能是不可预测的。

不带参数时，返回一个包含当前自动加载映射的结构。

如果给出了第三个参数“remove”，则该函数将被清除，并且在当前Octave会话期间不再加载。

参见:PKG\_ADD。

**11.10.6功能锁定**

有时需要使用mlock函数将一个函数锁定到内存中。这通常用于包含一些初始化的oct文件或mex文件中的动态链接函数，并且希望调用clear不会删除此初始化。

举个例子，

function my\_function ()

mlock ();

…

endfunction

防止my\_function被调用后从内存中删除，即使调用了clear。可以使用错误锁定来确定函数是否锁定在内存中，也可以使用munlock来解锁函数，下面的代码演示了这一点。

my\_function ();

mislocked ("my\_function")

⇒ ans = 1

munlock ("my\_function");

mislocked ("my\_function")

⇒ ans = 0

mlock的一个常见用途是防止从内存中删除持久变量，如下例所示:

function count\_calls ()

mlock ();

persistent calls = 0;

printf ("count\_calls() has been called %d times\n", ++calls);

endfunction

count\_calls ();

-| count\_calls() has been called 1 times

clear count\_calls

count\_calls ();

-| count\_calls() has been called 2 times

mlock也可以用来防止对m文件的更改，例如在外部编辑器中，对当前Octave会话产生任何影响;ignore\_function\_time\_stamp函数也可以达到类似的效果。

: mlock ()

将当前函数锁定到内存中，使其不能被clear删除。

参见:无锁，锁错，持久，清晰。

: munlock ()

: munlock (fcn)

解锁命名函数fcn，以便可以使用clear从内存中删除它。

如果没有命名函数，则解锁当前函数。

参见:mlock, mislocked, persistent, clear。

: tf = mislocked ()

: tf = mislocked (fcn)

如果命名函数fcn被锁定在内存中，则返回true。

如果没有命名函数，则在当前函数被锁定时返回true。

参见:mlock, munlock, persistent, clear。

**11.10.7功能优先级**

考虑到Octave可以用许多不同的方式定义一个函数，在一个特定的作用域中定义一个函数的多个版本是可能的，甚至是可能的。在特定范围内使用哪个函数的优先级由

1. 在给定范围内具有所需函数名的子函数。
2. 私有函数在包含当前函数的目录的私有目录中定义的函数。
3. 类构造函数构造用户类的函数，如面向对象编程章节所定义。
4. 类方法类的重载函数，如面向对象编程章节中所述。
5. 命令行函数在命令行上定义的函数。
6. 自动加载函数一个被标记为自动加载的函数，参见自动加载。
7. A路径上的函数在用户加载路径上可以找到的函数。此函数也可以有Oct-file、mex-file或m-file版本，这些版本之间的优先级是按此顺序排列的。
8. 内置函数是核心Octave的一部分，如numel, size等。

**11.11脚本文件**

脚本文件是包含(几乎)任何Octave命令序列的文件。它被读取和计算，就像您在Octave提示符下键入每个命令一样，并提供了一种方便的方法来执行逻辑上不属于函数的命令序列。

与函数文件不同，脚本文件不能以关键字function开头。如果是这样，Octave将假设它是一个函数文件，并且它定义了一个函数，应该在定义后立即求值。

脚本文件与函数文件的另一个不同之处在于，脚本文件中命名的变量不是局部变量，而是与命令行中可见的其他变量处于相同的作用域。

即使脚本文件可能不以function关键字开头，也可以在单个脚本文件中定义多个函数并一次加载(但不执行)所有函数。要做到这一点，文件中的第一个标记(忽略注释和其他空白)必须是函数以外的东西。如果你没有其他语句需要求值，你可以使用一个没有效果的语句，像这样:

# Prevent Octave from thinking that this

# is a function file:

1;

# Define function one:

function one ()

…

要让Octave读取这些函数并将其编译为内部表单，您需要确保该文件位于Octave的加载路径中(可通过path函数访问)，然后只需键入包含这些命令的文件的基本名称。(Octave使用与搜索函数文件相同的规则来搜索脚本文件。)

如果文件中的第一个标记(忽略注释)是function, Octave将编译该函数并尝试执行它，并打印一条消息，警告在函数定义之后出现的任何非空白字符。

注意，Octave不会尝试查找任何标识符的定义，直到需要对其求值。这意味着Octave将编译以下语句，如果它们出现在脚本文件中，或者在命令行中输入，

# not a function file:

1;

function foo ()

do\_something ();

endfunction

function do\_something ()

do\_something\_else ();

endfunction

即使函数do\_something在函数foo中引用之前没有定义。这不是一个错误，因为Octave不需要解析函数引用的所有符号，直到函数实际求值。

由于Octave在需要定义之前不会查找定义，因此以下代码将始终打印' bar = 3 '，无论它是直接在命令行上键入的，还是从脚本文件中读取的，或者是函数体的一部分，即使存在一个名为bar的函数或脚本文件。我在八度的路径上。

eval ("bar = 3");

bar

如果在编译函数时解析了定义，那么这样出现在函数体中的代码可能会欺骗Octave。让Octave足够聪明地以一致的方式评估这些代码实际上是不可能的。解析器必须能够在编译时执行对eval的调用，这是不可能的，除非要求值的字符串中的所有引用也可以被解析，并且要求这将是过于严格的(字符串可能来自用户输入，或者依赖于函数求值之前不知道的东西)。

虽然Octave通常从脚本文件中执行命令。M，您可以使用函数源从任何文件执行命令。

: source (file)

: source (file, context)

解析并执行文件的内容。

如果不指定上下文，这相当于从脚本文件执行命令，但不要求将文件命名为file。M或在执行路径上。

脚本可以在调用当前函数(“调用者”)的函数的上下文中执行，也可以在顶级上下文中执行(“基”)，而不是在当前上下文中执行。

参见:run。

**11.11.1发布Octave脚本文件**

publish函数提供了动态记录脚本文件的可能性。与静态文档不同，publish运行脚本文件，在运行脚本时保存任何图形和输出，并以所需的输出格式将它们与静态文档一起呈现。静态文档可以使用Publishing Markup来增强和定制输出。

: publish (file)

: publish (file, output\_format)

: publish (file, option1, value1, …)

: publish (file, options)

: output\_file = publish (file, …)

从Octave脚本文件文件中以几种输出格式之一生成报告。

生成的报告在部分注释中解释Publishing Markup，这在GNU Octave手册中有详细解释。部分注释是以带有双注释字符的行开头的注释块。

假设下面的例子，使用一些发布标记，作为脚本文件pub\_example.m的内容:

## Headline title

#

# Some \*bold\*, \_italic\_, or |monospaced| Text with

# a <https://www.octave.org link to \*GNU Octave\*>.

##

# "Real" Octave commands to be evaluated

sombrero ()

%% MATLAB comment style ('%') is supported as well

%

% \* Bulleted list item 1

% \* Bulleted list item 2

%

% # Numbered list item 1

% # Numbered list item 2

要发布这个脚本文件，输入publish ("pub\_example.m")。

当使用一个输入参数调用时，将在相对于当前工作目录的子目录HTML中生成HTML报告。pub\_example中的任何Octave命令。M在单独的上下文中进行评估，并且在执行脚本文件时创建的任何图形都包含在报告中。

使用publish (file, output\_format)相当于使用结构体调用函数

options.format = output\_format;

publish (file, options)

如下所述。使用选项/值对也是如此

options.option1 = value1;

publish (file, options)

结构选项可以有以下字段名。如果未指定字段名，则使用默认值:

* ' format ' -发布的脚本文件的输出格式，其中之一

“html”(默认),“医生”、“乳胶”、“ppt”、“pdf”,或“xml”。

输出格式“doc”、“ppt”和“xml”目前不支持。要生成“doc”报告，使用您的办公套件打开生成的“html”报告。

在Octave中，通过在名为' \_\_publish\_<custom format>\_output\_\_.m '的函数文件中实现所有回调子函数来支持自定义格式。获取HTML格式类型的模板:

edit (fullfile (fileparts (which ("publish")), ...

"private", "\_\_publish\_html\_output\_\_.m"))

* ' outputDir ' -生成报告所在目录的完整路径。如果没有指定目录，报告将在相对于当前工作目录的子目录html中生成。
* ' stylesheet ' -不支持，仅用于MATLAB兼容。
* ' createThumbnail ' -不支持，仅用于MATLAB兼容性。
* ' figureSnapMethod ' -不支持，仅用于MATLAB兼容性。
* ' imageFormat ' -在评估代码时生成的任何图像的所需格式。

允许的图像格式取决于输出格式:

* ' html '， ' xml ' - ' png '(默认)，Octave支持的任何图像格式
* ' latex ' - ' epsc2 '(默认)，Octave支持的任何图像格式
* ' pdf ' - ' jpg '(默认)或' bmp '，注意MATLAB使用' bmp '作为默认值
* “医生”或“ppt”——“png”(默认),jpg, bmp,或“tiff”
* ' maxWidth '和' maxHeight ' -生成图像的最大宽度(高度)(以像素为单位)。空值表示没有限制。为了使该选项正常工作，必须设置这两个值。

'[] '(默认)，整数值≥0

* ' useNewFigure ' -为计算代码创建的图形使用一个新的图形窗口。这避免了已经打开的图形窗口的副作用。

' true '(默认)或' false '

* ' evalCode ' -评估Octave源文件的代码

' true '(默认)或' false '

* ' catchError ' -在计算代码时捕获错误并继续

' true '(默认)或' false '

* ' codeToEvaluate ' -在发布脚本文件之前应该评估的命令。这些Octave命令不会出现在生成的报告中。
* ' maxOutputLines ' -包含在输出中的代码计算的最大输出行数。

' Inf '(默认)或整数值> 0

* ' showCode ' -在生成的报告中显示评估的Octave命令

' true '(默认)或' false '

选项输出output\_file是一个字符串，包含生成报告的路径和文件名。

**参见:**grabcode。

与publish对应的是grabcode:

: grabcode filename

: grabcode url

: code\_str = grabcode (…)

从publish函数创建的报告中获取代码。

在发布的报告中抓取的代码必须由字符串' ##### SOURCE BEGIN ##### '和' ##### SOURCE END ##### '包围。publish函数自动创建这种格式。

如果未请求返回值，则将代码保存到临时文件中，并在默认编辑器中打开。注意:临时文件必须保存为新的文件名，否则代码将丢失。

如果请求输出，抓取的代码将作为字符串code\_str返回。

例子:

publish ("my\_script.m");

grabcode ("html/my\_script.html");

上面的例子发布了my\_script。到默认位置html/my\_script.html。接下来，抓取已发布的Octave脚本，在一个新的临时文件中编辑其内容。

**参见:** publish。

**11.11.2 出版的标记**

**11.11.2.1如何发布脚本中的标记**

要使用发布标记，首先在新行开头键入' ## '或' %% '。对于MATLAB兼容性，' %%% '的处理方式与' %% '相同。

“##”或“%%”之后的行以“#”或“%”开头，后面至少有一个空格。这些行被解释为部分。节结束于不以“#”或“%”开头的第一行，或者在到达文档末尾时结束。

从文档的第一行开始的节，后面跟着另一个可能为空的节的开头，被解释为文档标题和介绍文本。

请看下面的例子:

%% Headline title

%

% Some \*bold\*, \_italic\_, or |monospaced| Text with

% a <https://www.octave.org link to GNU Octave>.

%%

# "Real" Octave commands to be evaluated

sombrero ()

## Octave comment style supported as well

#

# \* Bulleted list item 1

# \* Bulleted list item 2

#

# # Numbered list item 1

# # Numbered list item 2

**11.11.2.2文本格式化**

章节内部支持基本的文本格式，参见下面给出的示例:

##

# \*bold\*, \_italic\_, or |monospaced| Text

另外，支持两个商标符号，只需包含字母“TM”或“R”。

##

# (TM) or (R)

**11.11.2.3部分**

通过在新行开头键入“##”或“%%”来开始一个部分。段落标题可以写在“##”或“%%”之后的第一行，中间用空格分隔。如果没有节标题，则该节将被解释为前一节的延续。对于MATLAB兼容性，' %%% '的处理方式与' %% '相同。

some\_code ();

## Section 1

#

## Section 2

some\_code ();

##

# Still in section 2

some\_code ();

%%% Section 3

%

%

**11.11.2.4预格式化代码**

要在节中编写预格式化的代码，请在每行开头的“#”之后缩进三个空格，并将代码的上下行留空，但这些行开头的“#”除外。

##

# This is a syntax highlighted for-loop:

#

# for i = 1:5

# disp (i);

# endfor

#

# And more usual text.

**11.11.2.5预格式化文本**

要在节中编写预格式化的文本，请在每行开头的“#”之后缩进两个空格，并将预格式化文本的上下行留空，但这些行开头的“#”除外。

##

# This following text is preformatted:

#

# "To be, or not to be: that is the question:

# Whether 'tis nobler in the mind to suffer

# The slings and arrows of outrageous fortune,

# Or to take arms against a sea of troubles,

# And by opposing end them? To die: to sleep;"

#

# --"Hamlet" by W. Shakespeare

**11.11.2.6项目符号列表**

若要创建项目符号列表，请键入

##

#

# \* Bulleted list item 1

# \* Bulleted list item 2

#

得到如下输出

* Bulleted list item 1
* Bulleted list item 2

注意空格行，除了项目符号列表前后的“#”或“%”!

**11.11.2.7编号列表**

若要创建编号列表，请键入

# #

#

编号列表项1

# #编号列表项2

#

得到如下输出

* Numbered list item 1
* Numbered list item 2

请注意除了编号列表前后的“#”或“%”之外的空行!

**11.11.2.8包含文件内容**

包含外部文件的内容，例如，一个名为“my\_function”的文件。在与发布的Octave脚本相同的位置，使用以下语法将其包含在Octave语法高亮显示中。

或者，您可以将完整路径或相对路径写入文件。

##

#

# <include>my\_function.m</include>

#

# <include>/full/path/to/my\_function.m</include>

#

# <include>../relative/path/to/my\_function.m</include>

#

**11.11.2.9包含图形**

要包含外部图形，例如，在与发布的Octave脚本相同的位置包含一个名为' my\_graphic.png '的图形，请使用以下语法。

或者，您可以将完整路径写入图形。

##

#

# <<my\_graphic.png>>

#

# <</full/path/to/my\_graphic.png>>

#

# <<../relative/path/to/my\_graphic.png>>

#

**11.11.2.10包含url**

基本上，URL写在开始的“<”和结束的“>”之间。

##

# <https://www.octave.org>

在这些角度内并且与URL之间至少间隔一个空格的文本是用于链接的显示文本。

##

# <https://www.octave.org GNU Octave>

一个以' <octave: '开头的链接，后面跟着一个GNU octave函数的名字，可选地加上一个显示的文本，结果是一个到在线GNU octave文档函数索引的链接。

##

# <octave:DISP The display function>

**11.11.2.11数学方程**

可以直接在部分中插入LaTeX内联数学，用单个' $ '符号包围，或者用双' $$ '符号包围显示数学。

##

# Some shorter inline equation $e^{ix} = \cos x + i\sin x$.

#

# Or more complicated formulas as displayed math:

# $$e^x = \lim\_{n\rightarrow\infty}\left(1+\dfrac{x}{n}\right)^{n}.$$

**11.11.2.12 HTML标记**

如果发布的输出是HTML报告，则可以插入仅在这种输出中可见的HTML标记。

##

# <html>

# <table style="border:1px solid black;">

# <tr><td>1</td><td>2</td></tr>

# <tr><td>3</td><td>3</td></tr>

# </html>

**11.11.2.13 LaTeX标记**

如果发布的输出是LaTeX或PDF报告，您可以插入LaTeX标记，它只在这种输出中可见。

##

# <latex>

# Some output only visible in LaTeX or PDF reports.

# \begin{equation}

# e^x = \lim\limits\_{n\rightarrow\infty}\left(1+\dfrac{x}{n}\right)^{n}

# \end{equation}

# </latex>

**11.11.3木星笔记本**

Jupyter笔记本是一种流行的技术，用于以综合的方式同时显示代码、文本和图形输出。Octave可以使用jupyter\_notebook函数将结果发布到Jupyter笔记本。

: notebook = jupyter\_notebook (notebook\_filename)

: notebook = jupyter\_notebook (notebook\_filename, options)

在GNU Octave中运行并填充文件notebook\_filename中的Jupyter Notebook。

支持文本和图形Octave输出。

这个类有一个公共属性notebook，它是一个表示json解码的Jupyter notebook的结构。此属性是故意公开的，以启用高级笔记本操作。

注意:不支持低于4.0的Jupyter Notebook版本(nbformat)。

第二个可选参数options是一个带字段的结构体:

* tmpdir设置临时工作目录。

以下设置支持%plot magic:

* %plot -f <format>或%plot -format <format>:指定图像存储格式。支持的格式有:

PNG(默认)

SVG(注意:如果SVG图像没有出现在笔记本中，则很可能与Jupyter notebook安全机制有关，并且需要显式地“信任”它们)。

JPG

* "%plot -r <number>"或"%plot -resolution <number>":图像分辨率。
* %plot -w <number>或%plot -width <number>:指定图像宽度。
* "%plot -h <number>"或"%plot -height <number>":指定图像高度。

示例：

## Run all cells and generate the filled notebook

## Instantiate an object from a notebook file

notebook = jupyter\_notebook ("myNotebook.ipynb");

## Run the code and embed the results in the notebook property

notebook.run\_all ();

## Generate a new notebook by overwriting the original notebook

notebook.generate\_notebook ("myNotebook.ipynb");

## Run just the second cell and generate the filled notebook

## Instantiate an object from a notebook file

notebook = jupyter\_notebook ("myNotebook.ipynb");

## Run the code and embed the results in the notebook property

notebook.run (2)

## Generate a new notebook in a new file

notebook.generate\_notebook ("myNewNotebook.ipynb");

## Generate an Octave script from a notebook

## Instantiate an object from a notebook file

notebook = jupyter\_notebook ("myNotebook.ipynb");

## Generate the Octave script

notebook.generate\_octave\_script ("jup\_script.m");

**参见:**jsondecode, jsonencode。

**11.12函数句柄和匿名函数**

将一个函数存储在变量中是非常方便的，这样它就可以传递给另一个函数。例如，执行数值最小化的函数需要访问应该最小化的函数。

**11.12.1函数句柄**

函数句柄是指向另一个函数的指针，用语法定义

@function-name

例如

f = @sin;

创建一个名为f的函数句柄，它引用函数sin。

函数句柄用于间接调用其他函数，或将一个函数作为参数传递给另一个函数，如quad或fsolve。例如:

f = @sin;

quad (f, 0, pi)

⇒ 2

您可以使用feval来使用函数句柄调用函数，或者简单地编写函数句柄的名称，后跟参数列表。如果没有参数，则必须使用空参数列表'()'。例如:

f = @sin;

feval (f, pi/4)

⇒ 0.70711

f (pi/4)

⇒ 0.70711

: tf = is\_function\_handle (x)

如果x是函数句柄则返回true。

参见:isa, typeinfo, class, functions。

: s = functions (fcn\_handle)

返回一个包含函数句柄fcn\_handle信息的结构。

结构s总是包含这三个字段:

function

函数名。对于匿名函数(没有名称)，这将是实际的函数定义。

type

函数类型。

anonymous

private函数是匿名的。

overloaded

函数重载现有函数。

simple

该函数是内置函数或m-file函数。

subfunction

该函数是m文件中的子函数。

nested

函数是嵌套的。

file

将被调用以执行该函数的m文件。对于匿名函数和内置函数，此字段为空。

此外，某些函数类型可能在附加字段中返回更多信息。

**警告:：**函数仅用于调试目的。它的行为将来可能会改变，程序不应该依赖于任何特定的输出格式。

**参见**：func2str, str2func。

: str = func2str (fcn\_handle)

返回一个字符串，其中包含函数句柄fcn\_handle引用的函数名。

参见:str2func, functions。

: hfcn = str2func (str)

返回由字符串str构造的函数句柄。

输入可以是函数的名称，例如"sin"，也可以是定义函数的字符串，例如"@(x) sin (x + pi)"。

编程注意:在大多数情况下，使用匿名函数语法并让Octave解析器创建函数句柄比使用str2func更好。例如:

hfcn = @sin ;

hfcn = @(x) sin (x + pi) ;

参见：func2str, functions.

: vars = symvar (str)

识别字符串str中的符号变量名。

常见的常量名称如i, j, pi, Inf和Octave函数如sin或plot被忽略。

任何标识的名称都以字符串的单元格数组返回。如果没有找到变量，则数组为空。

例子:

symvar ("x^2 + y^2 == 4")

⇒ {

[1,1] = x

[2,1] = y

}

**11.12.2匿名函数**

匿名函数是使用语法定义的

@(argument-list) expression

在参数列表中找不到的任何变量都从封闭作用域继承。匿名函数对于从表达式创建简单的未命名函数或包装对其他函数的调用以使其适应quad等函数的使用非常有用。例如,

f = @(x) x.^2;

quad (f, 0, 10)

⇒ 333.33

从表达式x.^2创建一个简单的未命名函数并将其传递给quad，

quad (@(x) sin (x), 0, pi)

⇒ 2

包装另一个函数，和

a = 1;

b = 2;

quad (@(x) betainc (x, a, b), 0, 0.4)

⇒ 0.13867

将具有多个参数的函数调整为quad所需的形式。在本例中，传递给beta的a和b的值是从当前环境继承的。

请注意，出于性能原因，最好使用现有Octave函数的句柄，而不是定义包装现有函数的匿名函数。如果代码写成。sin (x)的积分会快5倍

quad (@sin, 0, pi)

而不是使用匿名函数@(x) sin (x)。有许多运算符具有函数等价，可能比匿名函数更好的选择。而不是写

f = @(x, y) x + y

这应该被编码为

f = @plus

请参阅操作符重载，以获得也具有函数形式的操作符列表。

**11.13命令格式和功能**

除了上面描述的函数语法(即，调用一个函数，如fun (arg1, arg2，…))，一个函数可以使用命令语法调用(例如，调用一个函数，如fun arg1 arg2…)。在这种情况下，所有参数都以字符串的形式传递给函数。例如,

my\_command hello world

等同于

my\_command ("hello", "world")

命令调用的一般形式是

cmdname arg1 arg2 …

直接翻译成

cmdname ("arg1", "arg2", …)

如果在命令语法中需要将包含空格的参数传递给函数，则可以使用(双-)引号。例如,

my\_command "first argument" "second argument"

等同于

my\_command ("first argument", "second argument")

任何函数都可以作为命令使用，只要它接受字符串输入参数。例如:

toupper lower\_case\_arg

⇒ ans = LOWER\_CASE\_ARG

由于参数作为字符串传递给相应的函数，因此不可能传递存储在变量中的输入参数。在这种情况下，必须使用函数语法调用命令。例如:

strvar = "hello world";

toupper strvar

⇒ ans = STRVAR

toupper (strvar)

⇒ ans = HELLO WORLD

此外，不能使用命令语法将函数的返回值赋给变量。只有第一个返回参数被分配给内置变量ans。如果命令的输出参数应该被分配给一个变量，或者函数的多个输出参数应该被返回，则必须使用函数语法。

应该注意的是，混合使用命令语法和二进制运算符可能会在使用函数语法的数学和逻辑表达式中产生明显的歧义。例如，这三个表述

arg1 - arg2

arg1 -arg2

arg1-arg2

可能是用户想要的arg1和arg2之间的减法运算。但是，前两个也可以表示为对函数arg1的命令语法调用，在第一种情况下使用选项-和arg2，在第二种情况下使用选项-arg2。

Octave根据以下规则使用空格来解释这些表达式:

* 由普通符号组成的语句，没有任何仅用空格分隔的操作符，总是被视为命令语法:

arg1 arg2 arg3 ... argn

* 没有任何空格的语句总是被视为函数语法:

arg1+arg2

arg1&&arg2||arg3

arg1+=arg2\*arg3

* 如果第一个符号是一个常量(或特殊值命名常量pi、i、i、j、j、e、NaN或Inf)，后跟一个二进制运算符，则不管第二个符号后面是否有空格或什么，该语句都被视为函数语法:

7 -arg2

pi+ arg2

j \* arg2 -arg3

* 如果第一个符号是一个函数或变量，并且没有空格分隔操作符和第二个符号，则该语句被视为命令语法:

arg1 -arg2

arg1 &&arg2 ||arg3

arg1 +=arg2\*arg3

* 任何其他空格组合将导致该语句被视为函数语法。

注1:如果一个特殊值的命名常量被重新定义为变量，解释器仍然会用函数语法处理语句。

注2:试图在作为命令语法处理的命令中使用arg1变量将导致错误。

**11.14用八度分布的函数的组织**

Octave的许多标准函数都是作为函数文件分发的。它们按主题松散地组织在octave-home/share/octave/version/m的子目录中，以便更容易找到它们。

以下是所有函数文件子目录的列表，以及您将在其中找到的函数类型。

@ftp

FTP对象的类函数。

+containers

容器类的包。

audio

播放和录制声音的功能。

deprecated

过时的函数，最终将从Octave中删除。

elfun

初等函数，主要是三角函数。

general

各种矩阵操作，如fluid、rot90和triu，以及其他基本函数，如ismatrix、narginchk等。

geometry

与Delaunay三角剖分相关的函数。

gui

GUI元素(如对话框、消息框等)的函数。

help

Octave内置帮助系统的函数。

image

图像处理工具。这些功能需要X窗口系统。

io

输入输出函数

java

与Java集成相关的函数。

linear-algebra

线性代数的函数。

miscellaneous

不属于其他地方的函数。

ode

求解常微分方程的函数。

optimization

与最小化、优化和查找根相关的功能。

path

用于管理Octave用于查找函数的目录路径的函数。

pkg

用于在Octave中安装外部函数包的包管理器。

plot

显示和打印二维和三维图形的功能。

polynomial

处理多项式的函数。

Prefs

实现用户定义首选项的函数。

set

用于创建和操作唯一值集的函数。

signal

用于信号处理应用的函数。

sparse

处理稀疏矩阵的函数。

specfun

特殊函数，如贝塞尔或因子。

special-matrix

创建特殊矩阵形式的函数，如希尔伯特或范德蒙德矩阵。

startup

Octave的系统范围启动文件。

statistics

统计函数

Strings

杂项字符串处理函数。

testfun

用于对其他函数执行单元测试的函数。

time

与时间和日期处理相关的功能。

**12错误和警告**

Octave包括几个打印错误和警告消息的功能。当您编写的函数遇到异常情况需要采取特殊措施时，应使用本章描述的函数打印错误信息。

由于Octave的许多函数都使用了这些函数，因此理解它们也很有用，这样就可以处理错误和警告。

**12.1错误处理**

错误是当程序处于无法继续的状态时发生的事情。一个例子是调用函数时输入参数太少。在这种情况下，函数应该终止并给出一条错误消息，告知用户缺少输入参数。

由于错误可能在程序求值期间发生，因此能够检测到错误的发生，从而可以修复错误是非常方便的。这可以通过try语句中描述的try语句来实现。

**12.1.1抛出错误**

错误最常见的用法是检查函数的输入参数。如果调用函数f时没有任何输入参数，下面的示例将调用error函数。

function f (arg1)

if (nargin == 0)

error ("not enough input arguments");

endif

endfunction

当调用error函数时，它打印给定的消息并返回到Octave提示符。这意味着不会执行对error调用后的代码。

还可以为错误分配标识字符串。如果错误具有这样的ID，则用户可以捕获该错误，这将在下一节中描述。要为错误分配ID，只需使用两个字符串参数调用error，其中第一个是标识字符串，第二个是实际错误。注意，错误id的格式为"NAMESPACE: error - name "。命名空间"Octave"用于Octave自己的错误。任何其他字符串都可以作为用户自己错误的名称空间。

: error (template, …)

: error (id, template, …)

显示错误信息并停止m-file的执行。

在模板字符串模板的控制下，使用与printf函数族相同的规则格式化可选参数(请参阅格式化输出)，并在stderr流上打印结果消息。只有在模板字符串后面有附加参数的情况下，才会对单引号字符向量进行格式化。如果没有其他参数，则按字面意思使用模板字符串(即，不解释单引号字符向量中的任何转义序列)。该消息以' error: '作为前缀。

调用error还会设置Octave的内部错误状态，这样控制将返回到顶层，而无需评估任何进一步的命令。这对于终止函数或脚本非常有用。

如果错误消息没有以换行符结束，Octave将打印所有导致错误的函数调用的回溯信息。例如，给定以下函数定义:

function f () g (); end

function g () h (); end

function h () nargin == 1 || error ("nargin != 1"); end

调用函数f将产生一个消息列表，可以帮助您快速找到错误的确切位置:

f ()

error: nargin != 1

error: called from:

error: h at line 1, column 27

error: g at line 1, column 15

error: f at line 1, column 15

如果错误消息以换行符结束，Octave将打印该消息，但不会显示任何回溯消息，因为它将控制权返回给顶层。例如，修改上一个示例中的错误消息以换行符结束会导致Octave只打印一条消息:

function h () nargin == 1 || error ("nargin != 1\n"); end

f ()

error: nargin != 1

输入到error的空字符串("")将被忽略，代码将继续运行，就好像语句是NOP一样。这是为了与MATLAB兼容。它还可以编写诸如

err\_msg = "";

if (CONDITION 1)

err\_msg = "CONDITION 1 found";

elseif (CONDITION2)

err\_msg = "CONDITION 2 found";

…

endif

error (err\_msg);

它只会在发现错误时停止执行。

实现说明:为了与MATLAB兼容，无论模板是否用单引号定义，只要有两个或多个输入参数，模板中的转义序列(例如，"\n" => newline)都会被处理。要禁用转义序列扩展，请在序列前使用第二个反斜杠(例如，"\\n")或使用regexptranslate函数。

参见:警告，lastror。

由于在函数的输入出现错误时通常会使用错误，因此Octave支持函数来简化此类代码。当print\_usage函数被调用时，它会读取调用print\_usage函数的帮助文本，并显示一个有用的错误。如果帮助文本是用Texinfo编写的，则可能会显示一条错误消息，该消息只包含帮助文本的@deftypefn部分所描述的函数原型。当帮助文本不是用Texinfo编写时，错误消息包含整个帮助消息。

考虑下面的函数。

## -\*- texinfo -\*-

## @deftypefn {} f (@var{arg1})

## Function help text goes here…

## @end deftypefn

function f (arg1)

if (nargin == 0)

print\_usage ();

endif

endfunction

当在没有输入参数的情况下调用它时，它会产生以下错误。

f ()

-| error: Invalid call to f. Correct usage is:

-|

-| -- f (ARG1)

-|

-|

-| Additional help for built-in functions and operators is

-| available in the online version of the manual. Use the command

-| 'doc <topic>' to search the manual index.

-|

-| Help and information about Octave is also available on the WWW

-| at https://www.octave.org and via the help@octave.org

-| mailing list.

: print\_usage ()

: print\_usage (name)

打印函数名的用法消息。

在没有输入参数的情况下调用print\_usage函数时，将显示当前正在执行的函数的使用情况消息。

参见：help。

: beep ()

从扬声器产生哔哔声(或视觉铃声)。

该函数将报警字符“\a”发送到终端。根据用户的配置，这可能会产生一个可听到的哔哔声，一个可视的铃声，或者什么都没有。

参见:puts, fputs, printf, fprintf。

: val = beep\_on\_error ()

: old\_val = beep\_on\_error (new\_val)

: old\_val = beep\_on\_error (new\_val, "local")

查询或设置内部变量，该变量控制Octave在打印错误消息之前是否会尝试敲响终端铃声。

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。

**12.1.2捕获错误**

当发生错误时，可以使用try语句检测和处理，如try语句中所述。例如，下面的代码段对for循环期间发生的错误数量进行计数。

number\_of\_errors = 0;

for n = 1:100

try

…

catch

number\_of\_errors++;

end\_try\_catch

endfor

上面的例子对所有错误都一视同仁。然而，在许多情况下，有必要区分错误，并根据错误采取不同的操作。函数的作用是:返回一个包含上次发生错误信息的结构。例如，可以将上面的代码更改为计算与' \* '操作符相关的错误数量。

number\_of\_errors = 0;

for n = 1:100

try

…

catch

msg = lasterror.message;

if (strfind (msg, "operator \*"))

number\_of\_errors++;

endif

end\_try\_catch

endfor

或者，可以在catch关键字后面的变量中找到lasterror函数的输出，如下面的示例中显示了如何将错误重定向为警告:

try

…

catch err

warning(err.identifier, err.message);

…

end\_try\_catch

: lasterr = lasterror ()

: lasterror (err)

: lasterror ("reset")

查询或设置最后一次错误消息结构。

当不带参数调用时，返回一个包含最后一个错误消息和与此错误相关的其他信息的结构。结构的要素是:

message

最后一条错误消息的文本

identifier

此错误消息的消息标识符

stack

包含有关消息发生位置的信息的结构。如果无法获得信息，则该结构可能为空结构。结构的字段是:

file

发生错误的文件的名称

name

发生错误的函数名

line

发生错误的行号

column

一个可选字段，其中包含发生错误的列号

最后一个错误结构可以通过传递标量结构err作为输入来设置。err的任何与上述匹配的字段都被设置，而任何未指定的字段都用默认值初始化。

如果使用参数“reset”调用lasterror，则所有字段都被设置为默认值。

参见:lasterr, error, lastwarn。

: [msg, msgid] = lasterr ()

: lasterr (msg)

: lasterr (msg, msgid)

查询或设置最后一次错误信息。

在没有输入参数的情况下调用时，返回最后一个错误消息和消息标识符。

使用一个参数，将最后一个错误消息设置为msg。

还可以使用两个参数设置最后一个消息标识符。

参见:lastror, error, lastwarn。

下一个示例计算索引错误。使用函数lasterror返回的结构的字段标识符捕获错误。

number\_of\_errors = 0;

for n = 1:100

try

…

catch

id = lasterror.identifier;

if (strcmp (id, "Octave:invalid-indexing"))

number\_of\_errors++;

endif

end\_try\_catch

endfor

使用Octave分发的函数可能会发出以下错误之一。

Octave:bad-alloc

指示无法分配内存。

Octave:invalid-context

指示错误是由无法在调用该错误的范围内执行的操作生成的。例如，从Octave提示符调用print\_usage()函数时会引发此错误。

Octave:invalid-fun-call

指示以不正确的方式调用函数，例如输入参数数量错误。

Octave:invalid-indexing

数据类型索引错误，例如数组的实值索引，结构中不存在字段。

Octave:invalid-input-arg

指示使用无效输入参数调用函数。

Octave:undefined-function

指示对未定义的函数的调用。该函数可能存在，但Octave无法在搜索路径中找到它。

当错误被处理后，可能会再次引发它。当需要检测错误，但程序仍应中止时，这很有用。这可以使用重新抛出函数。现在可以更改前面的示例，以计算与' \* '操作符相关的错误数量，但如果发生其他类型的错误，仍然会中止。

number\_of\_errors = 0;

for n = 1:100

try

…

catch

msg = lasterror.message;

if (strfind (msg, "operator \*"))

number\_of\_errors++;

else

rethrow (lasterror);

endif

end\_try\_catch

endfor

: rethrow (err)

重新发出由err定义的先前错误。

Err是一个必须至少包含“message”和“identifier”字段的结构。Err还可以包含一个字段“stack”，该字段提供有关假定错误位置的信息。通常从lastror返回err。

参见:lastror, lastr, error。

: err = errno ()

: err = errno (val)

: err = errno (name)

查询或设置系统相关变量errno。

在没有输入的情况下调用时，返回errno的当前值。

当使用数字输入值调用时，将errno的当前值设置为指定值。errno的前一个值返回为err。

当使用字符串名称调用时，返回errno的数值，该数值对应于指定的错误代码。如果name不是可识别的错误代码，则返回-1。

参见:errno\_list。

: S = errno\_list ()

返回一个包含系统相关errno值的结构。

参见:errno。

**12.1.3错误恢复**

Octave提供了几种从错误中恢复的方法。有try/catch块，unwind\_protect/unwind\_protect\_cleanup块，最后是onCleanup命令。

onCleanup命令将一个普通的Octave变量(触发器)与一个任意函数(动作)关联起来。每当Octave变量不存在时——无论是由于函数返回、错误，还是仅仅因为变量被clear删除了——赋值的函数就会被执行。

该函数可以执行清除所需的任何操作，例如关闭打开的文件句柄、打印错误消息或将全局变量恢复到初始值。最后一个例子对于Octave代码来说是一个非常方便的习惯用法。例如:

function rand42

old\_state = rand ("state");

restore\_state = onCleanup (@() rand ("state", old\_state));

rand ("state", 42);

…

endfunction # rand generator state restored by onCleanup

: obj = onCleanup (function)

创建一个特殊对象，在销毁时执行给定的功能。

如果将对象复制到多个变量（或单元格或结构数组元素）或从函数返回，则只有在清除对象的最后一个副本后才会执行函数。

输入函数是函数的句柄。句柄可以指向匿名函数，以便直接在onCleanup调用中放置命令。

编程说明：如果创建了多个本地onCleanup变量，则未指定调用它们的顺序。有关类似功能，请参阅 unwind.protect声明。

例如：

octave:1> trigger = onCleanup (@() disp ('onCleanup was executed'));

octave:2> clear trigger

onCleanup was executed

octave:3

**12.2处理警告**

与错误类似，在发生意外情况时发出警告。与错误不同，警告不会中止当前正在运行的程序。一个简单的警告例子是当一个数字被零除时。在这种情况下，Octave将发出警告并将值Inf赋值给结果。

a = 1/0

-| warning: division by zero

⇒ a = Inf

**12.2.1发出警告**

使用warning函数可以从任何代码发出警告。在最简单的形式中，warning函数接受一个描述警告的字符串作为其输入参数。例如，下面的代码控制变量' a '是否为负，如果不是，则发出警告并将' a '设置为零。

a = -1;

if (a < 0)

warning ("'a' must be non-negative. Setting 'a' to zero.");

a = 0;

endif

-| 'a' must be non-negative. Setting 'a' to zero.

由于警告对正在运行的程序不是致命的，因此不可能使用try语句或类似的语句捕获警告。但是，可以使用lastwarn函数将最后一个警告作为字符串访问。

也可以将标识字符串分配给警告。如果一个警告有这样一个ID，用户可以启用和禁用这个警告，这将在下一节中描述。要为警告分配ID，只需使用两个字符串参数调用warning，其中第一个是标识字符串，第二个是实际的警告。注意，警告id的格式为“NAMESPACE: warning - name”。命名空间"Octave"用于Octave自己的警告。任何其他字符串都可以作为用户自己的警告的名称空间。

: warning (template, …)

: warning (id, template, …)

: warning ("on", id)

: warning ("off", id)

: warning ("error", id)

: warning ("query", id)

: warning (state, id, "local")

: warning (warning\_struct)

: warning\_struct = warning (…)

: warning (state, mode)

显示警告信息或控制Octave的警告系统的行为。

第一个调用表单使用模板模板和可选的附加参数在标准错误流上显示消息。该消息使用与printf函数族相同的规则进行格式化(请参阅格式化输出)，并以字符串' warning: '作为前缀。当您想要将异常情况通知用户时，应该使用此函数，但只能在程序有意义的情况下使用。例如:

warning ("foo: maybe something wrong here");

如果警告消息没有以换行符结束，Octave将打印所有导致警告的函数调用的回溯信息。如果警告消息以换行符结束，Octave将在将控制返回到顶层时抑制回溯消息。有关更多详细信息和示例，请参见错误。

可选的警告标识符id允许用户启用或禁用由该标识符标记的警告。消息标识符是“NAMESPACE:WARNING-NAME”形式的字符串。Octave自己的警告使用"Octave"命名空间(参见warning\_ids)。例如:

warning ("MyNameSpace:check-something",

"foo: maybe something wrong here");

第二个调用表单用于更改和/或查询警告的状态。第一个输入参数必须是一个字符串状态(“on”、“off”、“error”或“query”)，后跟一个可选的警告标识符id或“all”(默认值)。

可选的输出参数warning\_struct是一个带有字段"state"和"identifier"的结构或结构数组。state参数可以有以下值:

"on"|"off":

启用或禁用由id标识的警告的显示，并可选择返回它们以前的状态粗壮。

"error":

将由id标识的警告转换为错误，并可选择返回其先前的状态。

"query":

返回由id标识的警告的当前状态。

一个结构体或结构体数组warning\_struct，带有字段“state”和“identifier”，可以作为输入来获得相同的结果。下面的例子展示了如何暂时禁用警告，然后恢复其原始状态:

loglog (-1:10);

## Disable the previous warning and save its original state

[~, id] = lastwarn ();

warnstate = warning ("off", id);

loglog (-1:10);

## Restore its original state

warning (warnstate);

如果提供了最后一个参数"local"，则将临时设置警告状态，直到当前函数结束。对本地设置的警告状态的更改会影响当前函数和从当前作用域调用的所有函数。在从当前函数返回时恢复先前的警告状态。如果在顶级工作区中使用，“local”选项将被忽略。

在没有输入参数的情况下，warning()等同于warning ("query"， "all")，只是在没有输出参数的情况下，警告的状态会显示在stderr上。

警告系统的详细程度也可以通过两种模式来控制:

"backtrace":

启用/禁用在警告消息后显示堆栈跟踪

"verbose":

启用/禁用在警告消息后显示附加信息

在这种情况下，state参数只能是“on”或“off”。

实现说明:为了与MATLAB兼容，无论模板是否用单引号定义，只要有两个或多个输入参数，模板中的转义序列(例如，"\n" => newline)都会被处理。要禁用转义序列扩展，请在序列前使用第二个反斜杠(例如，"\\n")或使用regexptranslate函数。

**参见:**warning\_ids, lastwarn, error。

: [msg, msgid] = lastwarn ()

: lastwarn (msg)

: lastwarn (msg, msgid)

查询或设置最后一次警告信息。

在没有输入参数的情况下调用时，返回最后一个警告消息和消息标识符。

通过一个参数，将最后一个警告消息设置为msg。

还可以使用两个参数将最后一个消息标识符设置为msgid。

参见:warning, lasterror, lasterr。

使用Octave分发的函数可以发出以下警告之一。

Octave:abbreviated-property-match

如果启用了Octave: compressed -property-match警告，则如果在指定对象属性的操作中使用了不精确或不明确的匹配，则会打印警告。例如，对于图形对象fig，带有属性'displayname'，如果启用了Octave:缩写-property-match警告，则get (fig， 'dis')将引发警告。缺省情况下，启用Octave: compressed -property-match警告。

Octave:addpath-pkg

如果启用了Octave:addpath-pkg警告，当包目录(即+package\_name)被添加到路径中时，Octave将发出警告。通常，只有包含包目录的父目录应该被添加到加载路径中。缺省情况下，启用Octave:addpath-pkg警告。

Octave:array-as-logical

如果启用了Octave:array-as-logical警告，当在If、while或until语句中使用大小大于1x1的数组作为真值时，Octave将发出警告。默认情况下，禁用Octave:array-as-logical警告。

Octave:array-to-scalar

如果启用了Octave:array-to-scalar警告，当尝试从数组到标量值的隐式转换时，Octave将发出警告。默认情况下，Octave:数组到标量警告是禁用的。

Octave:array-to-vector

如果启用了Octave:array-to-vector警告，当尝试从数组到向量值的隐式转换时，Octave将发出警告。默认情况下，Octave:array-to-vector警告被禁用。

Octave:assign-as-truth-value

如果启用了Octave:assign-as- true -value警告，则会对以下语句发出警告

if (s = t)

…

由于这样的陈述并不常见，很可能是意图写

if (s == t)

…

代替。

有时，在while或if语句的条件中编写包含赋值的代码是有用的。例如，像

while (c = getc ())

…

在C编程中很常见。

通过禁用Octave:assign-as- true -value警告，可以避免对此类语句的所有警告，但这也可能导致真正的错误，如

if (x = 1) # intended to test (x == 1)!

…

飞掠而过。

在这种情况下，可以通过使用一组额外的括号来编写特定语句来抑制错误。例如，将前面的示例写成

while ((c = getc ()))

…

将阻止为该语句打印警告，同时允许Octave对条件上下文中使用的其他赋值进行警告。

缺省情况下，启用Octave:assign-as- true -value警告。

Octave:autoload-relative-file-name

如果启用了Octave: autolload -relative-file-name, Octave将在解析带有函数文件相对路径的autolload()函数调用时发出警告。当在PKG\_ADD文件中使用autolload()调用时，当PKG\_ADD文件与autolload()命令引用的.oct文件不在同一目录中时，通常会发生这种情况。默认情况下，启用Octave: autolload -relative-file-name警告。

Octave:charmat-truncated

如果启用了Octave:charmat-truncated警告，则在将包含多行的字符矩阵转换为字符串时打印警告。在这种情况下，Octave解释器只保留第一行，而丢弃其他行。默认情况下，Octave:字符截断警告是启用的。

Octave:classdef-to-struct

如果启用了Octave:classdef-to-struct警告，则当使用struct (CLASSDEF\_OBJ)强制将classdef对象转换为struct时将发出警告。转换从对象中删除访问限制，并使私有和受保护的属性可见。默认情况下，Octave:classdef-to-struct警告是启用的。

Octave:colon-complex-argument

如果启用了Octave:冒号复杂参数警告，则当冒号操作符的三个参数(基数、增量、限制)之一是复杂值时发出警告。例如，1:3\*i将导致发出警告。默认情况下，启用Octave:冒号复杂参数警告。

Octave:colon-nonscalar-argument

如果启用了Octave:冒号-非标量参数警告，则当冒号操作符的三个参数(基数、增量、限制)中的一个不是标量时发出警告。例如，1:[3,5]将导致发出警告。默认情况下，启用Octave:冒号-非标量参数警告。

Octave:data-file-in-path

如果启用了Octave:data-file-in-path警告，则当Octave没有直接找到文件操作(如load或fopen)的目标，但能够在Octave的文件搜索路径中找到该文件时，会发出警告。该警告可能表明正在使用与程序员预期不同的文件目标。默认情况下，启用Octave:data-file-in-path警告。

Octave:datevec:date-format-spec

如果启用Octave:datevec:date-format-spec警告，则如果日期格式规范包含可疑的日期或时间说明符，则打印警告。典型的错误模式是使用大写日期说明符或小写时间说明符。默认情况下，启用Octave:datevec:date-format-spec警告。

Octave:deprecated-function

如果启用了Octave:deprecated-function警告，则当Octave遇到过时并计划从Octave中删除的函数时，将发出警告。默认情况下，启用Octave:deprecated-function警告。

Octave:deprecated-keyword

如果启用了Octave:deprecated-keyword警告，则当Octave遇到过时的关键字并计划从Octave中删除时，将发出警告。缺省情况下，启用Octave:deprecated-keyword警告。

Octave:deprecated-option

如果启用了Octave:deprecated-option警告，则在使用过时的选项或函数输入时发出警告。默认情况下，启用Octave:deprecated-option警告。

Octave:deprecated-property

如果启用了Octave:deprecated-property警告，则当Octave遇到过时且计划从Octave中删除的图形属性时，将发出警告。默认情况下，启用Octave:deprecated-property警告。

Octave:eigs:UnconvergedEigenvalues

如果Octave:eigs:UnconvergedEigenvalues警告被启用，那么如果计算的特征值的数量少于请求的特征值的数量，eigs函数将发出警告。默认情况下，Octave:eigs:UnconvergedEigenvalues警告是启用的。

Octave:empty-index

如果启用了Octave:empty-index警告，那么每当在没有索引的情况下使用索引操作符时，Octave将发出警告，例如x()。默认情况下，启用Octave:empty-index警告。

Octave:erase:chararray

如果启用Octave:erase:chararray警告，则如果输入模式是字符数组而不是字符串或字符串的单元数组，则擦除函数将发出警告。缺省情况下，启用Octave:erase:chararray警告。

Octave:function-name-clash

如果启用了Octave:function-name-clash警告，当Octave发现函数文件中定义的函数的名称与文件名不同时，将发出警告。(如果名称不一致，则忽略文件内声明的名称。)缺省情况下，启用Octave:function-name-clash警告。

Octave:future-time-stamp

如果启用了Octave:future-time-stamp警告，Octave将在发现具有未来时间戳的函数文件时打印警告。默认情况下，启用Octave:future-time-stamp警告。

Octave:glyph-render

如果启用了Octave: glyh -render警告，如果字符的字形无法用当前字体呈现，Octave将打印警告。默认情况下，Octave:字形渲染警告是启用的。

Octave:imag-to-real

如果启用了Octave: image -to-real警告，则会对复数到实数的隐式转换打印警告。默认情况下，Octave: image -to-real警告是禁用的。

Octave:infinite-loop

如果启用了Octave:infinite-loop警告，则在检测到无限循环时打印警告，例如for i = 1:Inf while(1)。默认情况下，启用了Octave:infinite-loop警告。

Octave:language-extension

在使用Octave语言独有的特性时打印警告，这些特性在MATLAB中可能仍然缺失。默认情况下，Octave:language-extension警告是禁用的。Octave的——traditional或——braindead启动选项也可以使用，参见命令行选项。

Octave:legacy-function

如果启用了Octave:legacy-function警告，则当Octave遇到MATLAB建议应避免的函数时将发出警告。该函数可能在将来的某个时候过时并被删除，在这种情况下，警告将更改为Octave:deprecated-function，并且该函数在被删除之前将继续存在两个Octave的后续版本。默认情况下，启用Octave:legacy-function警告。

Octave:logical-conversion

如果启用Octave:逻辑转换警告，则如果发生数组从数字到布尔的隐式转换，并且数组中的任何元素不等于0或1，则打印警告。默认情况下，Octave:逻辑转换警告是启用的。

Octave:lu:sparse\_input

如果启用了Octave:lu:sparse\_input警告，当使用稀疏输入和少于四个输出参数调用lu函数时，Octave将发出警告。在这种情况下，不会执行保持稀疏性的列排列，结果可能不准确。默认情况下，Octave:lu:sparse\_input警告是启用的。

Octave:missing-glyph

如果启用了Octave: glyh -render警告，如果当前字体没有为使用的字符提供字形，Octave将打印一个警告。默认情况下，Octave:missing-glyph警告是启用的。

Octave:missing-semicolon

如果启用了Octave:missing-分号警告，当函数定义中的语句没有以分号结束时，Octave将发出警告。默认情况下，Octave:missing-分号警告是禁用的。

Octave:mixed-string-concat

如果启用了Octave:mixed-string-concat警告，则在连接双引号和单引号字符串的混合物时打印警告。默认情况下，禁用Octave:mixed-string-concat警告。

Octave:nearly-singular-matrix

Octave:singular-matrix

如果一个(近似)奇异矩阵倒转，就会发出这些警告。默认情况下，Octave:near -singular-matrix和Octave:singular-matrix警告是启用的。

Octave:neg-dim-as-zero

如果启用了Octave: negative -dim-as-zero警告，则打印以下表达式的警告

eye (-1)

缺省情况下，禁用Octave: -dim-as-zero警告。

Octave:noninteger-range-as-index

如果启用了Octave:noninteger-range-as-index警告，则如果使用包含非整数值的范围对数组进行索引，则会打印警告。例如,

a = [1 2 3 4 5];

b = 2.2:4.2

⇒ 1.2 2.2 3.2

a(b)

⇒ 2 3 4

如果启用了Octave:noninteger-range-as-index警告，则会引发警告。默认情况下，启用Octave:noninteger-range-as-index警告。

Octave:num-to-str

如果启用了Octave:num-to-str警告，则在使用矩阵表示法中的字符串和数字混合构造字符串时，将打印数字到其UTF-8编码等效字符的隐式转换的警告。例如,

[ "f", 111, 111 ]

⇒ "foo"

如果启用了Octave:num-to-str警告，则会引发警告。默认情况下，启用Octave:num-to-str警告。

Octave:possible-matlab-short-circuit-operator

如果启用了Octave:possible-matlab-short-circuit-operator警告，Octave将警告在If或while条件中使用非短路操作符&和|。它们通常不会短路，但在特定条件下使用时确实会短路。缺省情况下，使能Octave:possible-matlab-short-circuit-operator警告。

Octave:pow2:imaginary-ignored

如果启用了Octave:pow2:imaginary-ignore警告，则如果pow2的任何一个输入是复杂的，则打印警告。默认情况下，启用Octave:pow2:imaginary-ignored警告。

Octave:recursive-path-search

如果启用了Octave:recursive-path-search警告，如果addpath使用双斜杠，Octave将发出警告。缺省情况下，启用Octave:recursive-path-search警告。

Octave:remove-init-dir

path函数改变了Octave用来查找函数的搜索路径。可以将路径设置为一个不包括Octave自己的内置函数的值。如果启用了Octave:remove-init-dir警告，那么当路径函数被以可能导致Octave无法工作的方式使用时，Octave将发出警告。缺省情况下，启用Octave:remove-init-dir警告。

Octave:reload-forces-clear

如果从同一个文件中加载了多个函数，Octave必须在重新加载任何一个函数之前清除所有函数。如果启用了Octave:reload-forces-clear警告，Octave会在发生这种情况时警告你，并打印一个强制清除的附加函数列表。默认情况下，启用Octave:reload-forces-clear警告。

Octave:separator-insert

如果在文字矩阵中可能自动插入逗号或分号，则打印警告。默认情况下，禁用Octave:separator-insert警告。

Octave:shadowed-function

如果启用了Octave:shadowed-function警告，如果在搜索路径中添加了包含遮蔽核心函数的函数，Octave将发出警告。默认情况下，启用Octave:shadowed-function警告。

Octave:single-quote-string

如果使用单引号字符引入字符串常量，则打印警告。默认情况下，禁用Octave:single-quote-string警告。

Octave:sqrtm:SingularMatrix

如果启用了Octave:sqrtm:SingularMatrix警告，则如果使用奇异输入矩阵调用矩阵平方根函数sqrtm，则会打印警告。默认情况下，Octave:sqrtm:SingularMatrix警告是启用的。

Octave:str-to-num

如果启用了Octave:str-to-num警告，则对字符串到其数字UTF-8编码字节序列的隐式转换打印警告。例如,

"abc" + 0

⇒ 97 98 99

如果启用了Octave:str-to-num警告，则会引发警告。默认情况下，禁用Octave:str-to-num警告。

Octave:LaTeX:internal-error

如果启用了Octave:LaTeX:internal-error警告，则每当用于绘图中的文本的LaTeX渲染器遇到问题时，都会打印一个警告。默认情况下，启用Octave:LaTeX:internal-error警告。

Octave:unimplemented-matlab-functionality

如果启用了Octave:unimplemented- MATLAB -functional警告，则当使用了一个MATLAB代码结构(Octave解释器将其解析为有效，但Octave尚未实现该功能)时，将打印一个警告。默认情况下，启用Octave:unimplemented-matlab-functional警告。

Octave:variable-switch-label

如果启用了Octave:variable-switch-label警告，如果开关标签不是常量或常量表达式，Octave将打印警告。缺省情况下，关闭Octave:variable-switch-label警告。

**12.2.2启用和禁用警告**

警告功能还允许您控制实际打印到屏幕上的警告。如果使用字符串参数“on”或“off”调用警告函数，则所有警告将被启用或禁用。

还可以通过字符串标识启用和禁用单个警告。下面的代码将发出警告

warning ("example:non-negative-variable",

"'a' must be non-negative. Setting 'a' to zero.");

而下面的代码不会发出警告

warning ("off", "example:non-negative-variable");

warning ("example:non-negative-variable",

"'a' must be non-negative. Setting 'a' to zero.");

**13调试**

Octave包含一个内置调试器来帮助开发脚本。这可用于在某一点或满足某些条件时中断Octave脚本的执行。一旦执行停止并进入调试模式，就可以检查和修改执行停止点的符号表以检查错误。

正常的命令行编辑和历史功能在调试模式下可用。

**13.1进入调试模式**

有两种基本方法可以中断Octave脚本的执行。这些是断点(请参阅下一节讨论的断点)和基于某些条件的中断。

Octave支持基于debug\_on\_interrupt, debug\_on\_warning和debug\_on\_error函数中设置的值的三种方式来停止执行。

: val = debug\_on\_interrupt ()

: old\_val = debug\_on\_interrupt (new\_val)

: old\_val = debug\_on\_interrupt (new\_val, "local")

查询或设置内部变量，该变量控制Octave在接收到中断信号(通常由C-c生成)时是否尝试进入调试模式。

如果在到达调试模式之前接收到第二个中断信号，则会发生正常中断。

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:debug\_on\_error, debug\_on\_warning。

: val = debug\_on\_warning ()

: old\_val = debug\_on\_warning (new\_val)

: old\_val = debug\_on\_warning (new\_val, "local")

查询或设置内部变量，该变量控制在遇到警告时Octave是否尝试进入调试器。

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:debug\_on\_error, debug\_on\_interrupt。

: val = debug\_on\_error ()

: old\_val = debug\_on\_error (new\_val)

: old\_val = debug\_on\_error (new\_val, "local")

查询或设置内部变量，该变量控制在遇到错误时Octave是否尝试进入调试器。

这还将禁止打印正常的回溯消息(您将只看到顶级错误消息)。

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:debug\_on\_warning, debug\_on\_interrupt。

**13.2退出调试模式**

使用dbcont或return退出调试模式，继续脚本的正常执行。

: dbcont

离开命令行调试模式，继续正常执行代码。

参见:dbstep, dbquit。

要退出调试模式并直接返回到提示符，而不执行任何额外的代码，请使用dbquit。

: dbquit

: dbquit all

立即退出调试模式，无需进一步执行代码。不带参数，退出当前调试级别。使用参数all，退出所有调试级别并返回到Octave提示符。

参见:dbcont, dbstep。

最后，在调试提示符处输入exit或quit将导致Octave正常终止。

**13.3断点**

可以通过使用dbstop函数在任何m-file函数中设置断点。

: dbstop fcn

: dbstop fcn line

: dbstop fcn line1 line2 …

: dbstop line1 …

: dbstop in fcn

: dbstop in fcn at line

: dbstop in fcn at line if "condition"

: dbstop in class at method

: dbstop if event

: dbstop if event ID

: dbstop (bp\_struct)

: rline = dbstop …

为内置调试器设置断点。

FCN是当前路径上的函数名。当已经处于调试模式时，可以省略fcn参数，并使用当前函数。子函数中的断点使用范围操作符' > '设置。例如:If文件。M有子函数fcn2，那么fcn2中的断点可以通过file>fcn2指定。

Line是要中断的行号。如果未指定line，则默认为文件fcn.m中的第一个可执行行。可以在单个命令中指定多个行;当使用函数语法时，行也可以作为单个向量参数传递([line1, line2，…])。

条件是可以在断点处存在的代码上下文中求值的任何Octave表达式。当遇到断点时，将计算condition，如果condition为真则停止执行。如果不能对condition求值，例如因为它引用了一个未定义的变量，则会抛出错误。有副作用的表达式(比如y++ > 1)会改变变量，通常应该避免使用。包含引号(' '，' ')或注释字符(' # '，' % ')的条件必须用引号括起来。(这不适用于从编辑器的上下文菜单输入的条件。)例如:

dbstop in axis at 246 if 'any (opt == "x")'

指定事件的表单不会在给定函数和行号处导致特定的断点。相反，它导致在遇到某些意外事件时进入调试模式。可能的值有

error

当报告错误时停止。这相当于同时指定debug\_on\_error (true)和debug\_on\_interrupt (true)。

caught error

当错误被try-catch块捕获(尚未实现)时停止。

interrupt

当发生中断时停止(Ctrl-C)。

naninf

当代码返回非有限值(尚未实现)时停止。

warning

当出现警告时停止。这相当于指定debug\_on\_warning (true)。

事件错误、捕获错误和警告都可以后跟一个指定错误ID或警告ID的字符串。如果这样做了，只有使用指定ID的错误才会导致执行停止。要在一组id上停止，必须发出多个dbstop命令。

可以使用具有相同语法的dbclear命令删除断点和事件。

通过发出命令bp\_state = dbstatus，可以保存所有断点并立即恢复它们;…;dbstop (bp\_state)。

可选的输出行是设置断点的实际行号。如果指定行不可执行，则此值可以与指定行不同。例如，如果在空白行尝试断点，那么Octave将在下一个可执行行设置真正的断点。

当文件被重新解析时，比如在GUI之外修改文件时，文件中的所有断点都会被清除。

参见:dbclear, dbstatus, dbstep, debug\_on\_error, debug\_on\_warning, debug\_on\_interrupt。

类方法中的断点也被支持(例如，dbstop ("@class/method"))。但是，不能在内置函数(例如，sin等)或动态加载的函数(例如，oct-files)中设置断点。

要在进入函数时立即设置断点，请使用行号1，或者完全省略行号，只给出函数名。当设置断点时，Octave将忽略前面的注释块，断点将设置在函数中的第一个可执行语句上。例如:

dbstop ("asind", 1)

⇒ 29

注意，返回值29意味着断点被有效地设置为第29行。可以使用dbstatus查询函数中断点的状态。

: dbstatus

: dbstatus fcn

: bp\_list = dbstatus …

报告活动断点的位置。

在没有输入或输出参数的情况下调用时，打印所有带有断点的函数列表以及设置这些断点的行号。

如果指定了函数名fcn，则只报告指定函数及其子函数的断点。

可选返回参数bp\_list是一个struct数组，包含以下字段:

name

带有断点的函数的名称。m文件中的子函数，比如fcn2，比如file。M，指定为file>fcn2。

file

函数代码所在的m文件的名称。

line

带有断点的行号。

cond

为使断点处于活动状态必须满足的条件，或者为无条件断点提供空字符串。

如果dbstop If error为true，但没有指定显式id，则返回值将有一个名为" errors "的空字段。如果指定了ID，则errors字段中每个ID有一行。如果dbstop If error为false，则没有" errors "字段。“warn”字段由dbstop类似地设置。

**参见:**dbstop, dbclear, dbwhere, dblist, dbstack。

重用前面的示例，dbstatus ("asind")将返回29。然后可以使用dbclear函数清除列出的断点。

: dbclear fcn

: dbclear fcn line

: dbclear fcn line1 line2 …

: dbclear line …

: dbclear all

: dbclear in fcn

: dbclear in fcn at line

: dbclear if event

: dbclear ("fcn")

: dbclear ("fcn", line)

: dbclear ("fcn", line1, line2, …)

: dbclear ("fcn", line1, …)

: dbclear (line, …)

: dbclear ("all")

删除函数fcn中行号line处的断点。

参数是

fcn

作为字符串变量的函数名。当已经处于调试模式时，可以省略此参数，并将使用当前函数。

line

要从中删除断点的行号。多个行可以作为单独的参数或作为一个向量给出。

event

错误、中断或警告等事件(详细信息请参见dbstop)。

在没有行号说明的情况下调用时，将清除指定函数中的所有断点。

如果请求行不是断点，则不执行任何操作。

特殊关键字"all"将清除所有文件中的所有断点。

**参见:**dbstop、dbstatus、dbwhere。

也可以在子函数中设置断点。例如，如果一个文件包含函数

function y = func1 (x)

y = func2 (x);

endfunction

function y = func2 (x)

y = x + 1;

endfunction

然后可以在子函数的开始处直接设置断点

dbstop func1>func2

⇒ 5

请注意，' > '是将子函数与包含子函数的m文件区分开来的字符。

在Octave脚本中设置断点的另一种简单方法是使用键盘函数。

: keyboard ()

: keyboard ("prompt")

停止m-file的执行，进入调试模式。

当键盘函数执行时，Octave打印提示并等待用户输入。然后计算输入字符串并打印结果。这样就可以检查函数中变量的值，并在必要时分配新值。要离开提示符并返回正常执行，请键入' return '或' dbcont '。键盘函数不返回退出状态。

如果在没有参数的情况下调用keyboard，则使用默认提示' debug> '。

参见:dbstop, dbcont, dbquit。

键盘函数被放置在脚本中用户希望停止执行的位置。它自动将正在运行的脚本设置为调试模式。

**13.4调试模式**

还有三个额外的支持功能，允许用户查找脚本Octave在执行过程中进入调试模式的位置，并打印脚本中围绕Octave进入调试模式点的代码。

: dbwhere

在调试模式下，报告当前停止执行的文件和行号。

参见:dbstack、dblist、dbstatus、dbcont、dbstep、dup、dbdown。

: dbtype

: dbtype lineno

: dbtype startl:endl

: dbtype startl:end

: dbtype fcn

: dbtype fcn lineno

: dbtype fcn startl:endl

: dbtype fcn startl:end

显示带有行号的脚本文件。

在调试模式下不带参数调用时，显示当前正在调试的脚本文件。

可选范围规范可用于仅列出文件的一部分。特殊关键字“end”是文件最后一行的有效行号规范。

当用函数名调用时，用行号列出该脚本文件。

另请参见:dbist、dbwhere、dbstatus、dbstop。

: dblist

: dblist n

在调试模式下，以当前要执行的行为中心列出正在调试的函数的n行。

如果未指定，n默认为10(+/- 5行)

参见:dbwhere, dbtype, dbstack。

您还可以使用isdebugmode来确定调试器当前是否处于活动状态。

: tf = isdebugmode ()

在调试模式下返回true，否则返回false。

参见:dbwhere, dbstack, dbstatus。

调试模式还允许使用命令dbstep对函数进行单行步进。

: dbstep

: dbstep n

: dbstep in

: dbstep out

: dbnext …

在调试模式下，执行接下来的n行代码。

如果省略n，则执行下一行代码。如果下一行代码本身是根据m文件定义的，则保留在现有函数中。

使用dbstep in将导致下一行的执行步进到下一行定义的任何m文件中。

使用dbstep out将导致执行继续，直到当前函数返回。

编程说明：dbnext 是 dbstep 的别名，可以互换使用。

参见:dbcont, dbquit。

在调试模式下，RETURN键将执行最后输入的命令。例如，在遇到断点并输入一次dbstep之后，这是很有用的。在那之后，人们可以一行一行地通过代码，只需要一个按键。可以使用auto\_repeat\_debug\_command函数禁用此特性。

: val = auto\_repeat\_debug\_command ()

: old\_val = auto\_repeat\_debug\_command (new\_val)

: old\_val = auto\_repeat\_debug\_command (new\_val, "local")

查询或设置内部变量，该变量控制在输入行为空时(只输入RET)是否自动重复调试命令。

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。

**13.5调用栈**

正在调试的函数可能是一系列函数调用的叶节点。在检查当前子例程中的值后，可能会发现问题发生在较早的代码段中。使用dup和dbdown在一系列函数调用中上下移动，以定位变量首次使用错误值的位置。Dbstack显示了整个函数调用系列，以及当前正在哪个级别进行调试。

: dbstack

: dbstack n

: dbstack -completenames

: [stack, idx] = dbstack (…)

显示或返回当前调试函数堆栈信息。

对于可选参数n，省略最内层的n个堆栈帧。

尽管可以接受，但参数-completenames将被静默忽略。Octave总是返回绝对文件名。

参数n和-completenames都可以指定，并且可以以任意顺序出现。

可选返回参数stack是一个struct数组，包含以下字段:

file

函数代码所在的m文件的名称。

name

带有断点的函数的名称。

line

活动断点的行号。

column

断点开始所在行的列号。

返回参数idx指定堆栈结构数组的哪个元素当前处于活动状态。

请参见:dup、dbdown、dbwhere、dblist、dbstatus。

: dbup

: dbup n

在调试模式下，将执行堆栈向上移动n帧。

如果n被省略，向上移动一帧。

参见:dbstack, dbdown。

: dbdown

: dbdown n

在调试模式下，向下移动执行堆栈n帧。

如果n被省略，则向下移动一帧。

参见:dbstack, dup。

**13.6分析**

Octave支持对每个函数级别的代码执行进行分析。如果启用了分析，则在运行Octave代码时记录对函数的每次调用(支持内置函数、操作符、oct和mex文件中的函数、Octave代码中的用户定义函数和匿名函数)。之后，这些数据可以帮助分析代码行为，并且特别有助于找到代码中的“热点”，这些“热点”消耗了大量的计算时间，并且是花费优化努力的最佳目标。

用于分析的主要命令是profile，它可用于启动或停止分析程序，也可用于随后查询收集的数据。数据以Octave数据结构返回，然后可以由其他例程或工具检查或进一步处理。

: profile on

: profile off

: profile resume

: profile clear

: S = profile ("status")

: T = profile ("info")

控制内置分析器。

profile on

启动分析器。任何以前收集的数据都将被清除。

profile off

停止剖析。稍后可以使用T = profile(“info”)检索和检查收集到的数据。

profile clear

清除所有收集的分析器数据。

profile resume

在不清除旧数据的情况下重新启动分析。所有新收集的统计信息将添加到现有统计信息中。

S = profile ("status")

返回一个结构，其中包含有关分析器当前状态的信息。目前，唯一的字段是ProfilerStatus，它要么是“on”，要么是“off”。

T = profile ("info")

在结构t中返回收集到的分析统计信息。平面配置信息在FunctionTable字段中返回，FunctionTable是一个结构数组，每个条目对应于一个被调用的函数，并且存在分析统计信息。此外，字段hierarchy包含了分层调用树。每个节点都有一个进入FunctionTable的索引，该索引标识它所对应的函数，以及用于调用次数和在调用树中此级别上花费的时间的数据字段。

参见：profshow, profexplore。

获取所收集数据的概览的一种简单方法是profshow。这个函数接受profile返回的分析器数据作为输入，并打印一个平面配置文件，例如:

Function Attr Time (s) Calls

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

>myfib R 2.195 13529

binary <= 0.061 13529

binary - 0.050 13528

binary + 0.026 6764

这表明大部分运行时间用于执行函数' myfib '，还有一小部分用于计算列出的二进制运算符。此外，它还显示了函数被调用的频率，并且分析器还记录了它是递归的。

: profshow (data)

: profshow (data, n)

: profshow ()

: profshow (n)

显示单个函数剖析器的结果。

打印最关键的n个函数的分析器数据(执行时间、调用次数)。结果按照在每个函数中花费的总时间降序排序。如果n未指定，则默认为20。

输入数据是配置文件(“info”)返回的结构。如果未指定，profshow将使用当前配置文件数据集。

对于递归函数，属性列显示' R '，对于所有其他函数类型，属性列为空白。

参见:profexplore, profile。

: profexport (dir)

: profexport (dir, data)

: profexport (dir, name)

: profexport (dir, name, data)

将分析器数据导出为HTML。

将data中的分析数据导出到文件夹dir中的一系列HTML文件中。初始文件将是data/index.html。

如果指定了name，则它必须是一个字符串，其中包含要导出的配置文件的“name”。这个名称包含在HTML中。

输入数据是配置文件(“info”)返回的结构。如果未指定，profexport将使用当前配置文件数据集。

参见:profshow, profexplore, profile。

: profexplore ()

: profexplore (data)

交互式地探索分层剖析器输出。

假设data是profile(“info”)返回的配置文件数据的结构，该命令打开一个交互式提示符，可用于探索调用树。键入help可获得可能的命令列表。如果省略了数据，则调用profile(“info”)并在其位置上使用它。

参见:profile, profshow。

**13.7分析器示例**

下面，我们将给出一个概要分析器会话的简短示例。请参阅分析，以获得详细的分析函数文档。考虑下面的代码:

global N A;

N = 300;

A = rand (N, N);

function xt = timesteps (steps, x0, expM)

global N;

if (steps == 0)

xt = NA (N, 0);

else

xt = NA (N, steps);

x1 = expM \* x0;

xt(:, 1) = x1;

xt(:, 2 : end) = timesteps (steps - 1, x1, expM);

endif

endfunction

function foo ()

global N A;

initial = @(x) sin (x);

x0 = (initial (linspace (0, 2 \* pi, N)))';

expA = expm (A);

xt = timesteps (100, x0, expA);

endfunction

function fib = bar (N)

if (N <= 2)

fib = 1;

else

fib = bar (N - 1) + bar (N - 2);

endif

endfunction

如果我们执行两个主要函数，我们得到:

tic; foo; toc;

⇒ Elapsed time is 2.37338 seconds.

tic; bar (20); toc;

⇒ Elapsed time is 2.04952 seconds.

但这并没有提供多少关于这些时间花在哪里的信息;例如，是对expm的单个调用更昂贵，还是递归时间步进本身更昂贵。为了获得更详细的图像，我们可以使用侧写器。

profile on;

foo;

profile off;

data = profile ("info");

profshow (data, 10);

这将打印出如下的表:

# Function Attr Time (s) Calls

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

7 expm 1.034 1

3 binary \* 0.823 117

41 binary \ 0.188 1

38 binary ^ 0.126 2

43 timesteps R 0.111 101

44 NA 0.029 101

39 binary + 0.024 8

34 norm 0.011 1

40 binary - 0.004 101

33 balance 0.003 1

条目是已执行的单个函数(只有10个最重要的函数)，以及每个函数的一些信息。像' binary \* '这样的条目表示操作符，而其他条目则是普通函数。它们既包括expm这样的内置程序，也包括我们自己的例程(例如时间步骤)。从这个概要文件中，我们可以立即推断出expm占用了最大比例的处理时间，尽管它只被调用一次。第二个昂贵的操作是常规时间步长的矩阵-向量积。

然而，时间并不是概要文件中唯一可用的信息。属性列显示timesteps递归地调用自己。这在本例中可能不那么引人注目(因为无论如何都很清楚)，但在更复杂的设置中可能很有帮助。至于为什么在输出中有“二进制”的问题，我们也可以很容易地阐明这一点。注意，data是一个结构数组(结构数组)，它包含字段FunctionTable。这将存储所示概要文件的原始数据。表第一列中的数字给出了所示函数所在的索引。查找data.FunctionTable(41)给出:

scalar structure containing the fields:

FunctionName = binary \

TotalTime = 0.18765

NumCalls = 1

IsRecursive = 0

Parents = 7

Children = [](1x0)

这里我们再次看到表中的信息，但是增加了Parents和Children字段。它们都是数组，其中包含直接调用有问题的函数(在这种情况下是条目7,expm)或被它调用(没有函数)的函数的索引。因此，反斜杠操作符已在expm内部使用。

现在我们来看看bar。为此，我们启动一个新的分析会话(profile on做这个;旧的数据会在分析器重新启动之前被删除):

profile on;

bar (20);

profile off;

profshow (profile ("info"));

这里给出

# Function Attr Time (s) Calls

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

1 bar R 2.091 13529

2 binary <= 0.062 13529

3 binary - 0.042 13528

4 binary + 0.023 6764

5 profile 0.000 1

8 false 0.000 1

6 nargin 0.000 1

7 binary != 0.000 1

9 \_\_profiler\_enable\_\_ 0.000 1

不出所料，bar也是递归的。在以次优方式递归计算斐波那契数的过程中，它被调用了13529次，而且大部分时间都花在了bar本身上。

最后，假设我们想同时分析foo和bar的执行情况。由于我们已经收集了bar的运行时数据，我们可以重新启动分析器，而不需要清除现有数据，并收集关于foo的缺失统计信息。这是通过:

profile resume;

foo;

profile off;

profshow (profile ("info"), 10);

正如您在下表中看到的，现在我们将两个概要文件混合在一起。

# Function Attr Time (s) Calls

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

1 bar R 2.091 13529

16 expm 1.122 1

12 binary \* 0.798 117

46 binary \ 0.185 1

45 binary ^ 0.124 2

48 timesteps R 0.115 101

2 binary <= 0.062 13529

3 binary - 0.045 13629

4 binary + 0.041 6772

49 NA 0.036 101

**14输入与输出**

Octave支持从提示符或文件中读写数据的几种方式。最简单的数据输入和输出(I/O)函数很容易使用，但只能提供有限的数据处理控制。为了获得更多的控制，Octave还提供了一组模仿C标准库的函数。

**14.1基本输入输出**

**14.1.1终端输出**

由于Octave通常在表达式求值后立即打印表达式的值，因此所有I/O函数中最简单的是一个简单的表达式。例如，下面的表达式将显示' pi '的值

pi

-| ans = 3.1416

只要可以接受将变量名(或“ans”)和值一起打印出来，这种方法就可以很好地工作。要打印变量的值而不打印其名称，请使用函数disp。

format命令对Octave使用disp和通过正常的回显机制打印值的方式提供了一些控制。

: disp (x)

: str = disp (x)

显示x的值。

例如:

disp ("The value of pi is:"), disp (pi)

-| the value of pi is:

-| 3.1416

注意，disp的输出总是以换行符结束。

如果请求输出值，则disp不打印任何内容，并以字符串形式返回格式化的输出。

参见:fdisp。

: str = list\_in\_columns (arg, width, prefix)

返回一个字符串，其中包含arg的元素，列中列出的总最大宽度为width和可选前缀prefix。

参数arg必须是字符串的单元格数组或字符数组。

如果宽度未指定或为空矩阵，或小于或等于零，则使用终端屏幕的宽度。换行符用于在输出字符串中换行。例如:

list\_in\_columns ({"abc", "def", "ghijkl", "mnop", "qrs", "tuv"}, 20)

⇒ abc mnop

def qrs

ghijkl tuv

whos ans

⇒

Variables in the current scope:

Attr Name Size Bytes Class

==== ==== ==== ===== =====

ans 1x37 37 char

Total is 37 elements using 37 bytes

参见:terminal\_size。

: [rows, cols] = terminal\_size ()

: terminal\_size ([rows, cols])

查询或设置终端窗口大小。如果不带参数调用，则返回一个包含终端窗口当前大小的双元素行向量(以字符(行和列)表示)。如果使用整数值的双元素向量调用，则设置终端大小并返回先前的设置。在使用readline进行命令行编辑时，不需要手动设置大小。

参见:list\_in\_columns。

: format

: format options

: format (options)

: [format, formatspacing, uppercase] = format

重置或指定由disp和Octave的正常回显机制产生的输出格式。

这个命令只影响数字的显示，而不影响如何存储或计算数字。要从默认的double类型更改内部表示，请使用转换函数之一，如single, uint8, int64等。任何改变显示有效数字数量的格式选项也将由output\_precision函数反映。

默认情况下，Octave以人类可读的形式显示5位有效数字(对于矩阵，选项' short '，选项' lowercase '和选项' loose '格式)。如果在没有任何选项的情况下调用format，或者指定了选项' default '，则恢复此默认格式。

下表列出了浮点数的有效格式选项。

default

恢复上述的默认格式状态。

short

5位有效数字的定点格式(默认)。

long

定点格式，16位有效数字。

与' short '格式一样，如果使用当前格式无法正确格式化矩阵，Octave将切换到指数' e '格式。

shorte

longe

指数格式。要表示的数字分为尾数和指数(10的幂)。尾数在短格式中有5位有效数字。在长格式中，双精度值以16位有效数字显示，单精度值以8位有效数字显示。例如，使用' short '格式，pi显示为3.1416e+00。可选地，后面的' e '可以拆分为第二个参数。

shortg

longg

根据数字的大小，在定点和指数格式之间进行最佳选择。例如，对于' short '格式，pi .^ [2;4;8;16;32]显示为

ans =

9.8696

97.409

9488.5

9.0032e+07

8.1058e+15

可选地，后面的' g '可以拆分为第二个参数。

shorteng

longeng

与' short '或' long '相同，但使用工程格式显示值，其中指数可被3整除。例如，使用' shorteng '格式，10 \* pi显示为31.416e+00。可选地，后面的' eng '可以拆分为第二个参数。

free

none

以自由格式打印输出，不尝试在小数点上排列矩阵的列。这是一个原始格式，相当于c++代码std::cout <<变量。一般来说，结果是一个包含6位有效数字的表示，其中不必要的精度(例如整数的末尾零)被抑制。复数被格式化为像这样的数字对'(0.60419,0.60709)'而不是像这样的' 0.60419 + 0.60709i '。

以下格式影响所有数值输出(浮点和整数类型)。

"+"

"+" "chars"

plus

plus chars

大于零的矩阵元素打印“+”符号，小于零的元素打印“-”符号，零矩阵元素打印空格。这种格式对于检查大型矩阵的稀疏性结构非常有用。对于非常大的矩阵，绘制稀疏模式的函数spy会更清晰。

可选参数chars指定一个包含3个字符的列表，用于打印大于零、小于零和等于零的值。例如，使用“+”“+-”格式，则矩阵[1,0，-1;- 1,0,1]显示为

ans =

+.-

-.+

bank

以适合货币的格式打印变量(小数点右边有两位数字的固定格式)。只显示变量的实部，因为虚部对货币没有意义。

bit

在内存中打印数字的位表示，总是以最高有效位开头。例如，pi是这样打印的:

0 10000000000 1001001000011111101101010100010001000010110100011000

其中为了清晰起见，添加了空格，以按顺序显示符号位、11位指数和52位尾号。它们一起将圆周率表示为IEEE 754双精度浮点数的标准形式。单精度浮点数是类似的。

native-bit

打印存储在内存中的数字的位表示形式。对于大端机器，这与上面看到的格式位布局相同。对于小端机器，它将以相反的顺序打印字节，尽管字节中的位仍然会在左侧显示最高有效位。

例如，在x86-64上这种格式的pi值为:

00011000 00101101 01000100 01010100 11111011 00100001 00001001 01000000

此处显示为清晰起见添加了空格。将format bit与前面的位串进行比较，以查看分组到字节及其顺序。

hex

与上面的格式位相同，除了为简洁起见，将位一次分成四个十六进制数字。因此圆周率表示为:

400921fb54442d18

native-hex

与上面的格式native-bit相同，只是为了简洁起见，将比特一次分成四个十六进制数字。因此，圆周率在x86-64上表示为:

182d4454fb210940

rat

打印一个有理近似值，即，数值近似值为小整数的比率。例如，使用' rat '格式，圆周率显示为355/113。

以下两个选项影响科学符号和十六进制符号的显示。

lowercase (default)

在科学记数法中，指数字符使用小写字母“e”，表示10-15的十六进制数字使用小写字母“a-f”。

uppercase

科学记数法中的指数字符使用大写' E '，表示10-15的十六进制数字使用大写' A-F '。

以下两个选项影响所有矩阵的显示。

compact

删除列号标签周围和矩阵之间的空白行，生成每页更多数据的更紧凑的输出。

loose (default)

在列号标签的上方和下方以及矩阵之间插入空白行，以每页更少的数据生成更具可读性的输出。

如果使用多个竞争选项调用format，则使用最右边的选项，除了' default '，它将覆盖所有其他选项。如果出现错误，格式保持不变。

如果使用一到三个输出参数调用，并且没有输入，则返回当前格式、格式间距和大写首选项。同时指定输出和输入将产生错误。

参见:fixed\_point\_format, output\_precision, split\_long\_rows, print\_empty\_dimensions, rats。

**14.1.1.1分页屏幕输出**

当交互运行时，Octave通常将所有输出直接发送到命令窗口。然而，当使用Octave的CLI版本时，这可能会产生一个问题，因为在您可以读取它们之前，大量数据将流过。在这种情况下，最好使用诸如less或more之类的分页程序，每次只显示一个屏幕。使用less(和一些版本的more)，您还可以向前和向后扫描，并搜索特定项目。通过命令more on启用寻呼机。

通常，在Octave准备好打印顶级提示符或从标准输入中读取之前，分页器不会显示任何输出(例如，通过使用fscanf或scanf函数)。这意味着如果您要求Octave使用单个命令语句执行大量工作，那么在屏幕上显示任何输出之前可能会有一些延迟。函数fflush可用于强制将输出立即发送到分页(或任何其他流)。

您可以使用pager函数选择要作为分页运行的程序，并使用PAGER\_FLAGS函数配置分页本身。

: more

: more on

: more off

打开或关闭输出分页。

如果没有参数，则会切换当前状态。

当前状态可以通过page\_screen\_output来确定。

参见:page\_screen\_output, page\_output\_immediate, pagager, PAGER\_FLAGS。

: val = PAGER ()

: old\_val = PAGER (new\_val)

: old\_val = PAGER (new\_val, "local")

查询或设置内部变量，该变量指定用于在系统上显示终端输出的程序。

默认值通常是“less”、“more”或“pg”，具体取决于系统上安装的程序。参见安装Octave。

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:PAGER\_FLAGS, page\_output\_immediate, more, page\_screen\_output。

: val = PAGER\_FLAGS ()

: old\_val = PAGER\_FLAGS (new\_val)

: old\_val = PAGER\_FLAGS (new\_val, "local")

查询或设置指定要传递给分页器的选项的内部变量。

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。

参见:PAGER、more、page\_screen\_output、page\_output\_immediate。

: val = page\_screen\_output ()

: old\_val = page\_screen\_output (new\_val) ¶

: old\_val = page\_screen\_output (new\_val, "local")

查询或设置内部变量，该变量控制是否通过寻呼机发送用于超过一页的终端窗口的输出。

这允许您一次查看一个屏幕。一些寻呼机(如less -参见安装Octave)也能够在输出上向后移动。

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:more、page\_output\_immediate、pagager、PAGER\_FLAGS。

: val = page\_output\_immediately ()

: old\_val = page\_output\_immediately (new\_val)

: old\_val = page\_output\_immediately (new\_val, "local")

查询或设置内部变量，该变量控制Octave是否在分页可用时立即将输出发送到分页。

当该值为false时，Octave将缓冲其输出，并等待直到打印提示符之前将其刷新到分页器。这是默认值。

当page\_screen\_output为false时，该变量不起作用。

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:page\_screen\_output, more, PAGER, PAGER\_FLAGS。

: status = fflush (fid)

将输出刷新到文件描述符fid。

fflush在成功时返回0，出错时返回一个与操作系统相关的错误值(Unix为−1)。

编程注意:刷新对于确保所有挂起的输出在其他事件发生之前显示在屏幕上很有用。例如，在调用input之前刷新标准输出流总是一个好主意。

参见:fopen, fclose。

**14.1.2端子输入**

Octave有三个功能，可以方便地提示用户输入。输入和菜单功能通常用于管理与用户的交互对话框，键盘功能通常用于进行简单的调试。

: ans = input (prompt)

: ans = input (prompt, "s")

打印提示并等待用户输入。

例如,

input ("Pick a number, any number! ")

打印提示符

Pick a number, any number!

并等待用户输入一个值。用户输入的字符串被计算为表达式，因此它可以是文字常量、变量名或任何其他有效的Octave代码。

返回参数的数量、大小和类取决于输入的表达式。

如果您只对获取字面值字符串值感兴趣，则可以使用字符串“s”作为第二个参数来调用input。这告诉Octave直接返回用户输入的字符串，而不先求值。

因为可能有输出等待分页器显示，所以总是在调用input之前调用fflush (stdout)是一个好主意。这将确保在提示之前将所有挂起的输出写入屏幕。

参见:yes\_or\_no, kbhit, pause, menu, listdlg。

: choice = menu (title, opt1, …)

: choice = menu (title, {opt1, …})

显示带有标题标题和选项opt1，…的菜单，并等待用户输入。

如果GUI正在运行，则使用listdlg以图形方式显示菜单。否则，将在控制台上打印标题和菜单选项。

Title是一个字符串，选项可以作为单独的字符串或字符串的单元格数组输入。

返回值choice是从1开始计数的用户选择的选项的个数。如果用户终止对话框或做出无效选择，则返回0。

这个函数对于交互式程序很有用。可以传入的选项数量没有限制，但如果在一个屏幕上显示的选项太多，可能会让人感到困惑。

参见: input, listdlg.

: ans = yes\_or\_no ("prompt")

问用户一个是或否的问题。

如果答案是yes，则返回逻辑true;如果答案是no，则返回false。

接受一个参数prompt，它是在提出问题时要显示的字符串。提示符应该以空格结束;yes-or-no将字符串'(是或否)'添加到它。用户必须使用RET确认答案，并且可以编辑答案，直到答案被确认为止。

参见:input。

对于输入，在提示符处可以使用常规的命令行历史记录和编辑功能。

Octave还有一个功能，可以使用户不需要键入回车符就可以从键盘中获取单个字符。

: c = kbhit ()

: c = kbhit (1)

读取键盘上的单个按键。

如果带参数调用(通常为1)，不要等待按键，立即在键盘输入缓冲区中返回下一个按键，如果没有可用的按键，则返回空字符串("")。

例如,

c = kbhit ();

将c设置为键盘上键入的下一个字符。

c = kbhit (1);

与上面的示例相同，但不等待按下键，如果没有可用的键则返回空字符串。

参见: input, pause。

**14.1.3简单文件I/O**

save和load命令允许以各种格式向磁盘文件写入和读取数据。save命令写入文件的默认格式可以通过函数save\_default\_options和save\_precision来控制。

作为一个例子，下面的代码创建了一个3 × 3矩阵，并将其保存到文件' myfile.mat '中。

A = [ 1:3; 4:6; 7:9 ];

save myfile.mat A

一旦将一个或多个变量保存到文件中，就可以使用load命令将它们读入内存。

load myfile.mat

A

-| A =

-|

-| 1 2 3

-| 4 5 6

-| 7 8 9

: save file

: save options file

: save options file v1 v2 …

: save options file -struct STRUCT

: save options file -struct STRUCT f1 f2 …

: save - v1 v2 …

: str = save ("-", "v1", "v2", …)

将命名的变量v1、v2、…保存在文件文件中。

特殊的文件名' - '可以用来作为字符串返回变量的内容。如果没有列出变量名，Octave保存当前作用域中的所有变量。否则，可以使用完整的变量名或模式语法来指定要保存的变量。如果使用-struct修饰符，则标量结构的字段将被保存为具有相应字段名的变量。struct选项可以与特定的字段名f1, f2，…结合使用，只将某些字段写入文件。

下表列出了保存命令的有效选项。修改输出格式的选项覆盖save\_default\_options指定的格式。

如果使用函数形式调用save

save ("-option1", …, "file", "v1", …)

然后，选项、文件和变量名参数(v1，…)必须指定为字符串。

如果以文件名“-”调用，如果nargout为0，则将输出写入stdout，否则以字符串形式返回输出。

-append

追加到目标，而不是覆盖。

-ascii

将矩阵保存在文本文件中，不带标题或任何其他信息。矩阵必须是二维的，并且只将任何复数值的实部写入文件。数字以单精度格式存储，并用空格分隔。-ascii格式的其他选项有

-double

以双精度格式存储数字。

-tabs

用制表符分隔数字。

-binary

将数据保存为Octave的二进制数据格式。

-float-binary

将数据保存为Octave的二进制数据格式，但只使用单个精度。只有当您知道所有要保存的值都可以用单个精度表示时，才使用这种格式。

-hdf5

保存数据为HDF5格式。(HDF5是一种免费的、可移植的二进制格式，由伊利诺伊大学的国家超级计算应用中心开发。)此格式仅在Octave与HDF5库的链接一起构建时可用。

-float-hdf5

以HDF5格式保存数据，但只使用单一精度。只有当您知道所有要保存的值都可以用单个精度表示时，才使用这种格式。

-text

将数据保存为Octave的文本数据格式。(默认)

-v7.3

-V7.3

-7.3

Octave 尚未实现以 MATLAB 的 v7.3 二进制数据格式保存。

-v7

-V7

-7

-mat7-binary

将数据保存为MATLAB的v7二进制数据格式。

-v6

-V6

-6

-mat

-mat-binary

将数据保存为MATLAB v6二进制数据格式。

-v4

-V4

-4

-mat4-binary

将数据保存为MATLAB的v4二进制数据格式。

-zip

-z

使用gzip算法压缩文件。这适用于在Octave之外使用gzip压缩的文件，并且gzip也可以用于向后兼容转换文件。此选项仅在Octave使用到zlib库的链接构建时可用。

要保存的变量列表可以使用包含以下特殊字符的通配符模式(glob模式):

?

匹配任何单个字符

\*

匹配零个或多个字符。

[ list ]

匹配list指定的字符列表。如果第一个字符是!或^，匹配除list指定字符外的所有字符。例如，模式[a-zA-Z]将匹配所有小写和大写字母字符。

当使用-struct修饰符时，也可以在字段名规范中使用通配符(但不能在结构名本身中使用)。

除了使用MATLAB二进制数据文件格式或' -ascii '格式时，保存全局变量还保存了变量的全局状态。如果稍后使用' load '恢复该变量，它将被恢复为全局变量。

例子:

命令

save -binary data a b\*

以Octave的二进制格式保存变量' a '和所有以' b '开头的变量到文件数据中。

参见:load, save\_default\_options, save\_header\_format\_string, save\_precision, dlmread, csvread, fread。

有三个函数可以修改save的行为。

: val = save\_default\_options ()

: old\_val = save\_default\_options (new\_val)

: old\_val = save\_default\_options (new\_val, "local")

查询或设置内部变量，该变量指定保存命令的默认选项，并定义默认格式。

默认值是"-text" (Octave自己的基于文本的文件格式)。有关其他选择，请参阅save命令的文档。

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:save, save\_header\_format\_string, save\_precision。

: val = save\_precision ()

: old\_val = save\_precision (new\_val)

: old\_val = save\_precision (new\_val, "local")

查询或设置内部变量，该变量指定以文本格式保存数据时要保留的位数。

默认值为17，这是无损保存和恢复IEEE-754双值所需的最小值;对于IEEE-754单值，最小值为9。如果需要考虑文件大小，最好选择二进制格式来保存数据，而不是降低保存值的精度。

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:save\_default\_options。

: val = save\_header\_format\_string ()

: old\_val = save\_header\_format\_string (new\_val)

: old\_val = save\_header\_format\_string (new\_val, "local")

查询或设置内部变量，该变量指定在Octave保存的文本格式数据文件开头写入的注释行所使用的格式字符串。

格式字符串传递给strftime，必须以字符“#”开头，并且不包含换行符。如果save\_header\_format\_string的值为空字符串，则在文本格式数据文件中省略头注释。默认值为

"# Created by Octave VERSION, %a %b %d %H:%M:%S %Y %Z <USER@HOST>"

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:strftime, save\_default\_options。

: load file

: load options file

: load options file v1 v2 …

: S = load ("options", "file", "v1", "v2", …)

: load file options

: load file options v1 v2 …

: S = load ("file", "options", "v1", "v2", …)

从文件文件中加载命名变量v1、v2、…。

如果没有指定变量，那么将加载文件中找到的所有变量。与save一样，要提取的变量列表可以是全名，也可以使用模式语法。文件的格式是自动检测的，但可以通过提供适当的选项来覆盖。

如果使用函数形式调用load

load ("-option1", …, "file", "v1", …)

然后，选项、文件和变量名参数(v1，…)必须指定为字符串。

如果从文件中加载未标记为全局的变量，而已经存在同名的全局符号，则将其加载到全局符号表中。此外，如果一个变量在文件中被标记为全局，并且存在一个局部符号，则该局部符号将被移动到全局符号表中，并从文件中给定值。

如果使用单个输出参数调用，Octave将返回数据，而不是在符号表中插入变量。如果数据文件只包含数字(以制表符或空格分隔的列)，则返回一个值矩阵。否则，load返回一个结构体，其成员与文件中变量的名称相对应。

load命令可以读取以Octave的文本和二进制格式以及MATLAB的二进制格式存储的数据。如果使用zlib支持编译，它也可以加载gzip压缩文件。它将自动检测文件类型并从不同的浮点格式进行转换(目前只有IEEE大小端进格式，尽管将来可能会添加其他格式)。

下表列出了有效的加载选项。

-force

此选项可用于向后兼容，但将被忽略。Octave现在使用在文件中找到的同名变量覆盖当前内存中的变量。

-ascii

强制Octave假设文件包含文本格式的数字列，没有任何标题或其他信息。文件中的数据将以单个数字矩阵的形式加载，其中包含从文件名派生的变量名。

-binary

强制Octave假设文件是Octave的二进制格式。

-hdf5

强制 Octave 假定文档为 HDF5 格式。（HDF5 是由伊利诺伊大学国家超级计算应用中心开发的一种免费、可移植的二进制格式。请注意，Octave 只能读取使用 save 或 MATLAB 的 -v7.3 选项（以 HDF5 格式保存）自行创建的 HDF5 文档。仅当 Octave 是使用指向 HDF5 库的链接构建的时，此格式才可用。

-import

此选项可用于向后兼容，但将被忽略。Octave现在可以支持多维HDF数据，如果变量名是无效的Octave标识符，可以自动修改变量名。

-text

强制Octave假设文件是Octave的文本格式。

-v7.3

-V7.3

-7.3

强制 Octave 假定文档采用 MATLAB 的 v7.3 二进制数据格式。由于 v7.3 格式是基于 HDF5 的格式，因此这些文档通常也可以使用“-hdf5”选项打开。请注意，Octave 目前无法以此格式保存。

-v7

-V7

-7

-mat7-binary

强制Octave假设文件是MATLAB的版本7二进制格式。

-v6

-V6

-6

-mat

-mat-binary

强制Octave假设文件是MATLAB的版本6二进制格式。

-v4

-V4

-4

-mat4-binary

强制Octave假设文件是MATLAB的版本4二进制格式。

参见:save, dlmwrite, csvwrite, fwrite。

: str = fileread (filename)

: str = fileread (filename, param, value, …)

读取 filename 的内容并将其作为字符串返回。

param、value 是可选的参数和值对。有效选项包括：

"Encoding"

指定从文档读取时使用的编码。这是有效编码标识符的字符串。默认值为"utf-8"。

参见: fopen, fread, fscanf, importdata, textscan, type.

: fmtstr = native\_float\_format ()

以字符串形式返回本机浮点格式。

可以用类似于将数据写入屏幕的disp函数的方式将数据写入文件。fdisp的工作原理与disp类似，只是它的第一个参数是由fopen创建的文件指针。例如，下面的代码写入数据' myfile.txt '。

fid = fopen ("myfile.txt", "w");

fdisp (fid, "3/8 is ");

fdisp (fid, 3/8);

fclose (fid);

有关如何使用fopen和fclose的详细信息，请参阅打开和关闭文件。

: fdisp (fid, x)

在流fid上显示x的值。

例如:

fdisp (stdout, "The value of pi is:"), fdisp (stdout, pi)

-| the value of pi is:

-| 3.1416

注意，fdisp的输出总是以换行符结束。

参见:disp。

Octave还可以读写矩阵文本文件，如逗号分隔列表。

: dlmwrite (file, M)

: dlmwrite (file, M, delim, r, c)

: dlmwrite (file, M, key, val …)

: dlmwrite (file, M, "-append", …)

: dlmwrite (fid, …)

使用分隔符将数字矩阵M写入文本文件file。

file应该是由fopen给出的文件名或可写文件ID。

参数delim指定用于分隔行上的值的分隔符。如果没有指定分隔符，则使用逗号字符“，”。

r的值指定要添加到文件开头的仅限分隔符的行数。

c的值指定要在每行数据前添加的分隔符的数目。

如果给出了参数"-append"，则追加到文件的末尾。

此外，以下关键字值对可能出现在参数列表的末尾:

"append"

要么“开”要么“关”。参见上面的“-append”。

"delimiter"

见上文。

"newline"

分隔每行的字符。此选项存在三种特殊情况。“unix”改为“\n”，“pc”改为“\r\n”，“mac”改为“\r”。任何其他值都直接用作换行符分隔符。

"roffset"

见上文。

"coffset"

见上文c。

"precision"

写入文件时使用的精度。它可以是一个格式字符串(如fprintf所使用的)，也可以是一些有效数字。

dlmwrite ("file.csv", reshape (1:16, 4, 4));

dlmwrite ("file.tex", a, "delimiter", "&", "newline", "\n")

参见:dlmread, csvread, csvwrite。

: data = dlmread (file)

: data = dlmread (file, sep)

: data = dlmread (file, sep, r0, c0)

: data = dlmread (file, sep, range)

: data = dlmread (…, "emptyvalue", EMPTYVAL)

从文本文件中读取数字数据，该文件在数据值之间使用分隔符sep。

如果没有定义sep，则从文件本身确定字段之间的分隔符。

可选的标量参数r0和c0定义要读取的数据的起始行和列。这些值从0开始索引，即，第一个数据行对应于0的索引。

range参数精确指定读取哪些数据元素。参数的第一种形式是一个4元素向量，包含左上角和右下角[R0,C0,R1,C1]，其中索引从零开始。要指定最后一列(相当于索引时的end)，请使用说明符Inf。或者，使用电子表格样式的表单，如“A2..”可采用“Q15”或“T1:AA5”。最低的字母索引“A”指的是第一列。最低的行索引是1。

File应该是fopen给出的文件名或文件id。在后一种情况下，读取文件直到到达文件的末尾。

"emptyvalue"选项可用于指定用于填充空字段的值。默认值为零。请注意，任何非数字值，如文本，也会被“空值”替换。

参见:csvread, textscan, dlmwrite。

: csvwrite (filename, x)

: csvwrite (filename, x, dlm\_opt1, …)

将数字矩阵x以逗号分隔值(CSV)格式写入文件文件名。

这个函数等价于

dlmwrite (filename, x, ",", dlm\_opt1, …)

任何可选参数都直接传递给dlmwrite(参见dlmwrite)。

参见:csvread, dlmwrite, dlmread。

: x = csvread (filename)

: x = csvread (filename, dlm\_opt1, …)

将逗号分隔值(CSV)文件文件名读入矩阵x。

注意:只能读取包含数字数据的CSV文件。

这个函数等价于

x = dlmread (filename, "," , dlm\_opt1, …)

任何可选参数都直接传递给dlmread(参见dlmread)。

参见:dlmread, textcan, csvwrite, dlmwrite。

格式化的数据也可以从文本文件中读取或写入。

: [a, …] = textread (filename)

: [a, …] = textread (filename, format)

: [a, …] = textread (filename, format, n)

: [a, …] = textread (filename, format, prop1, value1, …)

: [a, …] = textread (filename, format, n, prop1, value1, …)

这个功能已经过时了。使用textcan代替。

从文本文件中读取数据。

根据格式读取和解析文件文件名。该函数的行为类似于strread，只是它通过解析文件而不是字符串来工作。有关详细信息，请参阅strread的文档。

除了strread支持的选项外，该函数还支持另外两个选项:

* "headerlines":文件名的第一个值，被跳过的行数。
* " endoline ":指定单个字符或"\r\n"。如果没有给出值，它将从文件中推断出来。如果设置为""(空字符串)，eol作为分隔符将被忽略。

可选输入n(格式重复计数)指定要使用格式字符串的次数或要读取的行数，以读取时先发生的为准。前者相当于要求数据输出向量的长度为n。请注意，当读取包含引用多行格式字符串的文件时，n应该是要读取的行数，而不是格式字符串使用的行数。

如果格式字符串为空(不仅仅是省略)并且文件只包含数字数据(不包括标题行)，则textread将返回一个矩形矩阵，其中列的数量与文件第一个数据行上的数字字段的数量相匹配。空字段作为零值返回。

例子:

Assume a data file like:

1 a 2 b

3 c 4 d

5 e

[a, b] = textread (f, "%f %s")

returns two columns of data, one with doubles, the other a

cellstr array:

a = [1; 2; 3; 4; 5]

b = {"a"; "b"; "c"; "d"; "e"}

[a, b] = textread (f, "%f %s", 3)

(read data into two culumns, try to use the format string

three times)

returns

a = [1; 2; 3]

b = {"a"; "b"; "c"}

With a data file like:

1

a

2

b

[a, b] = textread (f, "%f %s", 2)

returns a = 1 and b = {"a"}; i.e., the format string is used

only once because the format string refers to 2 lines of the

data file. To obtain 2x1 data output columns, specify N = 4

(number of data lines containing all requested data) rather

than 2.

参见:textscan, load, dlmread, fscanf, strread。

: C = textscan (fid, format)

: C = textscan (fid, format, repeat)

: C = textscan (fid, format, param, value, …)

: C = textscan (fid, format, repeat, param, value, …)

: C = textscan (str, …)

: [C, position, errmsg] = textscan (…)

从文本文件或字符串中读取数据。

与fid相关联的字符串str或文件将根据格式进行读取和解析。该函数是strread和textread的扩展。不同之处包括:从文件或字符串读取的能力、附加选项和附加格式说明符。

输入被解释为单词、分隔符(如空格)和文字的序列。分隔符和空格的字符由选项决定。该格式由散布在文字之间的格式说明符组成。在该格式中，空格在连续字面值之间形成分隔符，否则将被忽略。

输出C是一个单元格数组，其中的列数由格式说明符的数量决定。

输入的第一个单词与格式的第一个说明符匹配，并放置在输出的第一列中;第二个与第二个说明符匹配并放置在第二列中，依此类推。如果单词多于说明符，则重复该过程，直到处理完所有单词或满足repeat设置的限制(见下文)。

字符串格式描述了应该如何解析str中的单词。与fscanf中一样，格式不属于这些说明符之一的任何(非空白)文本都被视为文字。如果两个格式说明符之间有一个文字，那么相同的文字必须出现在匹配的单词之间的输入流中。

以下说明符是有效的:

%f

%f64

%n

该单词被解析为数字并转换为双精度。

%f32

该单词被解析为数字并转换为单个(浮点数)。

%d

%d8

%d16

%d32

%d64

该单词被解析为数字并转换为int8、int16、int32或int64。如果没有指定大小，则使用int32。

%u

%u8

%u16

%u32

%u64

该词被解析为数字，并转换为uint8、uint16、uint32或uint64。如果不指定size，则使用uint32。

%s

该单词被解析为以空格、行尾或选项中指定的分隔符之前的最后一个字符结束的字符串。

%q

该单词被解析为“带引号的字符串”。如果字符串的第一个字符是双引号(")，则该字符串包含匹配双引号之前的所有内容——包括空白、分隔符和行尾字符。如果输入中出现一对连续的双引号，则在输出中用单个双引号替换。例如，输入“He said”“Hello”“”将返回值“He said”Hello”。

%c

读取输入的下一个字符。这包括分隔符、空格和行结束符。

%[…]

%[^…]

在第一种形式中，单词由括号之间的字符组成最长的一段。字符范围可以用连字符指定;例如，%[0-9a-zA-Z]匹配所有字母数字字符(如果底层字符集是ASCII)。由于MATLAB从字面上处理连字符，因此此扩展仅适用于字母数字字符。要在集合中包含' - '，它应该出现在括号的第一个或最后;要包含']'，它应该是第一个字符。如果第一个字符是' ^ '，则该单词由未列出的字符组成。

%N…

对于%s， %c， %d， %f， %n， %u，一个可选的宽度可以指定为%Ns等。其中N为大于1的整数。对于%c，这将导致读取N个字符，而不是单个字符。对于其他说明符，它是读取字符数的上限;普通分隔符可以减少要读取的字符。对于复数，这个极限分别适用于实分量和虚分量。对于%f和%n，格式化指定符，如%n。允许使用Mf，其中M是要考虑的小数点后字符数的上界;跳过后面的数字。例如，指定符%8.2f将把12.345e6读成1.234e7。

%\*…

转换说明符的剩余部分指定的单词将被跳过。

literals

此外，格式可以包含文字字符串;这些在阅读时将被跳过。如果输入字符串与此字面值不匹配，则处理终止。

在第一个输出参数中返回与第一个说明符对应的解析词，其余说明符也同样如此。

默认情况下，如果只有一个输入参数，则format为"%f"。这意味着将数字从输入读入单列向量。如果format显式为空("")，则textscan将返回与输入的第一个数据行上的字段数相匹配的多个列中的数据。这两种方法只有在输入完全是数字时才适用。

例如，字符串

str = "\

Bunny Bugs 5.5\n\

Duck Daffy -7.5e-5\n\

Penguin Tux 6"

可以使用

a = textscan (str, "%s %s %f");

可选的数字参数repeat可用于限制读取的项数:

-1

读取所有字符串或文件直到末尾(默认)。

N

读取，直到出现以下两种情况中的第一个:1)格式已被处理N次，或2)输入的N行已被处理。零(0)是repeat的可接受值。目前，%q、%c和%[…]$转换中的行尾字符不计入行数。这与MATLAB不兼容，将来可能会更改。

文本的行为可以通过属性/值对来改变。识别以下属性:

"BufSize"

这指定要用于内部缓冲区的字节数。在读取大文件时，将该值设置为较大的值可以略微提高速度，特别是在输入包含长字符串的情况下。默认值为4096，如果指定了n，则为依赖于n的值。

"CollectOutput"

值1或true指示textcan在输出单元格数组中连接同一类的连续列。如果值为0或false(默认值)，则输出在不同的列中。

"CommentStyle"

指定输入中被视为注释并将被跳过的部分。value是注释样式，可以是(1)字符串或1x1单元格字符串，跳过它右边的所有内容;(2)两个字符串的单元格数组，跳过第一个和第二个字符串之间的所有内容。注释只在接受空格的地方被解析，而不作为分隔符。

"Delimiter"

如果value是一个字符串，value中的任何字符将被用于将输入分割成单词。如果value是字符串的单元格数组，则数组中的任何字符串都将用于将输入拆分为单词。(默认值=任何空格。)

"EmptyValue"

为非空格分隔的数据中的空数值返回的值。默认为NaN。当数据类型不支持NaN时(例如int32)，则默认值为零。

"EndOfLine"

value可以是空字符，也可以是指定行结束符的一个字符，或者是一对"\r\n" (CRLF)。在后一种情况下，“\r”、“\n”或“\r\n”中的任何一个都被算作(单个)换行符。如果没有给出值，则使用“\r\n”。

"HeaderLines"

跳过fid的第一个值行数。注意，这不是指第一行非注释行，而是指任何类型的第一行。

"MultipleDelimsAsOne"

如果value为非零，则将一系列连续分隔符(中间没有空格)视为单个分隔符。连续的分隔符系列不需要垂直对齐。如果没有此选项，则行尾前的单个分隔符不会导致该行被认为以空值结束，但行首的单个分隔符会导致该行被认为以空值开始。

"TreatAsEmpty"

将value中字符串的单次出现(由分隔符或空格包围)视为缺失值。

"ReturnOnError"

如果设置为数字1或true，则在遇到错误时正常返回，例如尝试使用%f读取字符串。如果设置为0或false，则返回错误且没有数据。

"Whitespace"

value中的任何字符都将被解释为空白并被修剪;空格的默认值是" \b\r\n\t"(注意空格)。除非空格设置为""(空)并且提供了至少一个"%s"格式转换说明符，否则空格总是空白的一部分。

当str或fid中的单词数不匹配格式转换说明符数的精确倍数时，textscan的行为取决于字符串或文件的最后一个字符是否是由endoline选项指定的行结束符:

last character = end-of-line

数据列用空字段、NaN或0(对于整数字段)填充，以便所有列具有相同的长度

last character is not end-of-line

数据列没有填充;Textscan可以返回长度不等的列

第二个输出位置提供了处理停止的位置，从文件或字符串的开头开始，以字符为单位。

参见:dlmread, fscanf, load, strread, textread。

importdata函数能够处理各种各样的数据。

: A = importdata (fname)

: A = importdata (fname, delimiter)

: A = importdata (fname, delimiter, header\_rows)

: [A, delimiter] = importdata (…)

: [A, delimiter, header\_rows] = importdata (…)

从文件fname导入数据。

输入参数:

* fname包含数据的文件名。
* delimiter分隔数据列的字符。使用\t表示制表符。(仅对ASCII文件有效)
* header\_rows数据开始前的头行数。(仅对ASCII文件有效)

支持不同的文件类型:

* ASCII表

使用指定的头行数和指定的分隔符导入ASCII表。

* 图像文件
* MATLAB文件
* 电子表格文件(取决于外部软件)
* WAV文件

**参见:**textcan, dlmread, csvread, load。

导入之后，可能需要在进一步分析之前对数据进行转换。rescale函数可以将数据集移动并规范化到指定的范围。

: B = rescale (A)

: B = rescale (A, l, u)

: B = rescale (…, "inputmin", inmin)

: B = rescale (…, "inputmax", inmax)

将矩阵元素缩放到指定的值范围。

当使用单个矩阵参数a调用时，重新缩放元素以占用区间[0,1]。

可选输入[l, u]将A缩放到下界为l，上界为u的区间。

可选输入“inputmin”将所有小于指定值inmin的元素替换为inmin。类似地，可选输入“inputmax”将所有大于指定值inmax的元素替换为inmax。如果未指定，则从数据本身取最小值和最大值(inmin = min (A(:))和inmax = max (A(:)))。

编程注意事项:应用公式为

B = l + ((- inmin) / (inmax - inmin)) \* (u - l)。

如果输入A是单一的，则输出矩阵B的类是单一的，但如果输入是双精度、整数或逻辑类型，则输出矩阵B的类是double。

**参见:**边界，最小值，最大值。

**14.1.3.1保存意外退出数据**

如果Octave由于某种原因意外退出，它将默认将工作空间中可用的变量保存到当前目录下的一个文件中。默认情况下，该文件名为' octave-workspace '，可以使用load命令加载到内存中。虽然默认行为通常是合理的，但可以通过以下函数进行更改。

: val = crash\_dumps\_octave\_core ()

: old\_val = crash\_dumps\_octave\_core (new\_val)

: old\_val = crash\_dumps\_octave\_core (new\_val, "local")

查询或设置内部变量，该变量控制Octave在崩溃或接收到挂起、终止或类似信号时是否尝试将所有当前变量保存到文件Octave -workspace。

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:octave\_core\_file\_limit, octave\_core\_file\_name, octave\_core\_file\_options。

: val = sighup\_dumps\_octave\_core ()

: old\_val = sighup\_dumps\_octave\_core (new\_val)

: old\_val = sighup\_dumps\_octave\_core (new\_val, "local")

查询或设置内部变量，该变量控制Octave是否在接收到挂起信号时尝试将所有当前变量保存到文件Octave -workspace。

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。

: val = sigquit\_dumps\_octave\_core ()

: old\_val = sigquit\_dumps\_octave\_core (new\_val)

: old\_val = sigquit\_dumps\_octave\_core (new\_val, "local")

查询或设置内部变量，该变量控制Octave是否在接收到退出信号时尝试将所有当前变量保存到文件Octave -workspace。

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。

: val = sigterm\_dumps\_octave\_core ()

: old\_val = sigterm\_dumps\_octave\_core (new\_val)

: old\_val = sigterm\_dumps\_octave\_core (new\_val, "local")

查询或设置内部变量，该变量控制Octave是否在接收到终止信号时尝试将所有当前变量保存到文件Octave -workspace。

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。

: val = octave\_core\_file\_options ()

: old\_val = octave\_core\_file\_options (new\_val)

: old\_val = octave\_core\_file\_options (new\_val, "local")

查询或设置内部变量，该变量指定在Octave终止时用于保存工作空间数据的选项。

octave\_core\_file\_options的值应该遵循与save函数的选项相同的格式。默认值是Octave的二进制格式。

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:crash\_dumps\_octave\_core, octave\_core\_file\_name, octave\_core\_file\_limit。

: val = octave\_core\_file\_limit ()

: old\_val = octave\_core\_file\_limit (new\_val)

: old\_val = octave\_core\_file\_limit (new\_val, "local")

查询或设置内部变量，该变量指定在写入崩溃转储文件时Octave将保存的最大内存量。

该限制以千字节为单位进行测量，并应用于顶级工作空间。崩溃转储文件的名称由octave\_core\_file\_name指定。

如果octave\_core\_file\_options标志指定二进制格式，那么octave\_core\_file\_limit将近似于文件的最大大小。如果使用文本文件格式，则文件可能比限制大得多。默认值是-1(无限制)。

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:crash\_dumps\_octave\_core, octave\_core\_file\_name, octave\_core\_file\_options。

: val = octave\_core\_file\_name () ¶

: old\_val = octave\_core\_file\_name (new\_val)

: old\_val = octave\_core\_file\_name (new\_val, "local")

查询或设置内部变量，该变量指定用于在Octave中止时保存来自顶级工作空间的数据的文件名。

默认值是“octave-workspace”。

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:crash\_dumps\_octave\_core, octave\_core\_file\_name, octave\_core\_file\_options。

**14.2 c风格的I/O函数**

Octave的C风格输入和输出函数提供了C编程语言标准I/O库的大部分功能。然而，一些输入函数的参数列表略有不同，因为Octave无法通过引用传递参数。

在下面的示例中，file指的是文件名，fid指的是fopen返回的整数文件号。

有三个文件总是可用的。虽然这些文件可以使用它们对应的数字文件id来访问，但您应该始终使用下表中给出的符号名称，因为这将使您的程序更容易理解。

: fid = stdin ()

返回与标准输入流相对应的数值。

当以交互方式使用Octave时，通过命令行编辑功能过滤stdin。

参见:stdout, stderr。

: fid = stdout ()

返回与标准输出流对应的数值。

写入标准输出的数据可以通过寻呼机进行过滤。

参见:stdin, stderr, page\_screen\_output。

: fid = stderr ()

返回与标准错误流对应的数值。

即使打开了分页，标准错误也不会发送到分页器。它对错误消息和提示非常有用。

参见:stdin, stdout。

**14.2.1打开和关闭文件**

当从文件中读取数据时，必须首先打开文件以供读取，同样地，在向文件写入数据时也是如此。函数的作用是:返回一个指向已打开文件的指针，该文件已准备好被读取或写入。一旦所有数据都从打开的文件中读取或写入，就应该关闭该文件。fclose函数就是这样做的。下面的代码演示了写入文件的基本模式，但是在读取文件时使用了非常相似的模式。

filename = "myfile.txt";

fid = fopen (filename, "w");

# Do the actual I/O here…

fclose (fid);

: fid = fopen (name)

: fid = fopen (name, mode)

: fid = fopen (name, mode, arch)

: fid = fopen (name, mode, arch, encoding)

: [fid, msg] = fopen (…)

: fid\_list = fopen ("all")

: [file, mode, arch, encoding] = fopen (fid)

为低级I/O打开文件或查询打开的文件和文件描述符。

fopen函数的第一种形式使用指定的模式(读写、只读等)、架构解释(IEEE大端序、IEEE小端序等)和文件编码打开命名文件，并返回一个整数值，该值可用于稍后引用该文件。如果出现错误，则将fid设置为−1,msg包含相应的系统错误消息。模式是一个或两个字符串，用于指定文件是要打开以进行读、写还是同时打开。编码是一个具有有效编码标识符的字符串。当从文件中读取或写入字符串时使用这种编码。缺省情况下，是UTF-8。

fopen函数的第二种形式返回一个文件id向量，该向量对应于所有当前打开的文件，不包括stdin、stdout和stderr流。

fopen函数的第三种形式返回给定文件id的打开文件的信息。

例如,

myfile = fopen ("splat.dat", "r", "ieee-le");

打开文件splat.dat进行读取。如果有必要，将读取二进制数值，假设它们以IEEE格式存储，首先是最低有效位，然后转换为本机表示。

打开一个已经打开的文件只是再次打开它，并返回一个单独的文件id。多次打开一个文件并不是错误，但是通过几个不同的文件id写入同一个文件可能会产生意想不到的结果。

mode的取值有

‘r’ (default)

打开文件进行阅读。

‘w’

打开文件进行写入。之前的内容将被丢弃。

‘a’

打开或创建一个文件，以便在文件末尾进行写入。

‘r+’

打开现有文件进行读写。

‘w+’

打开一个文件进行读写。之前的内容将被丢弃。

‘a+’

打开或创建一个文件，以便在文件末尾进行读写。

在模式字符串后添加“t”以文本模式打开文件，或在模式字符串后添加“b”以二进制模式打开文件。在Windows系统上，文本模式读写自动将换行转换为系统的适当行结束字符(Windows上的回车换行)。当未指定模式时，默认为二进制。

另外，您可以在模式字符串后面附加一个"z"来打开一个gzip文件进行读写。要使此操作成功，还必须以二进制模式打开该文件。

参数arch是一个字符串，指定文件的默认数据格式。arch的有效值为:

"native" or "n" (default)

当前机器的格式。

"ieee-be" or "b"

IEEE大端格式。

"ieee-le" or "l"

IEEE小端序格式。

当打开一个不存在的新文件时，权限将被设置为0666 - umask。

兼容性注意:Octave使用缓冲I/O打开文件。小的写操作会累积起来，直到内部缓冲区被填满，然后在单个操作中写入所有内容。这是非常有效的，并提高了性能。然而，MATLAB使用刷新的I/O打开文件，其中每个写操作都立即执行。如果必须在数据写入之后立即执行写操作，那么应该在写入之后调用fflush来刷新内部缓冲区。

参见:fclose, fgets, fget1, fscanf, fread, fputs, fdisp, fprintf, fwrite, fskipl, fseek, frewind, ftell, fof, terror, fclear, fflush, report, umask。

: status = fclose (fid)

: status = fclose ("all")

关闭文件描述符fid指定的文件。

如果成功，fclose返回0，否则返回-1。fclose调用的第二种形式关闭所有打开的文件，除了stdin、stdout、stderr和与gnuplot相关的任何fid。

参见:fopen, fflush, report。

: tf = is\_valid\_file\_id (fid)

如果fid指向打开的文件，则返回true。

参见:报告，打开。

**14.2.2简单输出**

打开文件以便写入字符串后，可以使用fputs函数将字符串写入文件。下面的例子展示了如何将字符串' Free Software is needed for Free Science '写入文件' Free .txt '。

filename = "free.txt";

fid = fopen (filename, "w");

fputs (fid, "Free Software is needed for Free Science");

fclose (fid);

: status = fputs (fid, string)

用文件描述符fid将字符串string写入文件。

将字符串写入文件，不需要额外的格式化。使用fdisp来自动添加适合本地计算机的换行符。

可选输出状态为0表示成功，或-1表示遇到错误。

参见:fdisp, fprintf, fwrite, fopen。

有一个非常类似于fputs的函数可用于向屏幕写入数据。puts函数的工作原理与fputs类似，只是它不接受文件指针作为输入。

: status = puts (string)

将字符串不带格式地写入标准输出。

将字符串逐字写入标准输出。使用disp自动追加适合本地计算机的换行符。

可选输出状态为0表示成功，或-1表示遇到错误。

参见:输出，输出。

**14.2.3面向行的输入**

要从文件中读取，必须使用fopen打开文件以供读取。然后可以使用fgetl从文件中读取一行，如下面的代码所示

fid = fopen ("free.txt");

txt = fgetl (fid)

-| Free Software is needed for Free Science

fclose (fid);

当然，这假设文件“Free .txt”存在，并且包含一行“Free Software is need for Free Science”。

: str = fgetl (fid)

: str = fgetl (fid, len)

从文件中读取字符，在读取换行符、EOF或len字符后停止。

读取的字符(不包括可能的尾随换行符)将作为字符串返回。

如果省略len, fgetl将读取到下一个换行符。

如果没有字符可读，fgetl返回−1。

要读取一行并返回结束换行符，请参见fgets。

参见:fgets, fscanf, read, fopen。

: str = fgets (fid)

: str = fgets (fid, len)

从文件中读取字符，在读取换行符、EOF或len字符后停止。

读取的字符，包括可能的尾随换行符，将作为字符串返回。

如果省略len，则fgets读取到下一个换行符。

如果没有字符可读，fgets返回−1。

要读取一行并放弃结束换行符，请参见fgetl。

参见:fputs, fgetl, fscanf, read, fopen。

: nlines = fskipl (fid)

: nlines = fskipl (fid, count)

: nlines = fskipl (fid, Inf)

从文件描述符fid指定的文件中读取并跳过计数行。

Fskipl丢弃字符，直到遇到行结束符的精确次数，或者直到找到文件结束符标记。

如果省略count，则默认为1。count也可以是Inf，在这种情况下，行被跳过，直到文件的末尾。此表单适用于计算文件中的行数。

返回跳过的行数(遇到的行尾序列)。

参见:fget1, fgets, fscanf, fopen。

**14.2.4格式化输出**

本节介绍如何调用printf和相关函数。

以下函数可用于格式化输出。它们是在C语言同名函数的基础上建模的，但是为了提高打印向量和矩阵值的性能，它们对格式模板的解释不同。

实现注意:为了与MATLAB兼容，即使模板字符串用单引号定义，模板字符串中的转义序列(例如，"\n" => newline)也会展开。

: printf (template, …)

: numbytes = printf (…)

在模板字符串模板的控制下，将可选参数打印到流标准输出，并返回打印的字符数。

请参阅GNU Octave手册的格式化输出部分，以获得模板字符串语法的完整描述。

可选输出numbytes返回打印的字节数。

实现注意:为了与MATLAB兼容，即使模板字符串用单引号定义，模板字符串中的转义序列(例如，"\n" => newline)也会展开。

参见:fprint, sprintf, scanf。

: fprintf (fid, template, …)

: fprintf (template, …)

: numbytes = fprintf (…)

这个函数相当于printf，不同之处在于输出被写入文件描述符fid而不是标准输出。

如果省略fid，则输出被写入stdout，使该函数完全等同于printf。

可选输出numbytes返回写入文件的字节数。

实现注意:为了与MATLAB兼容，即使模板字符串用单引号定义，模板字符串中的转义序列(例如，"\n" => newline)也会展开。

参见:fputs, fdisp, fwrite, fscanf, printf, sprintf, fopen。

: str = sprintf (template, …)

这类似于printf，不同之处在于输出以字符串形式返回。

C库函数要求您提供适当大小的字符串作为参数，与此不同的是，Octave的sprintf函数返回字符串，并自动调整大小以容纳转换后的所有项。

实现注意:为了与MATLAB兼容，即使模板字符串用单引号定义，模板字符串中的转义序列(例如，"\n" => newline)也会展开。

参见:printf, fprintf, sscanf。

printf函数可用于打印任意数量的参数。在调用中提供的模板字符串参数不仅提供了关于附加参数数量的信息，还提供了关于它们的类型以及应该使用什么样式来打印它们的信息。

模板字符串中的普通字符只是按原样写入输出流，而模板中由' % '字符引入的转换规范会导致对后续参数进行格式化并写入输出流。例如,

pct = 37;

filename = "foo.txt";

printf ("Processed %d%% of '%s'.\nPlease be patient.\n",

pct, filename);

产生如下输出

Processed 37% of 'foo.txt'.

Please be patient.

这个例子展示了如何使用' %d '转换来指定一个标量参数应该以十进制记数方式打印，' %s '转换来指定打印一个字符串参数，' %% '转换来打印一个文字' % '字符。

还有将整数参数打印为八进制、十进制或十六进制基数(分别为' %o '、' %u '或' %x ')的无符号值的转换;或者作为字符值(' %c ')。

浮点数可以使用' %f '转换以正常的定点记数法打印，也可以使用' %e '转换以指数记数法打印。' %g '转换使用' %e '或' %f '格式，具体取决于哪种格式更适合特定数字的大小。

通过在“%”和指示应用哪种转换的字符之间编写修饰符，可以更精确地控制格式。这稍微改变了转换的普通行为。例如，大多数转换规范允许您指定最小字段宽度和一个标志，该标志指示您是希望结果在字段内左对齐还是右对齐。

允许的特定标志和修饰符及其解释因特定转换而异。在下面几节中将对它们进行更详细的描述。

**14.2.5矩阵的输出转换**

当给定一个矩阵值时，Octave的格式化输出函数循环遍历格式模板，直到矩阵中的所有值都被打印出来。例如:

printf ("%4.2f %10.2e %8.4g\n", hilb (3));

-| 1.00 5.00e-01 0.3333

-| 0.50 3.33e-01 0.25

-| 0.33 2.50e-01 0.2

如果要在一次调用中打印多个值，那么当从一个值移动到下一个值时，输出函数不会返回到格式模板的开头。如果矩阵中的元素数量不是格式模板中转换次数的精确倍数，则可能导致输出混乱。例如:

printf ("%4.2f %10.2e %8.4g\n", [1, 2], [3, 4]);

-| 1.00 2.00e+00 3

-| 4.00

如果这不是你想要的，使用一系列的电话而不是仅仅一个。

**14.2.6输出转换语法**

本节详细介绍可以出现在printf模板字符串中的转换规范的精确语法。

模板字符串中不属于转换规范的字符将按原样打印到输出流。

printf模板字符串中的转换规范有一般形式:

% flags width [ . precision ] type conversion

例如，在转换说明符' %-10.8ld '中，' - '是一个标志，' 10 '指定字段宽度，精度为' 8 '，字母' l '是类型修饰符，' d '指定转换样式。(这个特定的类型说明符说要以十进制记数法打印数字参数，在至少10个字符宽的字段中至少有8位左对齐的数字。)

更详细地说，输出转换规范由一个初始' % '字符按顺序组成:

* 修改转换规范的正常行为的零个或多个标志字符。

指定最小字段宽度的可选十进制整数。如果正常转换产生的字符少于此值，则用空格填充字段以达到指定的宽度。这是一个最小值;如果普通转换产生的字符多于此数，则不会截断该字段。通常，输出在字段内是右对齐的。

您还可以指定字段宽度为“\*”。这意味着参数列表中的下一个参数(在要打印的实际值之前)被用作字段宽度。该值四舍五入到最接近的整数。如果值为负，这意味着设置' - '标志(见下文)并使用绝对值作为字段宽度。

* 可选精度，用于指定要为数字转换写入的位数。如果指定了精度，则它由句点(' . ')可选地后跟一个十进制整数(如果省略则默认为零)组成。

您还可以指定' \* '的精度。这意味着参数列表中的下一个参数(在要打印的实际值之前)被用作精度。该值必须为整数，如果为负数则忽略。

* 可选的类型修饰符字符。这个字符被Octave的printf函数忽略，但被识别为提供与C语言printf的兼容性。
* 指定要应用的转换的字符。

在不同的转换说明符之间，允许的确切选项及其解释方式各不相同。有关它们使用的特定选项的信息，请参阅各个转换的描述。

**14.2.7输出转换表**

下面是一个表格，总结了所有不同的转换:

‘%d’, ‘%i’

将整数打印为带符号的十进制数。有关详细信息，请参阅整数转换。' %d '和' %i '是输出的同义词，但与scanf一起用于输入时是不同的(参见输入转换表)。

‘%o’

将整数打印为无符号八进制数。有关详细信息，请参阅整数转换。

‘%u’

将整数打印为无符号十进制数。有关详细信息，请参阅整数转换。

‘%x’, ‘%X’

将整数打印为无符号十六进制数。' %x '使用小写字母，' %x '使用大写字母。有关详细信息，请参阅整数转换。

‘%f’

以普通(定点)记数法打印浮点数。有关详细信息，请参阅浮点转换。

‘%e’, ‘%E’

以指数表示法打印浮点数。“%e”使用小写字母，“%e”使用大写字母。有关详细信息，请参阅浮点转换。

‘%g’, ‘%G’

以普通(定点)或指数表示法打印浮点数，以其大小更合适的方式打印。' %g '使用小写字母，' %g '使用大写字母。有关详细信息，请参阅浮点转换。

‘%c’

打印单个字符。参见其他输出转换。

‘%s’

打印字符串。参见其他输出转换。

‘%%’

打印文字“%”字符。参见其他输出转换。

如果转换规范的语法无效，将发生不可预测的事情，因此不要这样做。特别是，MATLAB允许没有后续转换字符的裸百分比符号' % '。如果看到这样的代码，Octave将发出一个错误并停止。当不能保证要处理的字符串变量没有潜在的格式代码时，最好使用任何printf函数的两个参数形式，并将格式字符串设置为%s。或者，对于不需要与MATLAB向后兼容的代码，可以使用Octave函数puts或disp。

printf (strvar); # Unsafe if strvar contains format codes

printf ("%s", strvar); # Safe

puts (strvar); # Safe

如果没有提供足够的函数参数来为模板字符串中的所有转换规范提供值，或者参数的类型不正确，则结果是不可预测的。如果您提供的参数多于转换规范，则会忽略额外的参数值;这有时是有用的。

**14.2.8整数转换**

本节描述' %d '、' %i '、' %o '、' %u '、' %x '和' %x '转换规范的选项。这些转换以各种格式打印整数。

' %d '和' %i '转换规范都将数字参数打印为有符号十进制数;而' %o '、' %u '和' %x '分别将参数打印为无符号八进制、十进制或十六进制数。' %X '转换规范就像' %X '，除了它使用字符' ABCDEF '而不是' ABCDEF '作为数字。

以下标志是有意义的:

‘-’

在字段中左对齐结果(而不是正常的右对齐)。

‘+’

对于带符号的' %d '和' %i '转换，如果值为正数，则打印加号。

‘ ’

对于带符号的' %d '和' %i '转换，如果结果不以加号或减号开头，则用空格字符作为前缀。由于' + '标志确保结果包含一个符号，因此如果同时提供这两个标志，则忽略此标志。

‘#’

对于' %o '转换，这将强制前导数字为' 0 '，就像通过增加精度一样。对于' %x '或' %x '，这将在结果前分别添加一个' 0x '或' 0x '前缀。这对' %d '， ' %i '或' %u '转换没有任何用处。

‘0’

用零而不是空格填充字段。0放在任何符号或基数的指示之后。如果还指定了' - '标志，或者指定了精度，则忽略此标志。

如果提供了精度，则指定要出现的最小位数;必要时产生前导零。如果不指定精度，则根据需要打印数字。如果将零值转换为显式精度为零的值，则根本不会生成任何字符。

**14.2.9浮点转换**

本节讨论浮点数的转换规范:' %f '、' %e '、' %e '、' %g '和' %g '转换。

' %f '转换以定点记数法打印其参数，产生[-]ddd形式的输出。其中，小数点后的位数由您指定的精度控制。

' %e '转换以指数表示法打印它的参数，产生形式为[-]d.ddde[+|-]dd的输出。同样，小数点后的位数由精度控制。指数总是至少包含两位数字。' %E '转换是类似的，但指数用字母' E '而不是' E '标记。

如果指数小于-4或大于或等于精度，' %g '和' %g '转换将以' %e '或' %e '的样式(分别)打印参数;否则他们使用' %f '样式。从结果的小数部分删除尾随的零，并且只有在后跟数字时才会出现小数点字符。

以下标志可用于修改行为:

‘-’

左对齐字段中的结果。通常结果是右对齐的。

‘+’

总是在结果中包含一个正号或负号。

‘ ’

如果结果不是以加号或减号开头，则用空格作为前缀。由于' + '标志确保结果包含一个符号，因此如果同时提供这两个标志，则忽略此标志。

‘#’

指定结果应始终包含小数点，即使其后没有数字。对于' %g '和' %g '转换，这还强制将小数点后的尾随零留在原本要删除的位置。

‘0’

用零填充字段，而不是空格;0放在任何符号后面。如果还指定了' - '标志，则忽略此标志。

精度指定了“% f”、“% e”和“% e”转换的小数点字符。对于这些转换,默认精度为6。如果精度显式为0,则完全抑制小数点的字符。对于“% g”和“% g”转换,精度指定打印有多少位数数字。显著的数字是十进制前的第一个数字,然后是所有的数字。如果精度为0或不指定“% g”或“% g”,它被视为1的值。如果打印的值不能准确地表达在指定的数字中,那么值就会变成最接近的数字。

**14.2.10其他输出转换**

本节描述printf的各种转换。

' %c '转换打印单个字符。' - '标志可用于指定字段中的左对齐，但没有定义其他标志，也不能给出精度或类型修饰符。例如:

printf ("%c%c%c%c%c", "h", "e", "l", "l", "o");

输出“hello”。

' %s '转换打印一个字符串。对应的参数必须是字符串。可以指定精度来指示要写入的最大字符数;否则，字符串中的字符(不包括结束的null字符)将被写入输出流。' - '标志可用于指定字段中的左对齐，但没有为此转换定义其他标志或类型修饰符。例如:

printf ("%3s%-6s", "no", "where");

打印' nowhere '(注意前后的空格)。

**14.2.11格式化输入**

Octave提供scanf、fscanf和sscanf函数来读取格式化的输入。每个函数都有两种形式。一个可以用来从文件中提取数据向量，另一个更像c。

: [val, count, errmsg] = fscanf (fid, template, size)

: [v1, v2, …, count, errmsg] = fscanf (fid, template, "C")

在第一种形式中，根据模板从fid中读取，并在矩阵val中返回结果。

可选参数size指定要读取的数据量，可以是其中之一

Inf

尽可能多地读取，返回一个列向量。

nr

读取最多nr个元素，返回一个列向量。

[nr, Inf]

读取尽可能多，返回一个矩阵与nr行。如果读取的元素数不是nr的精确倍数，则最后一列用零填充。

[nr, nc]

读取最多nr \* nc个元素，返回具有nr行的矩阵。如果读取的元素数不是nr的精确倍数，则最后一列用零填充。

如果省略size，则假定值为Inf。

如果模板只指定字符转换，则返回字符串。

成功读取的条目数以count返回。

如果发生错误，errmsg包含一个系统相关的错误消息。

在第二种形式中，根据模板从fid读取，模板中的每个转换说明符对应一个标量返回值。这种形式更“像c”，也与以前版本的Octave兼容。成功转换的次数以count返回

请参阅GNU Octave手册的格式化输入部分，以获得模板字符串语法的完整描述。

参见:fgets, fgetl, read, scanf, sscanf, fopen。

: [val, count, errmsg] = scanf (template, size)

: [v1, v2, …, count, errmsg] = scanf (template, "C")

这相当于用fid = stdin调用fscanf。

目前在交互式程序中调用scanf是没有用的。

参见:fscanf, sscanf, printf。

: [val, count, errmsg, pos] = sscanf (string, template, size)

: [v1, v2, …, count, errmsg] = sscanf (string, template, "C")

这类似于fscanf，只是字符是从字符串string而不是流中获取的。

到达字符串的末尾被视为文件结束条件。除了fscanf返回的值之外，下一个要读取的字符的索引在pos中返回。

参见:fscanf, scanf, sprintf。

对scanf的调用表面上类似于对printf的调用，因为在模板字符串的控制下读取任意参数。虽然模板中转换规范的语法与printf非常相似，但模板的解释更倾向于自由格式输入和简单的模式匹配，而不是固定字段格式。例如，大多数scanf转换会跳过输入文件中的任意数量的“空白”(包括空格、制表符和换行符)，并且数字输入转换不像相应的输出转换那样有精度的概念。通常，期望模板中的非空白字符与输入流中的字符完全匹配。例如，请注意，当使用混合数字和字符串输出类型时，sscanf解析字符串和空白的方式不同:

teststr = "1 is a lonely number";

sscanf (teststr, "%s is a %s")

⇒ 1lonelynumber

sscanf (teststr, "%g is a %s")

⇒

1

108

111

110

101

108

121

[a, b, c] = sscanf ("1 is a lonely number", "%g is a %s %s", "C")

⇒ a = 1

⇒ b = lonely

⇒ c = number

当发生匹配失败时，scanf立即返回，将第一个不匹配的字符作为下一个要从流中读取的字符，并且scanf返回成功转换的所有项。

格式化输入函数的使用频率不如格式化输出函数高。部分原因是，正确使用它们需要一些小心。另一个原因是很难从匹配错误中恢复。

在模板字符串中允许的特定标志和修饰符以及它们的解释都将在以下部分中更详细地描述。

**14.2.12输入转换语法**

scanf模板字符串是一个包含普通多字节字符的字符串，其中散布着以' % '开头的转换规范。

模板中的任何空白字符都会导致读取和丢弃输入流中的任意数量的空白字符。匹配的空白字符不必与模板字符串中出现的空白字符完全相同。例如，在模板中写入'，'来识别逗号，逗号前后都有可选的空白。

模板字符串中不属于转换规范的其他字符必须与输入流中的字符完全匹配;如果不是这样，就会发生匹配失败。

scanf模板字符串中的转换规范有一般形式:

% flags width type conversion

更详细地说，输入转换规范由初始' % '字符按顺序组成:

* 一个可选的标志字符“\*”，表示忽略为该规范读取的文本。当scanf找到使用此标志的转换规范时，它按照转换规范的其余部分的指示读取输入，但它丢弃该输入，不返回任何值，也不增加成功赋值的计数。
* 指定最大字段宽度的可选十进制整数。当达到这个最大值或发现不匹配的字符时，从输入流中读取字符停止，以先发生的为准。大多数转换都会丢弃初始空白字符，并且这些丢弃的字符不会计入最大字段宽度。不丢弃初始空格的转换有明确的文档记录。
* 可选的类型修饰符字符。此字符被Octave的scanf函数忽略，但被识别为与C语言scanf兼容。
* 指定要应用的转换的字符。

在不同的转换说明符之间，允许的确切选项及其解释方式各不相同。有关它们允许的特定选项的信息，请参阅输入转换表中对各个转换的描述。

**14.2.13输入转换表**

下面是一个表格，总结了各种转换规格:

‘%d’

匹配以十进制形式写入的可选带符号整数。请参阅数字输入转换。

‘%i’

匹配C语言为指定整型常量而定义的任意格式的可选带符号整数。请参阅数字输入转换。

‘%o’

匹配以八进制形式书写的无符号整数。请参阅数字输入转换。

‘%u’

匹配以十进制基数书写的无符号整数。请参阅数字输入转换。

‘%x’, ‘%X’

匹配以十六进制写的无符号整数。请参阅数字输入转换。

‘%e’, ‘%f’, ‘%g’, ‘%E’, ‘%G’

匹配可选的带符号浮点数。请参阅数字输入转换。

‘%s’

匹配只包含非空白字符的字符串。请参阅字符串输入转换。

‘%c’

匹配一个或多个字符的字符串;读取的字符数由为转换给定的最大字段宽度控制。请参阅字符串输入转换。

‘%%’

这将匹配输入流中的文字' % '字符。没有使用相应的参数。

如果转换规范的语法无效，则未定义该行为。如果没有提供足够的函数参数来为模板字符串中执行赋值的所有转换规范提供地址，或者如果参数的类型不正确，则行为也是未定义的。另一方面，额外的参数会被忽略。

**14.2.14数字输入转换**

本节描述读取数值的scanf转换。

' %d '转换匹配十进制的可选带符号整数。

' %i '转换匹配C语言为指定整数常量定义的任何格式的可选带符号整数。

例如，任何字符串' 10 '、' 0xa '或' 012 '都可以在' %i '转换下作为整数读入。每个选项都指定一个十进制值为10的数字。

' %o '、' %u '和' %x '转换分别匹配八进制、十进制和十六进制的无符号整数。

' %X '转换与' %X '转换相同。它们都允许将大写或小写字母用作数字。

默认情况下，整数被读取为32位数字。使用' h '修饰符时，使用16位整数，使用' l '修饰符时，使用64位整数。

' %e '、' %f '、' %g '、' %e '和' %g '转换匹配可选的带符号浮点数。所有五个转换规范的行为相同，并且将读取任何浮点显示样式的数值。

**14.2.15字符串输入转换**

本节描述读取字符串和字符值的scanf输入转换:' %s '和' %c '。

' %c '转换是最简单的:它总是匹配固定数量的字符。最大字段with表示要读取多少字符;如果不指定最大值，则默认为1。此转换不会跳过初始空白字符。它精确地读取接下来的n个字符，如果不能得到那么多，它就会失败。

' %s '转换匹配一串非空白字符。它跳过并丢弃初始空白，但在读取内容后遇到更多空白时停止。

例如，读取输入:

hello, world

使用转换' %10c '产生"hello, wo"，但使用转换' %10s '读取相同的输入产生"hello，"。

**14.2.16二进制I/O**

Octave可以使用函数fread和fwrite读写二进制数据，这两个函数是按照具有相同名称的标准C函数的模式设计的。它们能够自动交换整数数据的字节顺序，并在读取数据时在支持的浮点格式之间进行转换。

: val = fread (fid)

: val = fread (fid, size)

: val = fread (fid, size, precision)

: val = fread (fid, size, precision, skip)

: val = fread (fid, size, precision, skip, arch)

: [val, count] = fread (…)

从文件描述符fid指定的文件中读取二进制数据。

可选参数size指定要读取的数据量，可以是其中之一

Inf

尽可能多地读取，返回一个列向量。

nr

读取最多nr个元素，返回一个列向量。

[nr, Inf]

读取尽可能多，返回一个矩阵与nr行。如果读取的元素数不是nr的精确倍数，则最后一列用零填充。

[nr, nc]

读取最多nr \* nc个元素，返回具有nr行的矩阵。如果读取的元素数不是nr的精确倍数，则最后一列用零填充。

如果省略size，则假定值为Inf。

可选参数precision是一个字符串，指定要读取的数据类型，可以是其中之一

"uint8" (default)

8位无符号整数。

"int8"

"integer\*1"

8位有符号整数。

"uint16"

"ushort"

"unsigned short"

16位无符号整型

"int16"

"integer\*2"

"short"

16位有符号整数。

"uint"

"uint32"

"unsigned int"

"ulong"

"unsigned long"

32位无符号整数

"int"

"int32"

"integer\*4"

"long"

32-bit signed integer.

"uint64"

64位无符号整数。

"int64"

"integer\*8"

64位带符号整数。

"single"

"float"

"float32"

"real\*4"

32位浮点数。

"double"

"float64"

"real\*8"

64位浮点数。

"char"

"char\*1"

8位单字符。

"uchar"

"unsigned char"

8位无符号字符。

"schar"

"signed char"

8位带符号字符。

默认精度为“uint8”。

precision参数还可以指定可选的重复计数。例如，' 32\*single '导致read读取32个单精度浮点数块。读入块与skip参数结合使用很有用。

precision参数还可以指定类型转换。例如，' int16=>int32 '导致read读取16位整数值并返回一个32位整数值数组。默认情况下，fread返回双精度数组。特殊形式' \*TYPE '是' TYPE=>TYPE '的简写。

转换计数和重复计数可以合并。例如，规范' 32\*single=>single '导致fread读取单精度浮点值块并返回单精度值数组，而不是默认的双精度值数组。

可选参数skip指定读取每个元素(或元素块)后要跳过的字节数。如果未指定，则假定值为0。如果最后的块读取未完成，则省略最后的跳过。例如,

fread (f, 10, "3\*single=>single", 8)

将省略最后的8字节跳过，因为最后一次读取将不是一个完整的3个值块。

可选参数arch是一个字符串，指定文件的数据格式。有效值是

"native" or "n"

当前机器的格式。

"ieee-be" or "b"

IEEE大端码。

"ieee-le" or "l"

IEEE小端序。

如果没有给定arch，则使用在调用创建文件描述符的fopen中使用的值。否则，用fread指定的值将覆盖fopen的值并确定数据格式。

输出参数val包含从文件中读取的数据。

可选的返回值count包含读取的元素数。

参见:fwrite, fgets, fget1, fscanf, fopen。

: count = fwrite (fid, data)

: count = fwrite (fid, data, precision)

: count = fwrite (fid, data, precision, skip)

: count = fwrite (fid, data, precision, skip, arch)

将二进制形式的数据写入文件描述符fid指定的文件。

参数data是要写入文件的值的矩阵。这些值是按列主顺序提取的。

其余的参数precision、skip和arch都是可选的，它们的解释方式与对read的描述一样。

输出计数是成功写入的数据项的数量。

编程注意:如果数据中的值太大而不能满足指定的精度，则fwrite的行为是未定义的。

另见:读取、输出、fprintf、fopen。

**14.2.17临时文件**

有时需要将数据写入只是临时的文件。当从Octave内部启动的外部程序需要访问数据时，这是最常用的。当Octave退出时，所有临时文件将被删除，因此无需手动执行此步骤。

: [fid, name, msg] = mkstemp ("template")

: [fid, name, msg] = mkstemp ("template", delete)

返回与从模板创建的具有唯一名称的新临时文件对应的文件描述符fid。

模板的最后六个字符必须为“XXXXXX”，并且这些字符将被替换为使文件名唯一的字符串。然后以模式读/写和系统相关的权限创建该文件(在GNU/Linux系统上，glibc 2.0.7及更高版本的权限为0600)。该文件以二进制模式打开，并带有O\_EXCL标志。

如果提供了可选参数delete且为true，则在Octave退出时将自动删除文件。

如果成功，fid是一个有效的文件ID, name是文件名，msg是一个空字符串。否则，fid为-1,name为空，并且msg包含与系统相关的错误消息。

参见:tempname, tempdir, P\_tmpdir, tmpfile, fopen。

: [fid, msg] = tmpfile ()

返回与具有唯一名称的新临时文件对应的文件ID。

该文件以二进制读/写(“w+b”)模式打开，并在关闭或Octave退出时自动删除。

如果成功，fid是一个有效的文件ID, msg是一个空字符串。否则，fid为-1,msg包含与系统相关的错误消息。

参见:tempname, mkstemp, tempdir, P\_tmpdir。

: fname = tempname ()

: fname = tempname (dir)

: fname = tempname (dir, prefix)

以字符串形式返回唯一的临时文件名。

如果省略prefix，则使用值“oct-”。

如果还省略了dir，则使用临时文件的默认目录(P\_tmpdir)。如果提供了dir，它必须存在，否则将使用临时文件的默认目录。

编程注意:因为命名的文件不是通过tempname打开的，所以有可能(尽管相对不太可能)在您的程序试图打开它时它将不可用。如果这是一个问题，请参阅tmpfile。

参见:mkstemp, tempdir, P\_tmpdir, tmpfile。

: dir = tempdir ()

返回临时文件的主机系统目录的名称。

目录名首先取自环境变量TMPDIR。如果不存在，则检查环境变量TMP(在Windows平台上具有更高优先级的环境变量TEMP)。如果这些都没有设置，则使用P\_tmpdir返回的系统默认值。

参见:P\_tmpdir, tempname, mkstemp, tmpfile。

: sys\_tmpdir = P\_tmpdir ()

返回临时文件的主机系统默认目录的名称。

编程注意:P\_tmpdir返回的值总是默认位置。如果用户使用TMPDIR环境变量覆盖了默认值，则此值可能与从tempdir返回的值不一致。

参见:tempdir, tempname, mkstemp, tmpfile。

**14.2.18文件结束与错误**

一旦文件被打开，就可以获取它的状态。例如，feof函数确定是否到达了文件的末尾。这在一次读取文件的一小部分时非常有用。下面的示例展示了如何从文件中一次读取一行，直到到达末尾。

filename = "myfile.txt";

fid = fopen (filename, "r");

while (! feof (fid) )

text\_line = fgetl (fid);

endwhile

fclose (fid);

请注意，在某些情况下，读取文件的整个内容然后处理它比逐行读取更有效。这样做的潜在好处是可以消除上面代码中的循环。

: status = feof (fid)

如果遇到文件描述符fid指定的文件的文件结束条件，则返回1，否则返回0。

注意，如果已经遇到文件的末尾，feof只会返回1，如果下一次读取操作将导致文件结束条件，则不会返回1。

参见:fread, frewind, fseek, fclear, fopen。

: msg = ferror (fid)

: [msg, err] = ferror (fid)

: […] = ferror (fid, "clear")

查询文件描述符fid指定的流的错误状态。

如果存在错误条件，则返回描述错误的字符串msg。否则，返回空字符串""。

第二个输入“clear”是可选的。如果提供，流上的错误状态将被清除。

可选的第二个输出是错误状态的数字指示。如果遇到错误条件，Err为1，否则为0。

请注意，error表示错误是否已经发生，而不是下一个操作是否会导致错误条件。

参见:fclear, fopen。

: fclear (fid)

清除由文件描述符fid指定的文件的流状态。

参见:terror, fopen。

: freport ()

打印已打开的文件列表，以及它们是否为读、写或两者都打开。

例如:

freport ()

-| number mode arch name

-| ------ ---- ---- ----

-| 0 r ieee-le stdin

-| 1 w ieee-le stdout

-| 2 w ieee-le stderr

-| 3 r ieee-le myfile

参见:fopen, fclose, is\_valid\_file\_id。

**14.2.19文件定位**

有三个函数可用于设置和确定给定文件的文件指针的位置。

: pos = ftell (fid)

返回文件指针的位置，作为文件描述符fid指定的文件开头的字符数。

参见:fseek, frewind, fof, fopen。

: status = fseek (fid, offset)

: status = fseek (fid, offset, origin)

将文件指针设置为文件fid内的位置偏移量。

指针是指向原点的偏移字符，原点可以是预定义变量SEEK\_SET(开始)、SEEK\_CUR(当前位置)或SEEK\_END(文件结束)之一，也可以是字符串"bof"、"cof"或"eof"。如果省略origin，则假定为SEEK\_SET。偏移量可以是正的、负的或零，但不是所有的原点和偏移量的组合都可以实现。

Fseek成功时返回0，错误时返回-1。

参见:fskipl, frewind, ftell, fopen, SEEK\_SET, SEEK\_CUR, SEEK\_END。

: fseek\_origin = SEEK\_SET ()

返回要传递给fseek的数值，以定位文件指针相对于文件开头的位置。

参见:SEEK\_CUR, SEEK\_END, fseek。

: fseek\_origin = SEEK\_CUR ()

返回要传递给fseek的数值，以相对于当前位置定位文件指针。

参见:SEEK\_SET, SEEK\_END, fseek。

: fseek\_origin = SEEK\_END ()

返回要传递给fseek的数值，以相对于文件末尾定位文件指针。

参见:SEEK\_SET, SEEK\_CUR, fseek。

: frewind (fid)

: status = frewind (fid)

将文件指针移动到文件描述符fid指定的文件的开头。

如果请求输出状态，则frewind返回0表示成功，如果遇到错误则返回-1。

编程注意:frewind等价于fseek (fid, 0, SEEK\_SET)。

参见:fseek, ftell, fopen。

下面的示例将当前文件位置存储在变量标记中，将指针移动到文件的开头，读取四个字符，然后返回到原始位置。

marker = ftell (myfile);

frewind (myfile);

fourch = fgets (myfile, 4);

fseek (myfile, marker, SEEK\_SET);

**15 绘图**

**15.1绘图简介**

早期版本的Octave通过使用gnuplot提供绘图功能。此功能仍然可用。但是，新版本的Octave使用OpenGL提供了更现代的绘图功能。使用哪个绘图系统由graphics\_toolkit函数控制。参见图形工具包。

函数调用graphics\_toolkit ("qt")选择qt /OpenGL系统，graphics\_toolkit ("fltk")选择fltk /OpenGL系统，graphics\_toolkit ("gnuplot")选择gnuplot系统。通过使用每个图形句柄的graphics\_toolkit属性，可以有选择地使用这三个系统。这在图形数据结构中有解释。

注意:基于opengl的工具包在内部使用单精度变量，这限制了可以显示的最大值大约为10^{38}。如果您的数据包含较大的值，则必须使用gnuplot工具包，该工具包支持最大10^{308}的值。类似地，单精度变量只能准确地表示6-9个base10数字。如果您的数据包含非常细微的差异(大约1e-8)，则无法使用基于opengl的图形工具包解决这些差异，因此需要使用gnuplot工具包。

注意:gnuplot图形工具包使用第三方程序gnuplot进行绘图。从Octave到gnuplot的通信是通过单向管道完成的。这对性能和功能都有影响。性能明显变慢，因为整个数据集(可能有很多兆字节)必须通过管道传递给gnuplot。功能会受到负面影响，因为管道从Octave到gnuplot是单向的。Octave无法了解用户与绘图窗口的交互(无论是调整大小、移动、关闭还是其他任何操作)。如果稍后要从Octave访问图形，建议不要与gnuplot窗口交互(或关闭)。

**15.2高层绘图**

Octave提供了使用高级函数创建许多不同类型的二维和三维图形的简单方法。

如果需要更详细的控制，请参阅图形数据结构和高级绘图。

**15.2.1二维图**

plot函数允许您创建具有线性轴的简单x-y图。例如,

x = -10:0.1:10;

plot (x, sin (x));

xlabel ("x");

ylabel ("sin (x)");

title ("Simple 2-D Plot");

显示如图15.1所示的正弦波。在大多数系统上，该命令将打开一个单独的绘图窗口来显示图形。

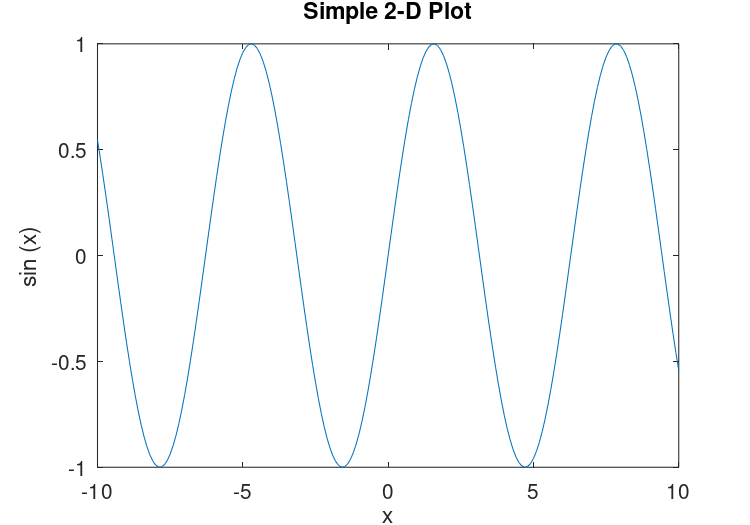


图15.1:简单二维图。

: plot (y)

: plot (x, y)

: plot (x, y, fmt)

: plot (…, property, value, …)

: plot (x1, y1, …, xn, yn)

: plot (hax, …)

: h = plot (…)

生成二维图。

许多不同的参数组合是可能的。最简单的形式是

plot (y)

其中参数作为y坐标的集合，x坐标的取值范围为1:numel (y)。

如果给出了多个参数，则将它们解释为

plot (y, property, value, …)

或

plot (x, y, property, value, …)

或

plot (x, y, fmt, …)

等等......。可以出现任意数量的参数集。x和y值的解释如下:

* 如果提供了单个数据参数，则将其作为y坐标的集合，并将x坐标作为元素的索引，从1开始。
* 如果x和y是标量，则绘制出一个点。
* Squeeze()应用于超过两个维度的参数，但不超过两个单例维度。
* 如果 x 或 y 是标量，另一个是矢量，则在标量和矢量的每个元素定义的坐标处绘制一系列点。
* 如果两个参数都是向量，则y的元素与x的元素对应。
* 如果x是一个向量，y是一个矩阵，那么y的列(或行)将相对于x绘制(使用匹配的组合，先尝试列)。
* 如果x是一个矩阵，y是一个向量，则y与x的列(或行)相对应(使用匹配的组合，先尝试列)。
* 如果两个参数都是矩阵，则将y的列与x的列进行对比。在这种情况下，两个矩阵必须具有相同的行数和列数，并且不会尝试转置参数以使行数匹配。

可以指定多个属性值对，但它们必须成对出现。这些参数应用于由plot绘制的线对象。可以修改的有用属性有“linestyle”、“linewidth”、“color”、“marker”、“markersize”、“markeredgecolor”、“markerfacecolor”。完整的属性列表记录在Line properties中。

fmt格式参数还可用于控制情节样式。它是一个由四个可选部分组成的字符串:"< linstyle ><marker><color><;displayname;>"。如果指定了标记，但没有指定线样式，则只绘制标记。类似地，如果指定了线条样式，但没有标记，则只绘制线条。如果两者都指定，则绘制线条和标记。如果没有fmt和没有属性/值对，那么默认的绘图样式是没有标记的实线，颜色由当前轴的“colororder”属性决定。

格式的参数:

线型

‘-’ 使用实线(默认)。

‘--’ 使用虚线。

‘:’ 使用虚线。

‘-.’ 使用虚线。

标志

‘+’ 十字

‘o’ 圆

‘\*’ 星

‘.’ 点

‘x’ 叉

‘|’ 垂直线

‘\_’ 水平线

‘s’ 正方形

‘d’ 菱形

‘^’ 上三角

‘v’ 下三角

‘>’ 右三角

‘<’ 左三角

‘p’ 五角星形

‘h’ 六角星

颜色

‘k’, "black" 黑色

‘r’, "red" 红色

‘g’, "green" 绿色

‘b’, "blue" 蓝色

‘y’, "yellow" 黄色

‘m’, "magenta" 品红

‘c’, "cyan" 青色

‘w’, "white" 白色

";displayname;"

分号之间的文本用于设置“displayname”属性，该属性决定了用于绘图图例的标签。

fmt参数也可用于分配图例标签。为此，在上述格式化序列之后的分号之间包含所需的标签，例如，“+b;Data Series 3;”。请注意，最后一个分号是必需的，如果省略它，Octave将生成一个错误。

下面是一些情节的例子:

plot (x, y, "or", x, y2, x, y3, "m", x, y4, "+")

该命令将用红色圆圈表示y，用实线表示y2，用品红色实线表示y3，用“+”表示点表示y4。

plot (b, "\*", "markersize", 10)

该命令将在变量b中绘制数据，点显示为“\*”，标记大小为10。

t = 0:0.1:6.3;

plot (t, cos(t), "-;cos(t);", t, sin(t), "-b;sin(t);");

这将绘制余弦和正弦函数，并在图例中相应地标记它们。

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。

可选的返回值h是创建的线对象的图形句柄向量。

要将绘图保存为PostScript或PNG等几种图像格式之一，请使用print命令。

**参见：**轴，框，网格，hold，图例，标题，xlabel, ylabel, xlim, ylim, ezplot, errorbar, fplot, line, plot3, polar, loglog, semilogx, semilogx, subplot。

plot函数可用于创建具有两个独立y轴的plot。

: plotyy (x1, y1, x2, y2)

: plotyy (…, fcn)

: plotyy (…, fun1, fun2)

: plotyy (hax, …)

: [ax, h1, h2] = plotyy (…)

绘制具有独立y轴和公共x轴的两组数据。

参数x1和y1定义了第一个图的参数，x1和y2定义了第二个图的参数。

默认情况下，参数使用feval (@plot, x, y)求值。然而，可以使用fcn参数修改plot的类型，在这种情况下，plot由feval (fcn, x, y)生成。fcn可以是函数句柄，内联函数或函数名字符串。

每个图使用的函数可以用fun1和fun2独立定义。

第一个参数hax可以是用于绘制x1和y1数据的主轴的轴句柄。它也可以是一个双元素向量，其轴柄指向主轴和副轴(参见输出轴)。

返回值ax是一个矢量，具有两个y轴的轴柄。H1和h2是由绘图命令生成的对象的句柄。

x = 0:0.1:2\*pi;

y1 = sin (x);

y2 = exp (x - 1);

ax = plotyy (x, y1, x - 1, y2, @plot, @semilogy);

xlabel ("X");

ylabel (ax(1), "Axis 1");

ylabel (ax(2), "Axis 2");

当将plotyy与subplot结合使用时，请确保首先调用subplot并将产生的轴句柄传递给plotyy。不要使用plotyy返回的任何轴句柄调用subplot，否则其他轴将被删除。

**参见：**plot, subplot。

函数semilogx、semilogy和loglog与plot函数类似，但它们产生的图中，其中一个或两个轴使用对数尺度。

: semilogx (y)

: semilogx (x, y)

: semilogx (x, y, property, value, …)

: semilogx (x, y, fmt)

: semilogx (hax, …)

: h = semilogx (…)

使用x轴的对数尺度生成二维图。

有关semilogx将接受的参数的描述，请参阅plot的文档。

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。

可选的返回值h是创建的绘图的图形句柄。

参见:情节，符号学，对数。

: semilogy (y)

: semilogy (x, y)

: semilogy (x, y, property, value, …)

: semilogy (x, y, fmt)

: semilogy (h, …)

: h = semilogy (…)

使用y轴的对数尺度生成二维图。

有关符号学可以接受的参数的描述，请参阅plot的文档。

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。

可选的返回值h是创建的绘图的图形句柄。

参见:plot, semilogx, loglog。

: loglog (y)

: loglog (x, y)

: loglog (x, y, prop, value, …)

: loglog (x, y, fmt)

: loglog (hax, …)

: h = loglog (…)

使用两个轴的对数尺度生成二维图。

有关loglog将接受的参数的描述，请参阅plot的文档。

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。

可选的返回值h是创建的绘图的图形句柄。

参见:plot, semilogx, semilogy。

函数bar, barh, stairs和stem用于显示离散数据。例如,

randn ("state", 1);

hist (randn (10000, 1), 30);

xlabel ("Value");

ylabel ("Count");

title ("Histogram of 10,000 normally distributed random numbers");

生成10000个正态分布随机数的直方图，如图15.2所示。注意，randn ("state"， 1);，将randn的随机数生成器初始化为一个已知值，以便返回的值是可重复的;这保证了生成的图形与本手册中的相同。

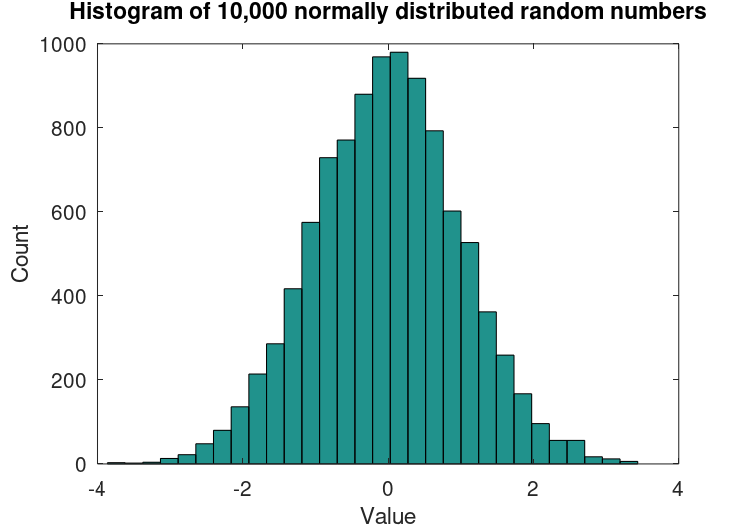


图15.2:直方图。

: bar (y)

: bar (x, y)

: bar (…, w)

: bar (…, style)

: bar (…, prop, val, …)

: bar (hax, …)

: h = bar (…, prop, val, …)

从X-Y数据的两个向量生成一个柱状图。

如果只给出一个参数y，则将其作为y值的向量，并且X坐标的范围为1:numel (y)。

可选输入w控制条的宽度。值为1.0将使每个柱完全接触相邻的柱。默认宽度为0.8。

如果y是一个矩阵，那么y的每一列都是一个单独的柱状图，绘制在同一个图上。默认情况下，列是并排绘制的。这种行为可以通过style参数来改变，它可以接受以下值:

"grouped" (default)

并排的条，条之间有间隙，并以x坐标为中心。

"stacked"

条形图是堆叠的，这样每个X值都有一个由多个片段组成的条形图。

"hist"

并排的条，条之间没有间隙，并以x坐标为中心。

"histc"

并排的条形图，条形图之间没有间隙，并与x坐标向左对齐。

可选的属性/值对直接传递给底层补丁对象。完整的属性列表记录在补丁属性中。

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。

可选的返回值h是创建的“bar series”hggroups的句柄向量，变量y的每列有一个句柄。这个系列使得可以更改一个bar series对象中的公共元素，并将更改反映在其他“bar series”中。例如,

h = bar (rand (5, 10));

set (h(1), "basevalue", 0.5);

在所有的酒吧系列的基础上改变位置。下面的示例使用属性/值对来修改脸和边缘颜色。

bar (randn (1, 100), "facecolor", "r", "edgecolor", "b");

条的默认颜色取自坐标轴的“ColorOrder”属性。当使用直方图选项(“hist”，“histc”)时，条形图的默认颜色是轴或图形的“Colormap”属性。条形条的颜色也可以使用“facecolor”属性手动设置，如下图所示。

h = bar (rand (10, 3));

set (h(1), "facecolor", "r")

set (h(2), "facecolor", "g")

set (h(3), "facecolor", "b")

参见:barh, hist, pie, plot, patch。

: barh (y)

: barh (x, y)

: barh (…, w)

: barh (…, style)

: barh (…, prop, val, …)

: barh (hax, …)

: h = barh (…, prop, val, …)

从X-Y数据的两个向量生成一个水平条形图。

如果只给出一个参数，则将其作为Y值的向量，并且X坐标的范围为1:numel (Y)。

可选输入w控制条的宽度。值为1.0将使每个柱完全接触相邻的柱。默认宽度为0.8。

如果y是一个矩阵，那么y的每一列都是一个单独的柱状图，绘制在同一个图上。默认情况下，列是并排绘制的。这种行为可以通过style参数来改变，它可以接受以下值:

"grouped" (default)

并排的条，条之间有间隙，并以y坐标为中心。

"stacked"

条形图是堆叠的，这样每个Y值都有一个由多个分段组成的条形图。

"hist"

并排的条，条之间没有间隙，并以y坐标为中心。

"histc"

并排的条形图，条形图之间没有间隙，并与y坐标向左对齐。

可选的属性/值对直接传递给底层补丁对象。完整的属性列表记录在补丁属性中。

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。

可选的返回值h是创建的条系列hg组的图形句柄。有关bar系列的使用说明，请参见bar。

参见:bar, hist, pie, plot, patch。

: hist (y)

: hist (y, nbins)

: hist (y, x)

: hist (y, x, norm)

: hist (…, prop, val, …)

: hist (hax, …)

: [nn, xx] = hist (…)

生成直方图计数或图。

使用一个矢量输入参数y，绘制具有10个箱子的值的直方图。直方图bins的范围由数据的范围(y中最大值和最小值的差值)决定。极值被集中到第一个和最后一个bins中。如果y是一个矩阵，那么绘制一个直方图，其中每个箱子包含一个柱状图，每个输入列是y。

如果第二个可选参数是一个标量bbin，它定义了箱子的数量。

如果第二个可选参数是一个向量x，它定义了箱子的中心。箱子的宽度由向量中相邻的值确定。箱子的总数是numel (x)。

如果提供了第三个参数norm，则直方图被规范化。如果norm是一个正标量，则结果条被归一化为norm。如果norm是长度列(y)的正标量向量，则y(:，i)的结果条归一化为norm(i)。

[nn, xx] = hist (rand (10, 3), 5, [1 2 3]);

sum (nn, 1)

⇒ ans =

1 2 3

直方图的外观可以通过向底层补丁对象指定属性/值对来修改。例如，可以修改面和边的颜色:

hist (randn (1, 100), 25, "facecolor", "r", "edgecolor", "b");

补丁属性的完整列表记录在补丁属性。财产。如果未指定，则直方图的默认颜色取自坐标轴或图形的“Colormap”属性。

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。

如果请求输出，则不进行绘图。相反，返回值nn(元素个数)和xx (bin中心)，这样bar (xx, nn)将绘制直方图。

参见:历史，酒吧，派，玫瑰。

: stemleaf (x, caption)

: stemleaf (x, caption, stem\_sz)

: plotstr = stemleaf (…)

计算并显示向量x的茎叶图。

输入的x应该是一个整数向量。任何非整数值将通过x = fix (x)转换为整数值。默认情况下，x的每个元素将以元素的最后一位数字作为叶值，其余数字作为茎值绘制。例如，123将被绘制成茎' 12 '和叶' 3 '。第二个参数，标题，应该是一个字符数组，它提供数据的描述。它是作为输出的标题包含的。

可选输入stem\_sz设置每个茎的宽度。茎宽由10^(stem\_sz + 1)决定。默认茎宽为10。

stemleaf的输出由两部分组成:“围栏字母显示”，然后是茎叶图本身。在探索性数据分析中描述了围栏字母显示。简要地说，条目如下:

Fenced Letter Display

#% nx|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ nx = numel (x)

M% mi| md | mi median index, md median

H% hi|hl hu| hs hi lower hinge index, hl,hu hinges,

1 |x(1) x(nx)| hs h\_spreadx(1), x(nx) first

\_\_\_\_\_\_\_ and last data value.

\_\_\_\_\_\_|step |\_\_\_\_\_\_\_ step 1.5\*h\_spread

f|ifl ifh| inner fence, lower and higher

|nfl nfh| no.\ of data points within fences

F|ofl ofh| outer fence, lower and higher

|nFl nFh| no.\ of data points outside outer

fences

茎叶图在每一行上显示茎的值，后面跟着由叶子数字组成的字符串。如果stem\_sz不为1，则连续的叶子值用“，”分隔。

在没有返回参数的情况下，将立即显示该图。如果提供了输出参数，则以字符串数组的形式返回绘图。

叶数字没有排序。如果需要排序的叶值，在调用stemleaf (xs)之前使用xs = sort (x)。

茎叶图和相关的显示在:第3章，探索性数据分析，J. W. Tukey, Addison-Wesley, 1977。

参见:hist, print。

: printd (obj, filename)

: out\_file = printd (…)

将任何可接受的对象转换为filename后缀所选择的格式。

如果请求可选输出out\_file，则返回创建的文件名。

此函数旨在方便操作诸如stemleaf等函数的输出。

参见: stemleaf。

: stairs (y)

: stairs (x, y)

: stairs (…, style)

: stairs (…, prop, val, …)

: stairs (hax, …)

: h = stairs (…)

: [xstep, ystep] = stairs (…)

制作一个阶梯图。

参数x和y可以是向量或矩阵。如果只给出一个参数，则将其作为Y值的向量，并将X坐标作为元素的索引(X = 1:numel (Y))。

可以使用与plot命令格式相同的行样式来定义用于绘图的样式。

可以指定多个属性/值对，但它们必须成对出现。完整的属性列表记录在Line properties中。

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。

如果请求一个输出参数，则返回创建的绘图的图形句柄。如果指定了两个输出参数，则生成数据，但不绘制数据。例如,

stairs (x, y);

和

[xs, ys] = stairs (x, y);

plot (xs, ys);

是等价的。

参见:bar, hist, plot, stem。

: stem (y)

: stem (x, y)

: stem (…, linespec)

: stem (…, "filled")

: stem (…, prop, val, …)

: stem (hax, …)

: h = stem (…)

绘制一个二维茎图。

如果只给出一个参数，则将其作为y值，并从元素的索引中获取x坐标。

如果y是一个矩阵，那么矩阵的每一列被绘制成一个单独的干图。在这种情况下，x可以是一个向量，和y中的行数一样长，或者它可以是一个和y一样大的矩阵。

默认颜色为“b”(蓝色)，默认线条样式为“-”，默认标记为“o”。可以通过linespec参数以与plot命令相同的方式更改行样式。如果存在" filling "参数，则将填充词干顶部的标记。例如,

x = 1:10;

y = 2\*x;

stem (x, y, "r");

畦10茎，高2 ~ 20，红色;

可以指定可选的属性/值对来控制绘图的外观。完整的属性列表记录在Line properties中。

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。

可选的返回值h是一个“茎系列”hg组的句柄。单个hg组句柄将构成情节的所有图形元素作为其子元素;这允许通过修改“stem series”hg组的单个属性来改变多个图形对象的属性。

例如,

x = [0:10]';

y = [sin(x), cos(x)]

h = stem (x, y);

set (h(2), "color", "g");

set (h(1), "basevalue", -1)

改变第二个“茎系列”的颜色并移动第一个“茎系列”的基线。

Stem系列特性

linestyle

茎的线形。(默认:“-”)

linewidth

茎的宽度。(默认值:0.5)

color

茎的颜色，如果没有单独指定，则标记。(默认值:"b"[蓝色])

marker

在每个茎的顶部使用的标记符号。(默认值是“o”)

markeredgecolor

标记笔的边缘颜色。(默认:"color"属性)

markerfacecolor

用于“填充”标记的颜色。(默认值:"none" [unfill])

markersize

记号笔的大小。(默认值:6)

baseline

实现基线的行对象的句柄。使用set和返回的句柄来更改基线的图形属性。

basevalue

绘制基线的y值。(默认值:0)

参见:stem3, bar, hist, plot, stairs。

: stem3 (x, y, z)

: stem3 (…, linespec)

: stem3 (…, "filled")

: stem3 (…, prop, val, …)

: stem3 (hax, …)

: h = stem3 (…)

绘制三维茎图。

茎从高度z向x和y确定的x-y平面的位置绘制。默认颜色为“b”(蓝色)，默认线条样式为“-”，默认标记为“o”。

可以通过linespec参数以与plot命令相同的方式更改行样式。如果存在" filling "参数，则将填充词干顶部的标记。

可以指定可选的属性/值对来控制绘图的外观。完整的属性列表记录在Line properties中。

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。

可选返回值h是“stem series”hg组的句柄，该组包含用于绘图的行和标记对象。有关“stem系列”对象的描述，请参阅stem。

例子:

theta = 0:0.2:6;

stem3 (cos (theta), sin (theta), theta);

在一个圆圈上画出31根高度从0到6的茎。

实现注意:带有rgb三元组的颜色定义无效。

参见:stem, bar, hist, plot。

: scatter (x, y)

: scatter (x, y, s)

: scatter (x, y, s, c)

: scatter (…, style)

: scatter (…, "filled")

: scatter (…, prop, val, …)

: scatter (hax, …)

: h = scatter (…)

绘制二维散点图。

在由向量x和y中的坐标定义的每个点上绘制一个标记。

标记的大小由s决定，s可以是一个标量，也可以是与x和y长度相同的向量。如果s没有给出，或者是一个空矩阵，则使用默认值36平方点(标记大小本身是sqrt (s))。

标记的颜色由c决定，c可以是定义固定颜色的字符串;一个3元素向量，表示颜色的红、绿、蓝成分;一个与x长度相同的向量，给出当前颜色图的缩放索引;或定义每个标记的RGB颜色的Nx3矩阵。

使用的标记可以通过style参数改变;它是一个字符串，以与plot命令相同的方式定义标记。如果没有指定标记，则默认为“0”或圆形。如果给出了参数" filling "，则标记被填充。

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。

可选的返回值h是创建的散射对象的图形句柄。

例子:

x = randn (100, 1);

y = randn (100, 1);

scatter (x, y, [], sqrt (x.^2 + y.^2));

编程注意:属性的完整列表记录在Scatter properties。

**参见**:scatter3, patch, plot。

: plotmatrix (x, y)

: plotmatrix (x)

: plotmatrix (…, style)

: plotmatrix (hax, …)

: [h, ax, bigax, p, pax] = plotmatrix (…)

一个矩阵相对于另一个矩阵的列的散点图。

给定具有匹配行数的参数x和y, plotmatrix绘制一组对应于的轴

plot (x(:, i), y(:, j))

当用单个参数x调用时，这相当于

plotmatrix (x, x)

除了轴集的对角线将被直方图hist (x(:， i))取代。

可以使用style参数更改要使用的标记，这是一个以与plot命令相同的方式定义标记的字符串。

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。

可选返回值h提供散点图中各个图形对象的句柄，而ax返回散点图轴对象的句柄。

Bigax是一个隐藏轴对象，它包围着其他轴，这样xlabel、title等命令将与这个隐藏轴相关联。

最后，p返回与直方图关联的图形对象，并对相应的轴对象进行pax。

例子:

plotmatrix (randn (100, 3), "g+")

参见:scatter, plot。

: pareto (y)

: pareto (y, x)

: pareto (hax, …)

: h = pareto (…)

画一个帕累托图。

帕累托图是一种条形图，它以一种可以确定过程改进优先级的方式排列信息;它组织和显示信息，以显示数据的相对重要性。该图表类似于直方图或条形图，不同之处在于，条形图沿着x轴从左到右按递减幅度排列。

使用帕累托图背后的基本思想(帕累托原则)是，大多数效果是由于一小部分原因造成的。对于质量改进，导致问题的前几个原因(图中最左边的条形图)通常占结果的大部分。因此，针对这些“主要原因”加以消除，可形成最具成本效益的改进方案。

通常只有大小数据y存在，在这种情况下，x被取为范围1:长度(y)。如果x给定，它可能是字符串数组，字符串单元数组或数值向量。

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。

可选的返回值h是一个2元素向量，其中一个图形句柄用于创建的条形图，另一个句柄用于创建的线形图。

使用帕累托的一个例子是

Cheese = {"Cheddar", "Swiss", "Camembert", ...

"Munster", "Stilton", "Blue"};

Sold = [105, 30, 70, 10, 15, 20];

pareto (Sold, Cheese);

参见:bar, barh, hist, pie, plot。

: rose (theta)

: rose (theta, nbins)

: rose (theta, bins)

: rose (hax, …)

: h = rose (…)

: [th r] = rose (…)

绘制角度直方图。

使用一个矢量参数th，绘制具有20个角度bin的直方图。如果th是一个矩阵，那么th的每一列产生一个单独的直方图。

如果给定了nbin并且是一个标量，则直方图由nbin生成。如果bins是一个向量，那么每个bins的中心由bins中的值定义，并且bins的数量由bins中的元素数量给出。

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。

可选的返回值h是表示每个直方图的线对象的图形句柄向量。

如果请求两个输出参数，则不会绘制直方图，而是返回绘制直方图所需的极向量。

例子

[th, r] = rose ([2\*randn(1e5,1), pi + 2\*randn(1e5,1)]);

polar (th, r);

参见:hist, polar。

等高线、等高线和等高线函数从三维数据生成二维等高线图。

: contour (z)

: contour (z, vn)

: contour (x, y, z)

: contour (x, y, z, vn)

: contour (…, style)

: contour (hax, …)

: [c, h] = contour (…)

创建一个二维等高线图。

绘制矩阵z的水平曲线(等高线)，使用等高线计算的等高线矩阵c从相同的参数;参见后者的解释。

等高线的外观可以用线条样式样式来定义，其方式与绘图相同。只使用线条样式和颜色;任何由样式定义的标记都将被忽略。

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。

可选输出c包含等高线格式的等高线级别。

可选返回值h是包含等高线的hg组的图形句柄。

例子:

x = 0:3;

y = 0:2;

z = y' \* x;

contour (x, y, z, 2:3)

参见:ezcontour, contourc, contourf, contour3，标签，网格，曲面，坐标，颜色图，绘图。

: contourf (z)

: contourf (z, vn)

: contourf (x, y, z)

: contourf (x, y, z, vn)

: contourf (…, style)

: contourf (hax, …)

: [c, h] = contourf (…)

创建一个填充间隔的二维等高线图。

绘制矩阵z的水平曲线(等高线)，并用当前颜色图中的颜色填充线之间的区域。

等高线曲线取自等高线矩阵c，等高线矩阵c是等高线计算得到的。参见后者的解释。

等高线的外观可以用线条样式样式来定义，其方式与绘图相同。只使用线条样式和颜色;任何由样式定义的标记都将被忽略。

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。

可选输出c包含等高线格式的等高线级别。

可选返回值h是包含等高线的hg组的图形句柄。

下面的示例图填充了峰值函数的轮廓。

[x, y, z] = peaks (50);

contourf (x, y, z, -7:9)

参见:ezcontourf, contour, contourc, contour3，标签，meshc, surfc, axis, colormap, plot。

: c = contourc (z)

: c = contourc (z, vn)

: c = contourc (x, y, z)

: c = contourc (x, y, z, vn)

: [c, lev] = contourc (…)

计算等高线(Z值恒定的等高线)。

矩阵z包含由x和y确定的矩形网格上方的高度值。如果只提供单个输入z，则x取1:列(z)， y取1:行(z)。最小数据大小为2x2。

可选的输入vn要么是表示要计算的等高线数量的标量，要么是包含将计算线条的Z值的矢量。当vn为矢量时，等高线的数量为numel (vn)。但是，要在给定值处计算单个等高线，请使用vn = [val, val]。如果省略vn，则默认为10。

返回值c是一个2xn矩阵，包含如下格式的等高线

c = [lev1, x1, x2, …, levn, x1, x2, ...

len1, y1, y2, …, lenn, y1, y2, …]

其中等高线n的水平(高度)为len，长度为len。

可选的返回值lev是一个包含等高线的Z值的向量。

例子:

x = 0:2;

y = x;

z = x' \* y;

c = contourc (x, y, z, 2:3)

⇒ c =

2.0000 1.0000 1.0000 2.0000 2.0000 3.0000 1.5000 2.0000

4.0000 2.0000 2.0000 1.0000 1.0000 2.0000 2.0000 1.5000

参见:轮廓，轮廓，轮廓3，标签。

: contour3 (z)

: contour3 (z, vn)

: contour3 (x, y, z)

: contour3 (x, y, z, vn)

: contour3 (…, style)

: contour3 (hax, …)

: [c, h] = contour3 (…)

创建三维等高线图。

contour3在对应于每个轮廓的z级绘制矩阵z的水平曲线(等高线)。这与等高线相反，等高线在相同的Z水平上绘制所有等高线并产生二维图。

等高线曲线取自等高线矩阵c，等高线矩阵c是等高线计算得到的。参见后者的解释。

等高线的外观可以用线条样式样式来定义，其方式与绘图相同。只使用线条样式和颜色;任何由样式定义的标记都将被忽略。

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。

可选输出c是等高线格式的等高线级别。

可选返回值h是包含等高线的hg组的图形句柄。

例子:

contour3 (peaks (19));

colormap cool;

hold on;

surf (peaks (19), "facecolor", "none", "edgecolor", "black");

参见：轮廓，轮廓，轮廓，标签，网格，表面，轴，颜色图，情节。

errorbar、semilogxerr、semilogyerr和loglogerr函数生成带有错误条标记的图表。例如,

rand ("state", 2);

x = 0:0.1:10;

y = sin (x);

lerr = 0.1 .\* rand (size (x));

uerr = 0.1 .\* rand (size (x));

errorbar (x, y, lerr, uerr);

axis ([0, 10, -1.1, 1.1]);

xlabel ("x");

ylabel ("sin (x)");

title ("Errorbar plot of sin (x)");

生成图15.3所示的图。

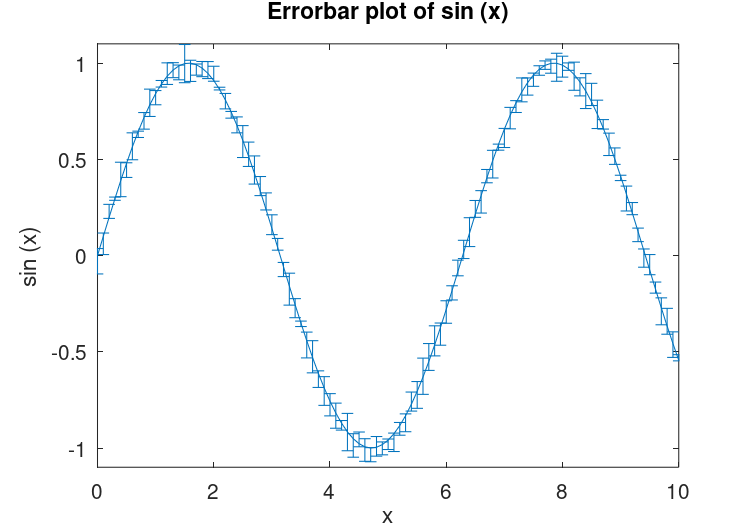


图15.3:错误栏图。

: errorbar (y, ey)

: errorbar (y, …, fmt)

: errorbar (x, y, ey)

: errorbar (x, y, err, fmt)

: errorbar (x, y, lerr, uerr, fmt)

: errorbar (x, y, ex, ey, fmt)

: errorbar (x, y, lx, ux, ly, uy, fmt)

: errorbar (x1, y1, …, fmt, xn, yn, …)

: errorbar (hax, …)

: h = errorbar (…)

创建带有错误条的二维图。

许多不同的参数组合是可能的。最简单的形式是

errorbar (y, ey)

其中第一个参数作为y坐标的集合，第二个参数ey是y值周围的误差，x坐标作为元素的索引(1:numel (y))。

函数的一般形式是

errorbar (x, y, err1, …, fmt, …)

在x和y参数之后，可以有1、2或4个参数，根据错误值的性质和绘图格式fmt指定错误值。

err (scalar)

当误差是标量时，所有点共享相同的误差值。错误栏是对称的，从data-err绘制到data+err。fmt参数决定err是在x方向，y方向(默认)，还是两者都在。

err (vector or matrix)

每个数据点都有一个特定的误差值。错误栏是对称的，从data(n)-err(n)绘制到data(n)+err(n)。

lerr, uerr (scalar)

误差有一个低端值和一个上端值。错误栏不是对称的，从data-lerr绘制到data+uerr。

lerr, uerr (vector or matrix)

每个数据点都有一个低侧误差和一个上侧误差。错误条不是对称的，是从data(n)-lerr(n)到data(n)+ userr (n)绘制的。

任何数量的数据集(x1,y1, x2,y2，…)都可以出现，只要用格式字符串fmt分隔它们。

如果y是矩阵，则x和误差参数也必须是具有相同维数的矩阵。y的列与x的相应列相对应，误差条取自误差参数的相应列。

如果缺少fmt，则假定为yerrorbars(“~”)绘图样式。

如果提供了fmt参数，那么就像在普通绘图中一样，将其解释为指定线条样式、标记和颜色。此外，fmt可能包括一个错误栏样式，它必须位于普通格式代码之前。支持以下错误栏样式:

‘~’

设置yerrorbars绘图样式(默认)。

‘>’

设置xerrorbars绘图样式。

‘~>’

设置xyerrorbars绘图样式。

‘#~’

设置yboxes的绘图样式。

‘#’

设置xbox的情节风格。

‘#~>’

设置xyboxes的情节样式。

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。

可选的返回值h是表示数据图和错误栏的hggroup对象的句柄。

注意:为了与MATLAB兼容，通过所有数据点绘制一条线。然而，大多数科学误差条图是带有误差条的点的散点图。为此，向fmt参数添加标记样式，如"."。或者，通过使用set (h，“linestyle”，“none”)修改返回的图形句柄来删除行。

例子:

errorbar (x, y, ex, ">.r")

生成y与x的xerrorbar图，从x-ex到x+ex绘制x个错误条。由于使用了标记“。”，因此没有绘制连接线，错误条显示为红色。

errorbar (x, y1, ey, "~",

x, y2, ly, uy)

生成y1和y2相对于x的误差条图。y1的误差条从y1-ey绘制到y1+ey, y2的误差条从y2-ly绘制到y2+ y。

errorbar (x, y, lx, ux,

ly, uy, "~>")

生成y与x的xyerrorbar图，其中x个错误条从x-lx绘制到x+ux, y个错误条从y-ly绘制到y+ y。

参见:semilogxerr, semilogyerr, loglogerr, plot。

: semilogxerr (y, ey)

: semilogxerr (y, …, fmt)

: semilogxerr (x, y, ey)

: semilogxerr (x, y, err, fmt)

: semilogxerr (x, y, lerr, uerr, fmt)

: semilogxerr (x, y, ex, ey, fmt)

: semilogxerr (x, y, lx, ux, ly, uy, fmt)

: semilogxerr (x1, y1, …, fmt, xn, yn, …)

: semilogxerr (hax, …)

: h = semilogxerr (…)

使用x轴的对数尺度和每个数据点的误差条生成二维图。

许多不同的参数组合是可能的。最常见的形式是

semilogxerr (x, y, ey, fmt)

它生成y与x的半对数图，其中y尺度的误差由ey定义，图格式由fmt定义。有关可用格式和其他信息，请参阅错误栏。

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:errorbar, semilogyerr, loglogerr。

: semilogyerr (y, ey)

: semilogyerr (y, …, fmt)

: semilogyerr (x, y, ey)

: semilogyerr (x, y, err, fmt)

: semilogyerr (x, y, lerr, uerr, fmt)

: semilogyerr (x, y, ex, ey, fmt)

: semilogyerr (x, y, lx, ux, ly, uy, fmt)

: semilogyerr (x1, y1, …, fmt, xn, yn, …)

: semilogyerr (hax, …)

: h = semilogyerr (…)

使用y轴的对数尺度和每个数据点的误差条生成二维图。

许多不同的参数组合是可能的。最常见的形式是

semilogyerr (x, y, ey, fmt)

它生成y与x的半对数图，其中y尺度的误差由ey定义，图格式由fmt定义。有关可用格式和其他信息，请参阅错误栏。

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:errorbar, semilogxerr, loggerr。

: loglogerr (y, ey)

: loglogerr (y, …, fmt)

: loglogerr (x, y, ey)

: loglogerr (x, y, err, fmt)

: loglogerr (x, y, lerr, uerr, fmt)

: loglogerr (x, y, ex, ey, fmt)

: loglogerr (x, y, lx, ux, ly, uy, fmt)

: loglogerr (x1, y1, …, fmt, xn, yn, …)

: loglogerr (hax, …)

: h = loglogerr (…)

在带有误差条的双对数轴上生成二维图。

许多不同的参数组合是可能的。最常见的形式是

loglogerr (x, y, ey, fmt)

它生成y对x的双对数图，其中y尺度的误差由ey定义，图格式由fmt定义。有关可用格式和其他信息，请参阅错误栏。

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:errorbar, semilogxerr, semilogyerr。

最后，极坐标函数允许您轻松地在极坐标中绘制数据。然而，显示坐标仍然是矩形和线性的。例如,

polar (0:0.1:10\*pi, 0:0.1:10\*pi);

title ("Example polar plot from 0 to 10\*pi");

生成如图15.4所示的螺旋图。

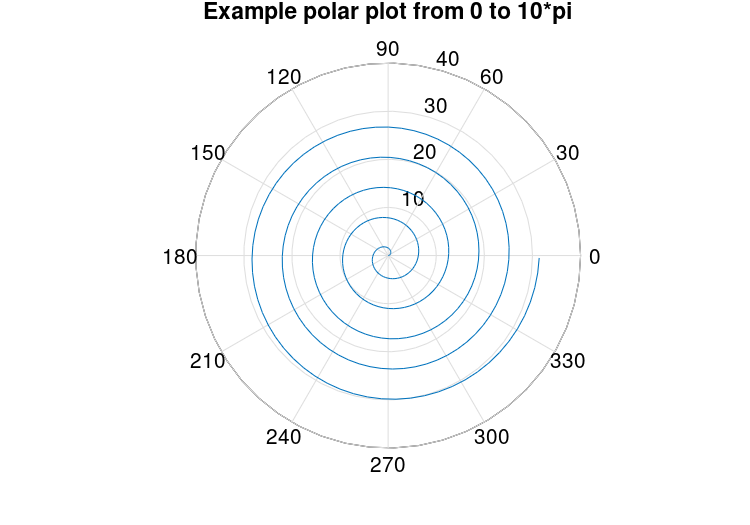


图15.4:极坐标图。

: polar (theta, rho)

: polar (theta, rho, fmt)

: polar (cplx)

: polar (cplx, fmt)

: polar (hax, …)

: h = polar (…)

从极坐标和创建一个二维图。

输入被假定为弧度，并被转换成度数来绘图。如果有度数，则必须在将数据传递给此函数之前将其转换(参见cart2pol)为弧度。

如果给定一个单复输入cplx，则实部用于，虚部用于。

可选参数fmt以与plot相同的方式指定行格式。

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。

可选的返回值h是创建的绘图的图形句柄。

实现注意:极轴是使用封装在hg组中的直线和文本对象绘制的。hggroup属性链接到原始轴对象，这样更改外观属性(例如fontname)将更新极轴。两个新属性被添加到原始轴中——rtick, tick——它们取代了xtick, ytick。第一个是径向(rho)方向上的刻度位置列表;第二个是在角(θ)方向上的刻度位置列表，以度指定，即在0-359范围内。

参见:rose, compass, plot, car2pol。

: pie (x)

: pie (…, explode)

: pie (…, labels)

: pie (hax, …)

: h = pie (…)

画一个二维饼状图。

当使用单个vector参数调用时，生成x中元素的饼状图。第i片的大小是元素xi所代表的x总和的百分比:pct = x(i) / sum (x)。

可选的输入explosion是一个与x长度相同的向量，如果它不为零，则“爆炸”饼图中的切片。

可选的输入标签是一个与x长度相同的字符串的单元格数组，指定每个切片的标签。

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。

可选的返回值h是生成绘图的补丁和文本对象的句柄列表。

注意:如果sum (x)≤1，则x的元素被直接解释为百分比，而不是由sum (x)规范化。此外，如果sum小于1，则饼图中会有一个缺失的切片来表示缺失的未指定百分比。

参见:pie3, bar, hist, rose。

: pie3 (x)

: pie3 (…, explode)

: pie3 (…, labels)

: pie3 (hax, …)

: h = pie3 (…)

绘制一个三维饼状图。

使用单个向量参数调用，生成x中元素的三维饼状图。第i片的大小是元素xi占x总和的百分比:pct = x(i) / sum (x)。

可选的输入explosion是一个与x长度相同的向量，如果它不为零，则“爆炸”饼图中的切片。

可选的输入标签是一个与x长度相同的字符串的单元格数组，指定每个切片的标签。

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。

可选的返回值h是生成绘图的补丁、表面和文本对象的图形句柄列表。

注意:如果sum (x)≤1，则x的元素被直接解释为百分比，而不是由sum (x)规范化。此外，如果sum小于1，则饼图中会有一个缺失的切片来表示缺失的未指定百分比。

参见：pie, bar, hist, rose。

: quiver (u, v)

: quiver (x, y, u, v)

: quiver (…, s)

: quiver (…, style)

: quiver (…, "filled")

: quiver (hax, …)

: h = quiver (…)

用箭头绘制二维矢量场。

在（x，y）定义的网格点处绘制矢量场的（u，v）分量。如果网格是均匀的，则可以将x和y指定为网格矢量，并使用网格网格来创建二维网格。

如果未给出x和y，则假定它们为（1：m，1：n），其中[m，n] = 大小（u）。

可选输入s是一个标量，用于定义相对于网格间距的场箭头的比例因子。值为1.0将导致最长的矢量正好填满一个网格正方形。值为0或“off”将禁用所有缩放。默认值为0.9。

可以使用与plot命令格式相同的线条样式、样式来定义用于打印的样式。如果指定了标记，则标记绘制在矢量的原点（即由x和y定义的网格点）。指定标记时，不会绘制箭头。如果给出参数“filled”，则填充标记。如果使用名称-值打印样式属性，则它们必须成对显示，并遵循任何其他打印样式参数。

如果第一个参数hax是一个轴句柄，则绘制到此轴中，而不是gca返回的当前轴。

可选返回值h是箭袋对象的图形句柄。箭袋对象对箭袋图的组件（主体、箭头和标记）进行重新组合，并允许将它们一起更改。

例如：

[x, y] = meshgrid (1:2:20);

h = quiver (x, y, sin (2\*pi\*x/10), sin (2\*pi\*y/10));

set (h, "maxheadsize", 0.33);

参见:quiver3, compass, feather, plot。

: quiver3 (x, y, z, u, v, w)

: quiver3 (z, u, v, w)

: quiver3 (…, s)

: quiver3 (…, style)

: quiver3 (…, "filled")

: quiver3 (hax, …)

: h = quiver3 (…)

用箭头绘制一个 3D 矢量场。

在由（x，y，z）定义的网格点处绘制矢量场的（u，v，w）分量。如果网格是均匀的，则可以将x、y和z指定为网格矢量，并使用网格网格来创建3-D网格。

如果未给出x和y，则假定它们为（1：m，1：n），其中 [m，n] = 大小（u）。

可选输入s是一个标量，用于定义相对于网格间距的场箭头的比例因子。值为1.0将导致最长的矢量正好填充一个网格立方体。值为0或“off”将禁用所有缩放。默认值为0.9。

可以使用与plot命令格式相同的线型样式来定义用于打印的样式。如果指定了标记，则标记将绘制在矢量的原点（由x，y，z定义的网格点）。指定标记时，不会绘制箭头。如果给出参数“filled”，则填充标记。如果使用名称-值打印样式属性，则它们必须成对显示，并遵循任何其他打印样式参数。

如果第一个参数hax是一个轴句柄，则绘制到此轴中，而不是gca返回的当前轴。

可选返回值h是箭袋对象的图形句柄。箭袋对象对箭袋图的组件（主体、箭头和标记）进行重新组合，并允许将它们一起更改。

[x, y, z] = peaks (25);

surf (x, y, z);

hold on;

[u, v, w] = surfnorm (x, y, z / 10);

h = quiver3 (x, y, z, u, v, w);

set (h, "maxheadsize", 0.33);

参见:quiver, compass, feather, plot。

: streamribbon (x, y, z, u, v, w, sx, sy, sz)

: streamribbon (u, v, w, sx, sy, sz)

: streamribbon (xyz, x, y, z, anlr\_spd, lin\_spd)

: streamribbon (xyz, anlr\_spd, lin\_spd)

: streamribbon (xyz, anlr\_rot)

: streamribbon (…, width)

: streamribbon (hax, …)

: h = streamribbon (…)

计算和显示流带。

流带是通过根据向量场的角度旋转围绕流线的法向量来构造的。

向量场由[u, v, w]给出，定义在矩形网格上，由[x, y, z]给出。流带从种子点[sx, sy, sz]开始。

Streamribbon可以用包含预先计算的流线数据的单元数组调用。为此，必须使用stream3函数创建xyz。Lin\_spd是向量场的线速度，可以由[u, v, w]通过平方和的平方根计算得到。角速度anlr\_spd是角速度在归一化向量场速度上的投影，可以用curl命令计算。如果需要更改积分器步长或流线顶点的最大数量，则此选项非常有用。

或者，可以从路径曲线的顶点xyz数组创建带状。Anlr\_rot包含路径曲线相邻顶点之间的边缘旋转角度。

输入参数width设置流带的宽度。

流带根据沿流带旋转的总角度着色。

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。

可选的返回值h是为每个streamribbon创建的绘图对象的图形句柄。

例子:

[x, y, z] = meshgrid (0:0.2:4, -1:0.2:1, -1:0.2:1);

u = - x + 10;

v = 10 \* z.\*x;

w = - 10 \* y.\*x;

streamribbon (x, y, z, u, v, w, [0, 0], [0, 0.6], [0, 0]);

view (3);

参见:流线，流线，流管，流管。

: streamtube (x, y, z, u, v, w, sx, sy, sz)

: streamtube (u, v, w, sx, sy, sz)

: streamtube (xyz, x, y, z, div)

: streamtube (xyz, div)

: streamtube (xyz, dia)

: streamtube (…, options)

: streamtube (hax, …) ¶

: h = streamtube (…)

按流线发散度缩放的图管。

Streamtube绘制的管道的直径由矢量场的散度缩放。向量场由[u, v, w]给出，定义在矩形网格上，由[x, y, z]给出。管从种子点[sx, sy, sz]开始，沿着流线绘制。

Streamtube也可以使用包含预先计算的流线数据的单元数组来调用。为此，必须使用stream3命令创建xyz。Div用于缩放管道。为了绘制由矢量场散度缩放的管，必须使用散度命令计算div。

管径为零对应流线上最小的结垢值，管径最大对应最大的结垢值。

也可以沿着任意顶点数组xyz绘制一个管。管的直径可以由顶点阵列的直径或常数指定。

输入参数选项是形式为[scale, n]的二维矢量。第一个参数缩放管径(默认为1)。第二个参数指定用于构造管周长的顶点数(默认为20)。

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。

可选的返回值h是为每个管创建的绘图对象的图形句柄。

参见:stream3, streamlined, streamribbon, ostreamtube。

: ostreamtube (x, y, z, u, v, w, sx, sy, sz)

: ostreamtube (u, v, w, sx, sy, sz)

: ostreamtube (xyz, x, y, z, u, v, w)

: ostreamtube (…, options)

: ostreamtube (hax, …) ¶

: h = ostreamtube (…)

计算和显示流管。

流管近似于沿流线连接圆形横流区域。流动的扩展是由局部横流散度决定的。

向量场由[u, v, w]给出，定义在矩形网格上，由[x, y, z]给出。流管从种子点[sx, sy, sz]开始。

这些管是根据局部矢量场强度着色的。

输入参数选项是形式为[scale, n]的二维矢量。第一个参数缩放流管的起始半径(默认为1)。第二个参数指定用于构造管周长的顶点数(默认为20)。

Ostreamtube可以用包含预先计算的流线数据的单元数组调用。为此，必须使用stream3函数创建xyz。如果需要更改积分器步长或流线顶点的最大数量，则此选项非常有用。

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。

可选的返回值h是为每个流管创建的绘图对象的图形句柄。

例子:

[x, y, z] = meshgrid (-1:0.1:1, -1:0.1:1, -3:0.1:0);

u = -x / 10 - y;

v = x - y / 10;

w = - ones (size (x)) / 10;

ostreamtube (x, y, z, u, v, w, 1, 0, 0);

参见:stream3, streamline, streamribbon, streamtube。

: streamline (x, y, z, u, v, w, sx, sy, sz)

: streamline (u, v, w, sx, sy, sz)

: streamline (…, options)

: streamline (hax, …)

: h = streamline (…)

绘制二维或三维矢量场的流线。

绘制由[u, v]或[u, v, w]给出的二维或三维矢量场的流线。向量场定义在由[x, y]或[x, y, z]给出的矩形网格上。流线从种子点[sx, sy]或[sx, sy, sz]开始。

输入参数选项是形式为[stepsize, max\_vertices]的二维矢量。第一个参数指定用于轨迹集成的步长(默认为0.1)。负值是允许的，这将逆转积分的方向。第二个参数指定用于创建流线的最大段数(默认为10,000)。

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。

可选的返回值h是包含字段行的hg组的图形句柄。

例子:

[x, y] = meshgrid (-1.5:0.2:2, -1:0.2:2);

u = - x / 4 - y;

v = x - y / 4;

streamline (x, y, u, v, 1.7, 1.5);

参见:stream2, stream3, streamribbon, streamtube, ostreamtube。

: xy = stream2 (x, y, u, v, sx, sy)

: xy = stream2 (u, v, sx, sy)

: xy = stream2 (…, options)

计算二维流线数据。

计算由[u, v]给出的向量场的流线。向量场定义在由[x, y]给出的矩形网格上。流线从种子点开始[sx, sy]。返回值xy包含顶点数组的单元格数组。如果起始点在向量场之外，则返回[]。

输入参数选项是形式为[stepsize, max\_vertices]的二维矢量。第一个参数指定用于轨迹集成的步长(默认为0.1)。负值是允许的，这将逆转积分的方向。第二个参数指定用于创建流线的最大段数(默认为10,000)。

返回值xy是一个包含场线段坐标的反转x 2矩阵。

例子:

[x, y] = meshgrid (0:3);

u = 2 \* x;

v = y;

xy = stream2 (x, y, u, v, 1.0, 0.5);

参见:流线型，流线型。

: xyz = stream3 (x, y, z, u, v, w, sx, sy, sz)

: xyz = stream3 (u, v, w, sx, sy, sz)

: xyz = stream3 (…, options)

计算三维流线数据。

计算由[u, v, w]给出的向量场的流线。向量场定义在由[x, y, z]给出的矩形网格上。流线从种子点[sx, sy, sz]开始。返回值xyz包含顶点数组的单元格数组。如果起始点在向量场之外，则返回[]。

输入参数选项是形式为[stepsize, max\_vertices]的二维矢量。第一个参数指定用于轨迹集成的步长(默认为0.1)。负值是允许的，这将逆转积分的方向。第二个参数指定用于创建流线的最大段数(默认为10,000)。

返回值xyz是一个包含字段线段坐标的nversions x 3矩阵。

例子:

[x, y, z] = meshgrid (0:3);

u = 2 \* x;

v = y;

w = 3 \* z;

xyz = stream3 (x, y, z, u, v, w, 1.0, 0.5, 0.0);

参见:stream2，流线型，streamribbon, streamtube, ostreamtube。

: compass (u, v)

: compass (z)

: compass (…, style)

: compass (hax, …)

: h = compass (…)

画出从极坐标原点出发的向量场的(u, v)分量。

代表每个矢量的箭头在原点有一端，在[u(i)， v(i)]有一端。如果给出单个复参数z，则u = real (z)， v = imag (z)。

可以使用与plot命令格式相同的行样式来定义用于绘图的样式。

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。

可选的返回值h是表示绘制向量的线对象的图形句柄向量。

a = toeplitz ([1;randn(9,1)], [1,randn(1,9)]);

compass (eig (a));

参见:极地，羽毛，箭，玫瑰，情节。

: feather (u, v)

: feather (z)

: feather (…, style)

: feather (hax, …)

: h = feather (…)

画出从x轴上等距离的点发出的向量场的(u, v)分量。

如果给出单个复参数z，则u = real (z)， v = imag (z)。

可以使用与plot命令格式相同的行样式来定义用于绘图的样式。

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。

可选的返回值h是表示绘制向量的线对象的图形句柄向量。

phi = [0 : 15 : 360] \* pi/180;

feather (sin (phi), cos (phi));

参见:plot, quiver, compass。

: pcolor (x, y, c)

: pcolor (c)

: pcolor (hax, …)

: h = pcolor (…)

生成二维密度图。

pcolor plot在由矩阵x和y表示的二维区域上用矩阵c绘制颜色的矩形。x和y是网格顶点的坐标，通常是网格的输出。如果x和y是向量，那么一个典型的顶点是(x(j)， y(i)， c(i,j))。因此，c的列对应不同的x值，c的行对应不同的y值。

c中的值被缩放以跨越当前颜色图的范围。可以通过命令轴或通过设置父轴的clim属性在颜色轴上放置限制。

网格中每个单元的表面颜色是通过插值每个单元顶点的c值来确定的;与此形成对比的是imagesc，它为c的每个元素呈现一个单元格。

着色修改了一个属性，该属性决定了每个单元格的表面颜色从c值插值的方式，以及单元格边缘的可见性。默认情况下，该属性是“faceted”，它为每个单元的表面呈现一种颜色，并使边缘可见。

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。

可选的返回值h是创建的表面对象的图形句柄。

参见:轴，阴影，网格，轮廓，图像。

: area (y)

: area (x, y)

: area (…, lvl)

: area (…, prop, val, …)

: area (hax, …)

: h = area (…)

y的列的面积图。

该图显示了每个列值对行和的贡献。它在功能上类似于图(x, cumsum (y, 2))，除了曲线下的面积是阴影。

如果省略x参数，则默认为1:行(y)。可以定义一个值lvl，该值确定应该在曲线下定义阴影的基本级别。默认级别为0。

附加的属性/值对直接传递给底层补丁对象。完整的属性列表记录在补丁属性中。

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。

可选的返回值h是包含区域补丁对象的hggroup对象的图形句柄。hg组的“BaseValue”属性可以用来调整阴影开始的级别。

例:验证身份sin^2 + cos^2 = 1

t = linspace (0, 2\*pi, 100)';

y = [sin(t).^2, cos(t).^2];

area (t, y);

legend ("sin^2", "cos^2", "location", "NorthEastOutside");

参见:plot, patch。

: fill (x, y, c)

: fill (x1, y1, c1, x2, y2, c2)

: fill (…, prop, val)

: fill (hax, …)

: h = fill (…)

创建一个或多个填充的二维多边形。

输入x和y是多边形顶点的坐标。如果输入是矩阵，那么行代表不同的顶点，每列产生一个不同的多边形。填充将在绘制之前关闭任何打开的多边形。

输入c决定了多边形的颜色。最简单的形式是单色规范，如绘图格式或RGB-triple。在这种情况下，多边形将有一个唯一的颜色。如果c是一个向量或矩阵，那么颜色数据首先使用坐标轴进行缩放，然后索引到当前颜色图中。行向量将为每个多边形(矩阵x和y中的一列)使用单一计算颜色上色。与x和y大小相同的矩阵c将计算每个顶点的颜色，然后在顶点之间插入面颜色。

可以为底层补丁对象指定多个属性/值对，但它们必须成对出现。完整的属性列表记录在补丁属性中。

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。

可选的返回值h是创建的补丁对象的图形句柄向量。

例子:红色方块

vertices = [0 0

1 0

1 1

0 1];

fill (vertices(:,1), vertices(:,2), "r");

axis ([-0.5 1.5, -0.5 1.5])

axis equal

参见:patch, fill3, taxis, colormap。

: fill3 (x, y, z, c)

: fill3 (x1, y1, z1, c1, x2, y2, z2, c2)

: fill3 (…, prop, val)

: fill3 (hax, …)

: h = fill3 (…)

创建一个或多个填充的3d多边形。

输入x, y和z是多边形顶点的坐标。如果输入是矩阵，那么行代表不同的顶点，每列产生一个不同的多边形。Fill3将在绘图前关闭所有打开的多边形。

输入c决定了多边形的颜色。最简单的形式是单色规范，如绘图格式或RGB-triple。在这种情况下，多边形将有一个唯一的颜色。如果c是一个向量或矩阵，那么颜色数据首先使用坐标轴进行缩放，然后索引到当前颜色图中。行向量将为每个多边形(矩阵x、y和z中的列)上色，并使用单一的计算颜色。与x, y和z大小相同的矩阵c将计算每个顶点的颜色，然后在顶点之间插入面部颜色。

可以为底层补丁对象指定多个属性/值对，但它们必须成对出现。完整的属性列表记录在补丁属性中。

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。

可选的返回值h是创建的补丁对象的图形句柄向量。

例如:斜红色矩形

vertices = [0 0 0

1 1 0

1 1 1

0 0 1];

fill3 (vertices(:,1), vertices(:,2), vertices(:,3), "r");

axis ([-0.5 1.5, -0.5 1.5, -0.5 1.5]);

axis ("equal");

grid ("on");

view (-80, 25);

参见:patch, fill, cab, colormap。

: comet (y)

: comet (x, y)

: comet (x, y, p)

: comet (hax, …)

沿着输入坐标向量(x, y)提供的轨迹生成一个简单的彗星样式动画。

如果未指定x，则默认为y的下标。

彗星的速度可以用p来控制，p表示每个点在移动到下一个点之前显示的时间。p的默认值是5 / nummel (y)。

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:彗星。

: comet3 (z)

: comet3 (x, y, z)

: comet3 (x, y, z, p)

: comet3 (hax, …)

沿着输入坐标向量(x, y, z)提供的轨迹生成一个简单的彗星样式动画。

如果只指定z，则x, y默认为z的索引。

彗星的速度可以用p来控制，p表示每个点在移动到下一个点之前显示的时间。p的默认值是5 / numel (z)。

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:彗星。

**15.2.1.1轴配置**

轴函数可用于更改现有图的轴限制和各种其他轴属性，例如纵横比和tic标记的外观。默认情况下，高级绘图功能(如绘图重置轴属性)。任何属性的定制，例如通过调用axis、xlim等，都应该在绘图完成之后进行，或者在调用hold函数之后进行。

: axis ()

: axis ([x\_lo x\_hi])

: axis ([x\_lo x\_hi y\_lo y\_hi])

: axis ([x\_lo x\_hi y\_lo y\_hi z\_lo z\_hi])

: axis ([x\_lo x\_hi y\_lo y\_hi z\_lo z\_hi c\_lo c\_hi])

: axis (option)

: axis (option1, option2, …)

: axis (hax, …)

: limits = axis ()

设置轴的限制和外观。

参数限制应该是2个、4个、6个或8个元素的向量。第一个和第二个元素指定x轴的下限和上限。第三、第四是y轴的界限，第五、第六是z轴的界限，第七、第八是颜色轴的界限。特殊值-Inf和Inf可用于表示应根据轴中的数据自动计算极限。

在没有任何参数的情况下，axis将打开自动缩放。

使用一个输出参数，limits = axis返回当前的轴限制。

指定限制的vector参数是可选的，其他字符串参数可用于指定各种轴属性。

以下选项控制轴的纵横比。

"equal"

使x轴单位距离等于y轴(和z轴)单位距离。

"square"

强制一个方轴宽高比。

"vis3d"

设置宽高比模式(“DataAspectRatio”，“PlotBoxAspectRatio”)为“手动”旋转而不拉伸。

"normal"

"fill"

恢复默认自动计算的宽高比。

以下选项控制轴限制的解释方式。

"auto"

"auto[xyz]"

"auto [xyz]"

为所有轴设置良好的自动计算限制，或者仅为指定的轴设置限制。

"manual"

修复当前轴的限制。

"tickaligned"

将坐标轴固定到最近刻度的极限。

"tight"

将坐标轴固定到数据的极限。

"padded"

将轴固定到数据的极限加上数据范围的7%左右的边距。

"image"

相当于"tight"和"equal"。

以下选项会影响标记的外观。

"tic"

"tic[xyz]"

"tic [xyz]"

为所有轴打开刻度标记，或为指定轴打开刻度标记，为其余轴关闭刻度标记。

"label"

"label[xyz]"

"label [xyz]"

向所有轴旋转标签,或将其转换为指定的坐标轴和其余的轴。

"nolabel"

关闭所有轴的勾号标签。

注意:如果一个轴没有勾号，那么就没有标签。

以下选项影响轴上增加值的方向。

"xy"

默认的y轴，较大的值靠近顶部。

"ij"

与y轴相反，较小的值靠近顶部。

以下选项会影响轴的可见性。

"on"

使坐标轴可见。

"off"

把斧子藏起来。

如果第一个参数hax是一个轴句柄，那么操作这个轴，而不是gca返回的当前轴。

示例1:设置X/Y限制并强制使用方形宽高比

axis ([1, 2, 3, 4], "square");

示例2:在所有轴上启用打勾标记，仅在y轴上启用打勾标记标签

axis ("tic", "labely");

参见:xlim, ylim, zlim, axis, daspect, pbaspect, box, grid。

类似地，颜色图的轴限可以通过轴函数来改变。

: caxis ([cmin cmax])

: caxis ("auto")

: caxis ("manual")

: caxis (hax, …)

: limits = caxis ()

查询或设置绘图的颜色轴限制。

limits参数应该是一个2元素的向量，指定要分配给颜色映射中的第一个和最后一个值的下限和上限。超出此范围的数据值被固定在第一个和最后一个颜色映射项上。

如果给出“auto”选项，则应用自动色图限制。自动算法将cmin设置为最小数据值，cmax设置为最大值。如果指定了“manual”，则“climmode”属性被设置为“manual”，并且“clim”属性中的数值用于限制。

如果第一个参数hax是一个轴句柄，那么操作这个轴，而不是gca返回的当前轴。

如果不带参数调用，则返回当前颜色轴限制。

编程注意:颜色轴影响图像、补丁和表面图形对象的显示，但仅当“cdata”属性具有索引数据并且“cdatamapping”属性设置为“缩放”时才会影响。具有真彩cdata或“直接”cdata映射的图形对象不受影响。

参见:colormap, axis。

xlim、ylim和zlim函数可用于获取或设置单独的轴极限。每个都有相同的形式。

: xlimits = xlim ()

: xmode = xlim ("mode")

: xmethod = xlim ("method")

: xlim ([x\_lo x\_hi])

: xlim ("mode")

: xlim ("method")

: xlim (hax, …)

查询或设置当前绘图的x轴范围。

不带参数调用xlim返回当前绘图的x轴范围。

通过输入查询“mode”，返回当前的x限制计算模式，可以是“自动”或“手动”。

使用输入查询“method”，返回当前的x限制计算方法，可以是“tickaligned”、“tight”或“pfilled”。

如果传递2元素矢量 [x\_lo x\_hi]，则x轴的限制将设置为这些值，并且模式将设置为“手动”。特殊值-Inf和Inf可用于指示应自动计算轴下限或轴上限。

电流限制计算“模式”可以是以下之一

"auto"（默认）

根据绘图数据和当前指定的限值计算方法自动计算限值。

"manual"

将轴限制固定为当前值。

电流限制计算方法可以是以下之一

"tickaligned"（默认）

计算包含所有数据并向外延伸到最近的刻度线的限制。

"tight"

计算完全符合数据范围的限制。

"padded"

计算在数据周围留下约7%数据范围余量的限制。

如果第一个参数hax是轴句柄，则对该轴而不是gca返回的当前轴进行操作。

编程注意事项：xlim函数通过修改坐标区对象的“xlim”、“xlimmode”和“xlimitmethod”属性来运行。这些属性可以通过get/set直接检查和更改。

参见: ylim, zlim, axis, set, get, gca。

: ylimits = ylim ()

: ymode = ylim ("mode")

: ymethod = ylim ("method")

: ylim ([y\_lo y\_hi])

: ylim ("mode")

: ylim ("method")

: ylim (hax, …)

查询或设置当前绘图的y轴范围。

不带参数调用ylim返回当前绘图的y轴范围。

通过输入查询“mode”，返回当前y限制计算模式，可以是“自动”或“手动”。

使用输入查询“method”，返回当前y限制计算方法，可以是“tickaligned”、“tight”或“pfilled”。

如果传递2元素矢量[y\_lo y\_hi]，则y轴的限制将设置为这些值，并且模式将设置为“手动”。特殊值-Inf和Inf可用于指示应自动计算轴下限或轴上限。

电流限制计算“模式”可以是以下之一

"auto"（默认）

根据绘图数据和当前指定的限值计算方法自动计算限值。

"manual"

将轴限制固定为当前值。

电流限制计算方法可以是以下之一

"tickaligned"（默认）

计算包含所有数据并向外延伸到最近的刻度线的限制。

"tight"

计算完全符合数据范围的限制。

"padded"

计算在数据周围留下约7%数据范围余量的限制。

如果第一个参数hax是轴句柄，则对该轴而不是gca返回的当前轴进行操作。

编程注意：ylim函数通过修改坐标区对象的“ylim”、“ylimmode”和“ylimitmethod”属性来运行。这些属性可以通过get/set直接检查和更改。

参见: xlim, zlim, axis, set, get, gca。

: zlimits = zlim ()

: zmode = zlim ("mode")

: zmethod = zlim ("method")

: zlim ([z\_lo z\_hi])

: zlim ("mode")

: zlim ("method")

: zlim (hax, …)

查询或设置当前绘图的z轴限制。

不带参数调用zlim返回当前绘图的z轴范围。

通过输入查询“mode”，返回当前z限制计算模式，可以是“自动”或“手动”。

使用输入查询“method”，返回当前z限制计算方法，可以是“tickaligned”、“tight”或“pfilled”。

如果传递2元素矢量 [z\_lo z\_hi]，则z轴的限制将设置为这些值，并且模式将设置为“手动”。特殊值-Inf和Inf可用于指示应自动计算轴下限或轴上限。

电流限制计算“模式”可以是以下之一

"auto"（默认）

根据绘图数据和当前指定的限值计算方法自动计算限值。

"manual"

将轴限制固定为当前值。

电流限制计算方法可以是以下之一

"tickaligned"（默认）

计算包含所有数据并向外延伸到最近的刻度线的限制。

"tight"

计算完全符合数据范围的限制。

"padded"

计算在数据周围留下约7%数据范围余量的限制。

如果第一个参数hax是轴句柄，则对该轴而不是gca返回的当前轴进行操作。

编程注意事项：zlim函数通过修改坐标区对象的“zlim”、“zlimmode”和“zlimitmethod”属性来运行。这些属性可以通过get/set直接检查和更改。

参见: xlim, ylim, axis, set, get, gca.

xticks、yticks、zticks、rticks 和 thetaticks 函数可用于获取或设置相应轴上的刻度线位置和模式。每个都具有相同的形式，尽管模式选项目前不适用于 rtick 和 thetaticks。

: tickval = xticks

: mode = xticks ("mode")

: xticks (tickval)

: xticks ("auto")

: xticks ("manual")

: … = xticks (hax, …)

查询或设置当前轴的x轴上的刻度值。

在没有参数的情况下调用时，返回“xtick”轴属性中指定的当前刻度位置。可以通过调用具有刻度值矢量的xtick来更改这些位置。注意：不需要升序。

当使用参数“mode”调用时，xticks返回坐标区属性“xtickmode”的当前值。可以通过使用“自动”（算法确定刻度位置）或“手动”（无论轴大小调整或旋转如何，刻度值都保持固定）调用xtick来更改此属性。注意：指定xtick值还会将属性“xtickmode”设置为“manual”。

如果第一个参数hax是轴句柄，则在此轴上操作，而不是gca返回的当前轴。

调用xticks设置属性值时请求返回值将导致错误。

参见: xticklabels, yticks, zticks, rticks, thetaticks, get, set.

: tickval = yticks

: mode = yticks ("mode")

: yticks (tickval)

: yticks ("auto")

: yticks ("manual")

: … = yticks (hax, …)

查询或设置当前轴的 y 轴上的刻度值。

在没有参数的情况下调用时，返回“ytick”轴属性中指定的当前刻度位置。可以通过调用具有 tick 值矢量的 ytick 来更改这些位置。注意：不需要升序。

当使用参数“mode”调用时，yticks 返回坐标区属性“ytickmode”的当前值。可以通过使用“auto”（算法确定即时报价位置）或“manual”（无论轴大小调整或旋转，即时报价值都保持固定）调用 ytick 来更改此属性。注意：指定 ytick 值还会将属性“ytickmode”设置为“manual”。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

调用 yticks 设置属性值时请求返回值将导致错误。

参见: yticklabels, xticks, zticks, rticks, thetaticks, get, set.

: tickval = zticks

: mode = zticks ("mode")

: zticks (tickval)

: zticks ("auto")

: zticks ("manual")

: … = zticks (hax, …)

查询或设置当前轴的 z 轴上的刻度值。

在没有参数的情况下调用时，返回“ztick”轴属性中指定的当前刻度位置。可以通过调用具有 tick 值矢量的 ztick 来更改这些位置。注意：不需要升序。

当使用参数“mode”调用时，zticks 返回坐标区属性“ztickmode”的当前值。可以通过使用“auto”（算法确定即时报价位置）或“manual”（无论轴大小调整或旋转如何，即时报价值都保持固定）调用 ztick 来更改此属性。注意：指定 ztick 值还会将属性“ztickmode”设置为“manual”。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

在调用 zticks 设置属性值时请求返回值将导致错误。

参见: zticklabels, xticks, yticks, rticks, thetaticks, get, set。

: tickval = rticks

: rticks (tickval)

: … = rticks (hax, …)

查询或设置当前轴的 r 轴上的刻度值。

在没有参数的情况下调用时，返回“rtick”轴属性中指定的当前刻度位置。可以通过调用具有 tick 值矢量的 rtick 来更改这些位置。注意：不需要升序。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

在调用 rticks 设置属性值时请求返回值将导致错误。

注意：Octave 目前不实现 polaraxes 对象。因此，无法像对 x、y 和 z 轴的等效函数那样查询或设置“rtick”属性的“模式”。

参见: thetaticks, xticks, yticks, zticks, polar, get, set.

: tickval = thetaticks

: thetaticks (tickval)

: … = thetaticks (hax, …)

查询或设置当前轴的 theta 轴上的刻度值。

在没有参数的情况下调用时，返回“ttick”轴属性中指定的当前刻度位置。可以通过调用具有刻度值矢量的 thetatick 来更改这些位置。注意：不需要升序。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

在调用 thetaticks 设置属性值时请求返回值将导致错误。

注意：Octave 目前不实现 polaraxes 对象。因此，无法像对 x、y 和 z 轴的等效函数那样查询或设置“thetatick”属性的“模式”。

参见: rticks, xticks, yticks, zticks, polar, get, set.

xticklabels、yticklabels 和 zticklabels 函数可用于获取或设置分配给每个刻度位置的标签以及相应轴上的标记模式。每个都有相同的形式。

: labels = xticklabels

: mode = xticklabels ("mode")

: xticklabels (tickval)

: xticklabels ("auto")

: xticklabels ("manual")

: … = xticklabels (hax, …)

查询或设置当前轴的 x 轴上的刻度标签。

在没有参数的情况下调用时，返回在“xticklabel”轴属性中指定的当前刻度标签的字符串的元胞数组。可以通过调用带有字符串单元格数组的 xticklabels 来更改这些标签。注意：数字矢量将映射到字符串的元胞数组。如果指定的标签少于当前刻度数，则空白标签将追加到数组中。

当使用参数“mode”调用时，xticklabels 返回坐标区属性“xticklabelmode”的当前值。可以通过使用“auto”（算法确定刻度标签）或“manual”（刻度标签保持固定）调用 xticklabels 来更改此属性。注意：指定 xticklabel 值还会将“xticklabelmode”和“xticks”属性设置为“manual”。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

调用 xticklabels 设置属性值时请求返回值将导致错误。

参见: xticks, yticklabels, zticklabels, get, set.

: labels = yticklabels

: mode = yticklabels ("mode")

: yticklabels (tickval)

: yticklabels ("auto")

: yticklabels ("manual")

: … = yticklabels (hax, …)

查询或设置当前轴的 x 轴上的刻度标签。

在没有参数的情况下调用时，返回“yticklabel”轴属性中指定的当前刻度标签的字符串的元胞数组。可以通过调用带有字符串单元格数组的 yticklabels 来更改这些标签。注意：数字矢量将映射到字符串的元胞数组。如果指定的标签少于当前刻度数，则空白标签将追加到数组中。

当使用参数“mode”调用时，yticklabels 返回坐标区属性“yticklabelmode”的当前值。可以通过使用“auto”（算法确定刻度标签）或“manual”（刻度标签保持固定）调用 yticklabels 来更改此属性。注意：指定 yticklabel 值还会将“yticklabelmode”和“yticks”属性设置为“manual”。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

调用 xticklabels 设置属性值时请求返回值将导致错误。

参见: yticks, xticklabels, zticklabels, get, set.

: labels = zticklabels

: mode = zticklabels ("mode")

: zticklabels (tickval)

: zticklabels ("auto")

: zticklabels ("manual")

: … = zticklabels (hax, …)

查询或设置当前轴的 x 轴上的刻度标签。

在没有参数的情况下调用时，返回“zticklabel”轴属性中指定的当前刻度标签的字符串的元胞数组。可以通过调用带有字符串元胞数组的 zticklabels 来更改这些标签。注意：数字矢量将映射到字符串的元胞数组。如果指定的标签少于当前刻度数，则空白标签将追加到数组中。

当使用参数“mode”调用时，zticklabels 返回坐标属性“zticklabelmode”的当前值。可以通过使用“auto”（算法确定刻度标签）或“manual”（刻度标签保持固定）调用 zticklabels 来更改此属性。注意：指定 zticklabel 值还会将“zticklabelmode”和“zticks”属性设置为“manual”。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

调用 xticklabels 设置属性值时请求返回值将导致错误。

参见: zticks, xticklabels, zticklabels, get, set.

xtickangle、ytickangle 和 ztickangle 函数可用于获取或设置相应轴的标签旋转角度。每个都有相同的形式。

: angle = xtickangle ()

: angle = xtickangle (hax)

: xtickangle (angle)

: xtickangle (hax, angle)

查询或设置当前轴 x 轴上刻度标签的旋转角度。

在不带参数的情况下调用时，返回 axes 属性“XTickLabelRotation”中指定的刻度标签的旋转角度（以度为单位）。当使用数字标量角度调用时，逆时针旋转刻度标签以角度度数。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

编程说明：

1. “XTickLabelRotation”属性当前未在 Octave 中实现。可以设置和查询属性，但对绘图没有影响。
2. 在设置指定轮换的同时请求返回值将导致错误。

参见: ytickangle, ztickangle, get, set.

: angle = ytickangle ()

: angle = ytickangle (hax)

: ytickangle (angle)

: ytickangle (hax, angle)

查询或设置当前轴 y 轴上刻度标签的旋转角度。

在没有参数的情况下调用时，返回 axes 属性“YTickLabelRotation”中指定的刻度标签的旋转角度（以度为单位）。当使用数字标量角度调用时，逆时针旋转刻度标签以角度度数。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

编程说明：

1. “YTickLabelRotation”属性当前未在 Octave 中实现。可以设置和查询属性，但对绘图没有影响。
2. 在设置指定轮换的同时请求返回值将导致错误。

参见: xtickangle, ztickangle, get, set.

: angle = ztickangle ()

: angle = ztickangle (hax)

: ztickangle (angle)

: ztickangle (hax, angle)

查询或设置当前轴 z 轴上刻度标签的旋转角度。

在没有参数的情况下调用时，返回 axes 属性“ZTickLabelRotation”中指定的刻度标签的旋转角度（以度为单位）。当使用数字标量角度调用时，逆时针旋转刻度标签以角度度数。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

编程说明：

1. “ZTickLabelRotation”属性当前未在 Octave 中实现。可以设置和查询属性，但对绘图没有影响。
2. 在设置指定轮换的同时请求返回值将导致错误。

参见: xtickangle, ytickangle, get, set.

**15.2.1.2二维函数绘图**

Octave可以从函数句柄或定义函数的字符串绘制函数，而无需用户显式创建要绘制的数据。函数fplot还使用函数名和x坐标范围的限制(而不是x和y数据)生成具有线性轴的二维图。例如,

fplot (@sin, [-10, 10], 201);

生成一个与上面相同的图，但还包括一个图例，显示所绘制函数的名称。

: fplot (fcn)

: fplot (fcn, limits)

: fplot (…, tol)

: fplot (…, n)

: fplot (…, fmt)

: fplot (…, property, value, …)

: fplot (hax, …)

: [x, y] = fplot (…)

在由限制定义的范围内绘制函数fcn。

FCN是一个函数句柄、内联函数或包含要求值的函数名的字符串。

情节的界限为[xlo, xhi]或[xlo, xhi, ylo, yhi]的形式。如果没有指定限制，默认值为[- 5,5]。

接下来的三个参数都是可选的，它们中的任意数量可以以任意顺序给出。

Tol是用于绘图的相对容忍度，默认为2e-3(0.2%)。

N是要使用的最小点数。当指定n时，最大步长为(xhi - xlo) / n，为满足相对容差要求，仍可使用超过n个点。

fmt参数指定plot命令要使用的行样式。

也可以指定多个属性值对，但它们必须成对出现。这些参数应用于由plot绘制的线对象。

行属性的完整列表记录在行属性。

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。

在没有输出参数的情况下，结果会立即绘制出来。使用两个输出参数，将返回二维绘图数据。随后可以使用plot (x, y)手动绘制数据。

例子:

fplot (@cos, [0, 2\*pi])

fplot ("[cos(x), sin(x)]", [0, 2\*pi])

编程指出:

Fplot最适合连续函数。具有不连续的函数不可能很好地绘图。这一限制将来可能会被取消。

当函数接受并返回一个矢量参数时，Fplot性能会更好。在编写用户定义函数并使用。\*、。/等逐个元素操作符时，请考虑这一点。

**参见:**ezplot, plot。

其他可以直接从函数创建二维绘图的函数包括ezplot, ezcontour, ezcontourf和ezpolar。

: ezplot (f)

: ezplot (f2v)

: ezplot (fx, fy)

: ezplot (…, dom)

: ezplot (…, n)

: ezplot (hax, …)

: h = ezplot (…)

绘制由函数f定义的二维曲线。

函数f可以是字符串、内联函数或函数句柄，可以有一个或两个变量。如果f有一个变量，则函数在-2\*pi < x < 2\*pi的定义域上绘制500个点。

如果f2v是两个变量的函数，则在网格域-2\*pi <= x | y <= 2\*pi上计算隐式函数f(x,y) = 0，每个维度上有60个点。

例如:

ezplot (@(x, y) x.^2 - y.^2 - 1)

如果两个函数作为输入传递，那么参数函数

x = fx (t)

y = fy (t)

在-2\*pi <= t <= 2\*pi的域上绘制，有500个点。

如果dom是一个双元素向量，它表示x和y的最小值和最大值，或者表示参数图的t。如果dom是一个四元素向量，那么最小值和最大值是[xmin xmax ymin ymax]。

N是定义用于绘制函数的点的数量的标量。

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。

可选的返回值h是创建的线对象的图形句柄向量。

也见:plot, ezplot3, ezpolar, ezcontour, ezcontourf, ezmesh, ezmeshc, ezsurf, ezsurfc。

: ezcontour (f)

: ezcontour (…, dom)

: ezcontour (…, n)

: ezcontour (hax, …)

: h = ezcontour (…)

绘制函数的等高线。

F是一个字符串、内联函数或函数句柄，带有定义函数的两个参数。默认情况下，图在网格域-2\*pi <= x | y <= 2\*pi上，每个维度上有60个点。

如果dom是一个双元素向量，它表示x和y的最小值和最大值。如果dom是一个四元素向量，那么最小值和最大值是[xmin xmax ymin ymax]。

N是定义在每个维度中使用的点的数量的标量。

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。

可选的返回值h是创建的绘图的图形句柄。

例子:

f = @(x,y) sqrt (abs (x .\* y)) ./ (1 + x.^2 + y.^2);

ezcontour (f, [-3, 3]);

**参见:**contour, ezcontourf, ezplot, ezmeshc, ezsurfc。

: ezcontourf (f)

: ezcontourf (…, dom)

: ezcontourf (…, n)

: ezcontourf (hax, …)

: h = ezcontourf (…)

绘制函数的填充等高线。

F是一个字符串、内联函数或函数句柄，带有定义函数的两个参数。默认情况下，图在网格域-2\*pi <= x | y <= 2\*pi上，每个维度上有60个点。

如果dom是一个双元素向量，它表示x和y的最小值和最大值。如果dom是一个四元素向量，那么最小值和最大值是[xmin xmax ymin ymax]。

N是定义在每个维度中使用的点的数量的标量。

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。

可选的返回值h是创建的绘图的图形句柄。

例子:

f = @(x,y) sqrt (abs (x .\* y)) ./ (1 + x.^2 + y.^2);

ezcontourf (f, [-3, 3]);

参见:contourf, ezcontour, ezplot, ezmeshc, ezsurfc。

: ezpolar (f)

: ezpolar (…, dom)

: ezpolar (…, n)

: ezpolar (hax, …)

: h = ezpolar (…)

在极坐标中绘制一个二维函数。

函数f是一个字符串、内联函数或带有单个参数的函数句柄。函数的期望形式是= f()默认情况下，该图位于0 <= theta <= 2\*pi的域上，有500个点。

如果dom是一个包含两个元素的向量，它表示theta的最小值和最大值。

N是定义用于绘制函数的点的数量的标量。

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。

可选的返回值h是创建的绘图的图形句柄。

例子:

ezpolar (@(t) sin (5/4 \* t), [0, 8\*pi]);

参见:polar, ezplot。

**15.2.1.3二维几何形状**

: rectangle ()

: rectangle (…, "Position", pos)

: rectangle (…, "Curvature", curv)

: rectangle (…, "EdgeColor", ec)

: rectangle (…, "FaceColor", fc)

: rectangle (hax, …)

: h = rectangle (…)

绘制一个由pos和curve定义的矩形patch。

变量pos(1:2)定义了patch的左下角，pos(3:4)定义了它的宽度和高度。缺省情况下，pos的值为[0,0,1,1]。

变量曲线定义了矩形边长的曲率，它可以是一个值在0到1之间的标量或双元素向量。值为0表示这条边没有曲率，而值为1表示这条边完全弯曲成圆的弧线。如果curv是一个双元素向量，那么第一个元素是沿着patch的x轴的曲率，第二个元素是沿着y轴的曲率。

如果curv是一个标量，它表示矩形两边较短一方的曲率，另一方的曲率定义为

min (pos(1:2)) / max (pos(1:2)) \* curv

附加的属性/值对被传递给底层补丁命令。完整的属性列表记录在补丁属性中。

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。

可选的返回值h是创建的矩形对象的图形句柄。

参见:贴片、直线、圆柱体、椭球、球体。

**15.2.2三维绘图**

函数mesh生成网格曲面图。例如,

tx = ty = linspace (-8, 8, 41)';

[xx, yy] = meshgrid (tx, ty);

r = sqrt (xx .^ 2 + yy .^ 2) + eps;

tz = sin (r) ./ r;

mesh (tx, ty, tz);

xlabel ("tx");

ylabel ("ty");

zlabel ("tz");

title ("3-D Sombrero plot");

生成熟悉的“sombrero”图，如图15.5所示。注意使用函数meshgrid来创建X和Y坐标矩阵，以用于绘制Z数据。ndgrid函数类似于meshgrid，但适用于n维矩阵。

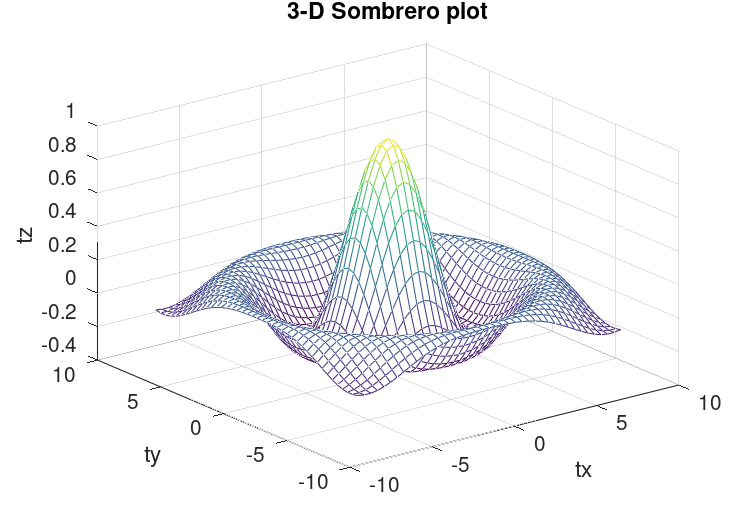


图15.5:网格图。

meshc函数类似于mesh，但也为表面生成轮廓图。

plot3函数显示任意三维数据，而不需要它形成一个曲面。例如,

t = 0:0.1:10\*pi;

r = linspace (0, 1, numel (t));

z = linspace (0, 1, numel (t));

plot3 (r.\*sin (t), r.\*cos (t), z);

xlabel ("r.\*sin (t)");

ylabel ("r.\*cos (t)");

zlabel ("z");

title ("plot3 display of 3-D helix");

显示如图15.6所示的三维螺旋。

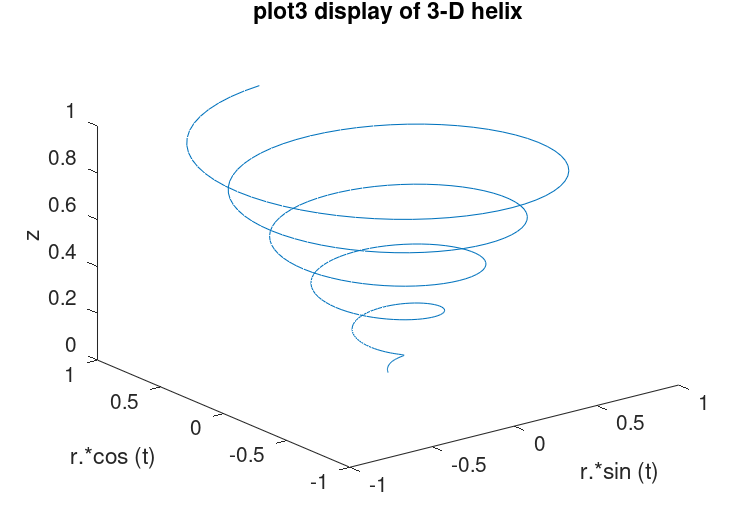


图15.6:三维螺旋。

最后，视图函数改变三维图的视点。

: mesh (x, y, z)

: mesh (z)

: mesh (…, c)

: mesh (…, prop, val, …)

: mesh (hax, …)

: h = mesh (…)

绘制三维线框网格。

线框网格使用矩形绘制。矩形[x, y]的顶点通常是网格的输出。在x-y平面上的二维矩形区域上。Z决定了每个顶点平面以上的高度。如果只给出一个z矩阵，则在网格上绘制x = 1:列(z)， y = 1:行(z)。因此，z的列对应不同的x值，z的行对应不同的y值。

网格的颜色是通过线性缩放z值来计算的，以适应当前颜色图的范围。使用颜色和/或更改颜色映射来控制外观。

可选的是，网格的颜色可以通过提供颜色矩阵c来独立于z指定。

任何属性/值对都直接传递给底层表面对象。完整的属性列表记录在表面属性。

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。

可选的返回值h是创建的表面对象的图形句柄。

参见:ezmesh, meshc, meshz, trimesh，轮廓，冲浪，表面，网格，隐藏，阴影，彩色地图，出租车。

: meshc (x, y, z)

: meshc (z)

: meshc (…, c)

: meshc (…, prop, val, …)

: meshc (hax, …)

: h = meshc (…)

绘制具有底层轮廓线的三维线框网格。

线框网格使用矩形绘制。矩形[x, y]的顶点通常是网格的输出。在x-y平面上的二维矩形区域上。Z决定了每个顶点平面以上的高度。如果只给出一个z矩阵，则在网格上绘制x = 1:列(z)， y = 1:行(z)。因此，z的列对应不同的x值，z的行对应不同的y值。

网格的颜色是通过线性缩放z值来计算的，以适应当前颜色图的范围。使用颜色和/或更改颜色映射来控制外观。

网格的颜色可以通过提供颜色矩阵c来独立于z指定。

任何属性/值对都直接传递给底层表面对象。完整的属性列表记录在表面属性。

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。

可选的返回值h是一个2元素向量，具有创建的表面对象和创建的等高线图的图形句柄。

参见:ezmeshc, mesh, meshz，轮廓，surfc, surface, meshgrid, hidden, shading, colormap，坐标轴。

: meshz (x, y, z)

: meshz (z)

: meshz (…, c)

: meshz (…, prop, val, …)

: meshz (hax, …)

: h = meshz (…)

绘制带有周围窗帘的三维线框网格。

线框网格使用矩形绘制。矩形[x, y]的顶点通常是网格的输出。在x-y平面上的二维矩形区域上。Z决定了每个顶点平面以上的高度。如果只给出一个z矩阵，则在网格上绘制x = 1:列(z)， y = 1:行(z)。因此，z的列对应不同的x值，z的行对应不同的y值。

网格的颜色是通过线性缩放z值来计算的，以适应当前颜色图的范围。使用颜色和/或更改颜色映射来控制外观。

网格的颜色可以通过提供颜色矩阵c来独立于z指定。

任何属性/值对都直接传递给底层表面对象。完整的属性列表记录在表面属性。

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。

可选的返回值h是创建的表面对象的图形句柄。

参见:网格，网格，轮廓，冲浪，表面，瀑布，网格，隐藏，阴影，颜色图，出租车。

: hidden

: hidden on

: hidden off

: mode = hidden (…)

控制网格隐藏线的去除。

当不带参数调用时，将切换隐藏行删除状态。

当以“on”或“off”模式之一调用时，将相应地设置状态。

可选的输出参数模式是当前状态。

隐藏线移除决定网格图后面的图形对象是可见的。默认情况下，网格是不透明的，网格后面的线是不可见的。如果关闭隐藏线移除，那么可以通过网格的面(开口)看到网格后面的物体，尽管网格网格线仍然是不透明的。

参见:mesh, meshc, meshz, ezmesh, ezmeshc, trimesh，瀑布。

: surf (x, y, z)

: surf (z)

: surf (…, c)

: surf (…, prop, val, …)

: surf (hax, …)

: h = surf (…)

绘制三维表面网格。

表面网格使用阴影矩形绘制。矩形[x, y]的顶点通常是网格的输出。在x-y平面上的二维矩形区域上。Z决定了每个顶点平面以上的高度。如果只给出一个z矩阵，则在网格上绘制x = 1:列(z)， y = 1:行(z)。因此，z的列对应不同的x值，z的行对应不同的y值。

表面的颜色是通过线性缩放z值来计算的，以适应当前颜色映射的范围。使用颜色和/或更改颜色映射来控制外观。

可选的是，表面的颜色可以通过提供颜色矩阵c来独立于z指定。

任何属性/值对都直接传递给底层表面对象。完整的属性列表记录在表面属性。

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。

可选的返回值h是创建的表面对象的图形句柄。

注意:表面的确切外观可以通过阴影命令或使用set来控制表面对象属性来控制。

参见:ezsurf, surfc, surfl, surfnorm, trissurf，轮廓，网格，表面，网格，隐藏，阴影，彩色地图，出租车。

: surfc (x, y, z)

: surfc (z)

: surfc (…, c)

: surfc (…, prop, val, …)

: surfc (hax, …)

: h = surfc (…)

绘制具有下面等高线的三维表面网格。

表面网格使用阴影矩形绘制。矩形[x, y]的顶点通常是网格的输出。在x-y平面上的二维矩形区域上。Z决定了每个顶点平面以上的高度。如果只给出一个z矩阵，则在网格上绘制x = 1:列(z)， y = 1:行(z)。因此，z的列对应不同的x值，z的行对应不同的y值。

表面的颜色是通过线性缩放z值来计算的，以适应当前颜色映射的范围。使用颜色和/或更改颜色映射来控制外观。

可选的是，表面的颜色可以通过提供颜色矩阵c来独立于z指定。

任何属性/值对都直接传递给底层表面对象。完整的属性列表记录在表面属性。

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。

可选的返回值h是创建的表面对象的图形句柄。

注意:表面的确切外观可以通过阴影命令或使用set来控制表面对象属性来控制。

参见:ezsurfc, surf, surfl, surfnorm, trissurf，轮廓，网格，表面，网格，隐藏，阴影，彩色地图，出租车。

: surfl (z)

: surfl (x, y, z)

: surfl (…, lsrc)

: surfl (x, y, z, lsrc, P)

: surfl (…, "cdata")

: surfl (…, "light")

: surfl (hax, …)

: h = surfl (…)

使用基于各种照明模型的阴影绘制3d表面。

表面网格使用阴影矩形绘制。矩形[x, y]的顶点通常是网格的输出。在x-y平面上的二维矩形区域上。Z决定了每个顶点平面以上的高度。如果只给出一个z矩阵，则在网格上绘制x = 1:列(z)， y = 1:行(z)。因此，z的列对应不同的x值，z的行对应不同的y值。

默认的照明模式“cdata”，改变表面对象的cdata属性，给人一种被照亮的表面的印象。

交替模式“光”创建一个光对象来照亮表面。

光源位置可以使用lsrc来指定，lsrc可以是2元素向量[方位角，仰角]，也可以是3元素向量[lx, ly, lz]。默认值是逆时针旋转45度到当前视图。

表面的材料属性可以使用4元素向量P = [AM D SP exp]来指定，默认为P =[0.55 0.6 0.4 10]。

"AM" 环境光强度

"D" 漫反射强度

"SP" 镜面反射强度

"EXP" 高光指数

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。

可选的返回值h是创建的表面对象的图形句柄。

例子:

colormap (bone (64));

surfl (peaks);

shading interp;

参见:漫射，镜面，冲浪，阴影，彩色地图，出租车。

: surfnorm (x, y, z)

: surfnorm (z)

: surfnorm (…, prop, val, …)

: surfnorm (hax, …)

: [Nx, Ny, Nz] = surfnorm (…)

求网格表面的法向向量。

如果x和y是向量，那么一个典型的顶点是(x(j)， y(i)， z(i,j))。因此，z的列对应不同的x值，z的行对应不同的y值。如果只给出一个输入z，则取x为1:列(z)， y为1:行(z)。

如果不要求返回参数，则绘制带有曲面法向量的曲面图。

任何属性/值输入对都被分配给表面对象。完整的属性列表记录在表面属性。

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。

如果请求输出参数，则在Nx, Ny和Nz中返回法向量的分量，并且不进行绘图。法向量是未归一化的(幅度!= 1)

len = sqrt (nx.^2 + ny.^2 + nz.^2);

nx ./= len; ny ./= len; nz ./= len;

使用surfnorm的一个例子是

surfnorm (peaks (25));

算法:法向量是通过网格中每个四边形面对角线的叉乘来计算，从而找到每个面中心的法向量。接下来，对于每个网格点，对四个最近的法向量进行平均，以获得网格点处表面的最终法线。

对于表面物体，“VertexNormals”属性包含等效信息，除了可能在表面边界附近，不同的插值方案可能产生略有不同的值。

参见:异型，quiver3, surf，网格。

: fv = isosurface (v, isoval)

: fv = isosurface (v)

: fv = isosurface (x, y, z, v, isoval)

: fv = isosurface (x, y, z, v)

: fvc = isosurface (…, col)

: fv = isosurface (…, "noshare")

: fv = isosurface (…, "verbose")

: [f, v] = isosurface (…)

: [f, v, c] = isosurface (…)

: isosurface (…)

计算三维体数据等值面。

等值面连接具有相同值的点，类似于等高线图，但在三维空间中。

输入参数v是一个三维数组，其中包含在一个卷上采样的数据。

输入等值面是指定等值面的值的标量。如果isoval省略或为空，则从v确定等值面的“好”值。

当使用单个输出参数调用时，isosurface返回一个结构数组fv，其中包含在点[x, y, z] = meshgrid (1:l, 1:m, 1:n)处计算的字段面和顶点，其中[l, m, n] = size (v)。输出fv可以直接用作patch函数的输入。

如果使用额外的输入参数x, y和z(与v大小相同的三维数组或长度与v的维数对应的向量)调用，则在指定点处获取体积数据。如果x、y或z为空，则网格对应于各自方向上的索引(1:n)(参见meshgrid)。

可选的输入参数col是一个与v大小相同的三维数组，它指定等边面的着色。如果有必要，颜色数据会被内插以匹配isoval。在本例中，输出结构数组具有额外的字段facevertexdata。

如果给定字符串输入参数"noshare"，则可以为不同的面返回多次顶点。默认行为是消除相邻面共享的顶点。

为了与MATLAB兼容，支持字符串输入参数"verbose"，但没有效果。

任何字符串参数都必须在其他参数之后传递。

如果使用两个或三个输出参数调用，则将有关面f、顶点v和颜色数据c的信息作为单独的数组返回，而不是单个结构数组。

如果在没有输出参数的情况下调用，则使用patch命令直接绘制等面几何形状，如果还没有出现，则将向轴添加一个光对象。

例如,

[x, y, z] = meshgrid (1:5, 1:5, 1:5);

v = rand (5, 5, 5);

isosurface (x, y, z, v, .5);

将在图形窗口中直接绘制随机等面几何形状。

具有不同附加颜色的等面几何的例子:

N = 15; # Increase number of vertices in each direction

iso = .4; # Change isovalue to .1 to display a sphere

lin = linspace (0, 2, N);

[x, y, z] = meshgrid (lin, lin, lin);

v = abs ((x-.5).^2 + (y-.5).^2 + (z-.5).^2);

figure ();

subplot (2,2,1); view (-38, 20);

[f, vert] = isosurface (x, y, z, v, iso);

p = patch ("Faces", f, "Vertices", vert, "EdgeColor", "none");

pbaspect ([1 1 1]);

isonormals (x, y, z, v, p)

set (p, "FaceColor", "green", "FaceLighting", "gouraud");

light ("Position", [1 1 5]);

subplot (2,2,2); view (-38, 20);

p = patch ("Faces", f, "Vertices", vert, "EdgeColor", "blue");

pbaspect ([1 1 1]);

isonormals (x, y, z, v, p)

set (p, "FaceColor", "none", "EdgeLighting", "gouraud");

light ("Position", [1 1 5]);

subplot (2,2,3); view (-38, 20);

[f, vert, c] = isosurface (x, y, z, v, iso, y);

p = patch ("Faces", f, "Vertices", vert, "FaceVertexCData", c, ...

"FaceColor", "interp", "EdgeColor", "none");

pbaspect ([1 1 1]);

isonormals (x, y, z, v, p)

set (p, "FaceLighting", "gouraud");

light ("Position", [1 1 5]);

subplot (2,2,4); view (-38, 20);

p = patch ("Faces", f, "Vertices", vert, "FaceVertexCData", c, ...

"FaceColor", "interp", "EdgeColor", "blue");

pbaspect ([1 1 1]);

isonormals (x, y, z, v, p)

set (p, "FaceLighting", "gouraud");

light ("Position", [1 1 5]);

参见:等距，等色，等距，平滑，还原进化，还原补丁，补丁。

: vn = isonormals (val, vert)

: vn = isonormals (val, hp)

: vn = isonormals (x, y, z, val, vert)

: vn = isonormals (x, y, z, val, hp)

: vn = isonormals (…, "negate")

: isonormals (val, hp)

: isonormals (x, y, z, val, hp)

: isonormals (…, "negate")

计算等值面的法线。

顶点法线vn是从包含等面几何数据的三维数组val (size: lxmxn)的梯度中计算出来的。法线指向val中较小的值。

如果使用一个输出参数vn调用，并且第二个输入参数vert保存等值面的顶点，则法线vn是在网格上的顶点vert上计算的[x, y, z] = meshgrid (1: 1, 1:m, 1:n)。输出参数vn与vert具有相同的大小，可用于设置相应补丁的“VertexNormals”属性。

如果使用额外的输入参数x、y和z(它们是与val大小相同的三维数组)调用，则在这些点处获取卷数据。可以将补丁句柄hp传递给函数，而不是顶点数据vert。

如果最后一个输入参数是字符串“negate”，计算等面几何的反向向量法线(即指向val中较大的值)。

如果没有给出输出参数，则直接更改与补丁句柄hp关联的补丁的属性“VertexNormals”。

参见:等值面，等值色，平滑。

: fvc = isocaps (v, isoval)

: fvc = isocaps (v)

: fvc = isocaps (x, y, z, v, isoval)

: fvc = isocaps (x, y, z, v)

: fvc = isocaps (…, which\_caps)

: fvc = isocaps (…, which\_plane)

: fvc = isocaps (…, "verbose")

: [faces, vertices, fvcdata] = isocaps (…)

: isocaps (…)

为三维数据的等值面创建端帽。

这个函数将帽放置在等值面的开口端。

输入参数v是一个三维数组，其中包含在一个卷上采样的数据。

输入等值面是指定等值面的值的标量。如果isoval省略或为空，则从v确定等值面的“好”值。

当使用单个输出参数调用时，isocaps返回一个结构数组fvc，其中包含字段:faces, vertex和facevertexdata。结果在点[x, y, z] = meshgrid (1:l, 1:m, 1:n)处计算，其中[l, m, n] = size (v)。输出fvc可以直接用作patch函数的输入。

如果使用额外的输入参数x, y和z(与v大小相同的三维数组或长度与v的维数对应的向量)调用，则在指定点处获取体积数据。如果x、y或z为空，则网格对应于各自方向上的索引(1:n)(参见meshgrid)。

可选参数\_\_\_caps可以是下列字符串值之一，用于定义数据的封装方式:

"above", "a" (default)

对于包含上述数据的端帽。

"below", "b"

用于将数据包含在isoval下面的端帽。

可选形参\_\_\_plane可以有以下字符串值之一，用于定义应该绘制哪个尾盖:

"all" (default)

对于所有的尾帽。

"xmin"

对于数据下x平面的端帽。

"xmax"

对于数据上x平面的端帽。

"ymin"

对于数据的下y平面上的端帽。

"ymax"

对于数据上y平面上的端帽。

"zmin"

对于数据的下z平面的端帽。

"zmax"

对于数据上z平面的端帽。

为了与MATLAB兼容，支持字符串输入参数"verbose"，但没有效果。

如果使用两个或三个输出参数调用，则面部、顶点和颜色数据facevertexdata的数据将以单独的数组返回，而不是单个结构。

如果在不带输出参数的情况下调用，则使用patch命令直接在当前图形中绘制尾盖。

参见:等值面，等值线，贴片。

: cdat = isocolors (c, v)

: cdat = isocolors (x, y, z, c, v)

: cdat = isocolors (x, y, z, r, g, b, v)

: cdat = isocolors (r, g, b, v)

: cdat = isocolors (…, hp)

: isocolors (…, hp)

计算等值面颜色。

如果使用一个输出参数调用，并且第一个输入参数c是一个包含索引颜色值的三维数组，第二个输入参数v是等面几何的顶点，则返回一个矩阵cdat，其中包含计算点[x, y, z] = meshgrid (1: 1, 1:m, 1:n)的几何颜色数据信息。输出参数cdat可用于手动设置等值面补丁对象的“FaceVertexCData”属性。

如果使用额外的输入参数x, y和z(与c大小相同的三维数组)调用，则在这些指定点获取颜色数据。

除了索引颜色数据c，还可以使用RGB值r, g, b来调用isocolors。如果没有给出输入参数x, y, z，则使用网格计算值。

可选地，一个补丁句柄hp可以作为所有函数调用变量的最后一个输入参数，顶点数据将从等面补丁对象中提取出来。最后，如果没有给出输出参数，那么由补丁句柄hp给出的补丁的颜色将被改变。

**参见:**等值面，等值面。

: smoothed\_data = smooth3 (data)

: smoothed\_data = smooth3 (data, method)

: smoothed\_data = smooth3 (data, method, sz)

: smoothed\_data = smooth3 (data, method, sz, std\_dev)

三维矩阵数据的平滑值。

例如，在计算等值面之前，可以使用此函数来减少数据中噪声的影响。

数据必须是一个非单态三维矩阵。输出的smoothed\_data是一个与data大小相同的矩阵。

选项输入法决定使用哪个卷积核进行平滑处理。可能的选择:

"box", "b" (default)

具有尖锐边缘的卷积核。

"gaussian", "g"

由非相关的三元正态分布函数表示的卷积核。

Sz要么是一个3元素向量，指定卷积核在x, y和z方向上的大小，要么是一个标量。在标量情况下，对所有三个维度([sz, sz, sz])使用相同的大小。默认值为3。

如果方法为“高斯”，则可选输入std\_dev定义三元正态分布函数的标准差。std\_dev是一个3元素向量，指定高斯卷积核在x, y和z方向上的标准偏差，或者是一个标量。在标量情况下，对所有三个维度使用相同的值。默认值为0.65。

参见:等值面，等值线，贴片。

: [nx, ny, nz, nv] = reducevolume (v, r)

: [nx, ny, nz, nv] = reducevolume (x, y, z, v, r)

: nv = reducevolume (…)

根据r中的值减少v中数据集的体积。

V是一个在前三维中非单态的矩阵。

R可以是代表x, y和z方向上的还原因子的3个元素的向量，也可以是一个标量，在这种情况下，在所有三个维度上使用相同的还原因子。

Reducevolume通过只取相应维度上的第r个元素来减少v的元素数。

可选地，x, y和z可以用来表示v的坐标集。它们可以是与v大小相同的矩阵，也可以是根据v的维数大小的向量，在这种情况下，它们被扩展为矩阵(参见网格)。

如果使用两个参数调用reducevolume，则假定x、y和z匹配v的各自索引。

简化后的矩阵以nv的形式返回。

可选地，分别以nx、ny和nz的形式返回简化后的坐标集。

例子:

v = reshape (1:6\*8\*4, [6 8 4]);

nv = reducevolume (v, [4 3 2]);

v = reshape (1:6\*8\*4, [6 8 4]);

x = 1:3:24; y = -14:5:11; z = linspace (16, 18, 4);

[nx, ny, nz, nv] = reducevolume (x, y, z, v, [4 3 2]);

参见:等值面，等值面。

: reduced\_fv = reducepatch (fv)

: reduced\_fv = reducepatch (faces, vertices)

: reduced\_fv = reducepatch (patch\_handle)

: reducepatch (patch\_handle)

: reduced\_fv = reducepatch (…, reduction\_factor)

: reduced\_fv = reducepatch (…, "fast")

: reduced\_fv = reducepatch (…, "verbose")

: [reduced\_faces, reduces\_vertices] = reducepatch (…)

在保留patch的整体形状的同时，减少patch对象中的面和顶点的数量。

输入patch可以用一个结构fv来表示，fv带有字段faces和vertex，也可以用两个矩阵faces和vertex来表示(参见，例如，isosurface的结果)，或者用一个patch对象patch\_handle的句柄来表示(参见patch)。

通过迭代地将patch的最短边缘折叠到其中点来减少patch中的面和顶点的数量(如所讨论的，例如，在这里:https://libigl.github.io/libigl/tutorial/tutorial.html#meshdecimation)。

目前，只支持由三角形组成的补丁。生成的补丁也只由三角形组成。

如果调用reducepatch时带有有效补丁patch\_handle的句柄，并且没有任何输出参数，那么给定的补丁将立即更新。

如果省略reduction\_factor，得到的结构reduced\_fv包含原始patch的大约50%的面。如果reduction\_factor是0(排除)和1(排除)之间的分数，则确定具有近似对应的面分数的patch。如果reduction\_factor是一个大于等于1的整数，则得到的patch大约有reduction\_factor面。根据patch的几何形状，得到的面数可能不同于reduction\_factor的给定值。当检测到许多共享顶点时尤其如此。

为了进行约简，有必要使接触面的顶点是共享的。自动检测共享顶点。可以通过传递可选字符串参数"fast"跳过此检测。

使用可选的字符串参数"verbose"，将额外的状态消息打印到命令窗口。

任何字符串输入参数必须在所有其他参数之后传递。

如果使用一个输出参数调用，则减少的面和顶点将在结构reduced\_fv中返回，其中包含字段faces和vertices(请参阅isosurface的一个输出选项)。

如果使用两个输出参数调用，则减少的面和顶点将在两个单独的矩阵reduced\_faces和reduced\_vertices中返回。

参见:等值面，等值面，还原，补丁。

: shrinkfaces (p, sf)

: nfv = shrinkfaces (p, sf)

: nfv = shrinkfaces (fv, sf)

: nfv = shrinkfaces (f, v, sf)

: [nf, nv] = shrinkfaces (…)

通过收缩因子sf减小贴片中面的大小。

patch对象可以通过图形句柄(p)，带有“faces”和“vertex”字段的patch结构(fv)，或者作为两个单独的面和顶点矩阵(f, v)来指定。

收缩因子sf是一个正数，指定新面将占用的原始面积的百分比。如果没有给定因子，则默认值为0.3(减少到原始大小的30%)。大于1.0的因子将导致面扩展。

给定一个补丁句柄作为第一个输入参数，并且没有输出参数，执行补丁面缩小并重新绘制补丁。

如果使用一个输出参数调用，则返回一个包含字段“faces”，“vertices”和“facevertexdata”的结构，其中包含收缩后的数据。这个结构可以直接用作patch函数的输入参数。

注意:对非凸面执行收缩操作可能会导致不良结果。

例如:一个三角形的3/4圆和相应的缩小版。

[phi r] = meshgrid (linspace (0, 1.5\*pi, 16), linspace (1, 2, 4));

tri = delaunay (phi(:), r(:));

v = [r(:).\*sin(phi(:)) r(:).\*cos(phi(:))];

clf ()

p = patch ("Faces", tri, "Vertices", v, "FaceColor", "none");

fv = shrinkfaces (p);

patch (fv)

axis equal

grid on

参见:patch。

: d = diffuse (sx, sy, sz, lv)

计算由法向量元素sx, sy, sz定义的表面的漫反射强度。

光源位置向量lv可以用2元矢量[方位角，仰角]表示，也可以用3元矢量[x, y, z]表示。

参见:specular, surfl。

: refl = specular (sx, sy, sz, lv, vv)

: refl = specular (sx, sy, sz, lv, vv, se)

用Phong的近似计算法向量元素sx, sy, sz定义的表面的镜面反射强度。

光源位置和观察者位置向量分别使用参数lv和vv指定。位置向量可以用2元素向量[方位角，仰角]表示，也可以用3元素向量[x, y, z]表示。

可选的第六个参数指定镜面指数(扩散)se。如果没有给出，se默认为10。

参见:diffuse, surfl。

: lighting (type)

: lighting (hax, type)

设置贴片或表面图形对象的照明。

type的有效参数是

"flat"

绘制具有切面照明效果的对象。

"gouraud"

用顶点间照明效果的线性插值绘制对象。

"none"

绘制没有光影效果的物体。

如果第一个参数hax是一个轴句柄，那么改变这个轴中对象的照明效果，而不是gca返回的当前轴。

只有当至少有一个光对象存在并且在相同的轴上可见时，照明效果才可见。

参见:光，填充，网格，补丁，pcolor，冲浪，表面，阴影。

: material shiny

: material dull

: material metal

: material default

: material ([as, ds, ss])

: material ([as, ds, ss, se])

: material ([as, ds, ss, se, scr])

: material (hlist, …)

: mtypes = material ()

: refl\_props = material (mtype\_string)

设置表面和补丁照明的反射率属性。

该函数改变当前轴上所有斑块和表面物体的环境、漫反射和镜面强度，以及镜面指数和镜面颜色反射率。在某种程度上，这可以用来模拟某些材料与光一起使用时的反射特性。

当使用字符串调用时，上述属性将根据下表中的值进行设置:

|  | ***mtype*** | **ambient- strength** | **diffuse- strength** | **specular- strength** | **specular- exponent** | **specular- color- reflectance** |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | *"shiny"* | 0.3 | 0.6 | 0.9 | 20 | 1.0 |  |
|  | *"dull"* | 0.3 | 0.8 | 0.0 | 10 | 1.0 |  |
|  | *"metal"* | 0.3 | 0.3 | 1.0 | 25 | 0.5 |  |
|  | *"default"* | *"default"* | *"default"* | *"default"* | *"default"* | *"default"* |  |

当使用三个元素的向量调用时，当前轴上所有补丁和表面物体的环境、漫反射和镜面强度都会更新。可选的第四个矢量元素更新镜面指数，可选的第五个矢量元素更新镜面颜色反射率。

图形句柄列表也可以作为第一个参数传递。在这种情况下，这些句柄和所有子补丁和表面对象的属性将被更新。

此外，可以使用单个输出参数调用material。如果在没有输入参数的情况下调用，则返回包含所有可用材料字符串的列单元向量mtypes。如果一个输入参数mtype\_string是材质的名称，则返回一个带有该材质反射属性的1x5单元向量refl\_props。在这两种情况下，都不会改变图形属性。

参见:光，填充，网格，补丁，pcolor，冲浪，表面。

: camlight

: camlight right

: camlight left

: camlight headlight

: camlight (az, el)

: camlight (…, style)

: camlight (hl, …)

: camlight (hax, …)

: h = camlight (…)

使用简单的界面为图形添加一个光对象。

当不带参数调用时，一个light对象被添加到当前绘图中，并被放置在相机当前位置的稍上方和右侧:这相当于camlight right。命令camlight left和camlight headlight的行为类似，放置在相机位置的左侧或相机位置的中心。

对于更多的控制，光的位置可以通过方位旋转az和仰角el来指定，两者都以度为单位，相对于相机的当前属性。

可选的字符串样式指定光源是局部点源(“local”，默认)还是放置在无限距离上(“infinite”)。

如果第一个参数hl是一个light对象的句柄，那么对这个light对象进行操作，而不是创建一个新对象。

如果第一个参数hax是一个轴句柄，那么在这个轴上创建一个新的light对象，而不是gca返回的当前轴。

可选的返回值h是灯光对象的图形句柄。这可以用来移动或进一步改变光对象的属性。

例子:

添加一个光对象到一个情节

sphere (36);

camlight

准确定位光源

camlight (45, 30);

在这里，光首先从相机位置向上倾斜(见camup)(见campos) 30度。然后向右偏航45度。两个旋转都以相机目标为中心(参见相机目标)。

返回一个句柄以进一步操作light对象

clf

sphere (36);

hl = camlight ("left");

set (hl, "color", "r");

参见:光。

: lightangle (az, el)

: lightangle (hax, az, el)

: lightangle (hl, az, el)

: hl = lightangle (…)

: [az, el] = lightangle (hl)

使用球坐标在当前轴上添加一个光对象。

光的位置由方位旋转az和仰角el指定，两者都以度为单位。

如果第一个参数hax是一个轴句柄，那么在这个轴上创建一个新的light对象，而不是gca返回的当前轴。

如果第一个参数hl是一个light对象的句柄，那么对这个light对象进行操作，而不是创建一个新对象。

可选的返回值hl是灯光对象的图形句柄。

例子:

添加一个光对象到一个情节

clf;

sphere (36);

lightangle (45, 30);

参见:光，视图，camlight。

: [xx, yy] = meshgrid (x, y)

: [xx, yy, zz] = meshgrid (x, y, z)

: [xx, yy] = meshgrid (x)

: [xx, yy, zz] = meshgrid (x)

给定向量x和y坐标，返回矩阵xx和yy对应于一个完整的二维网格。

xx的行是x的副本，yy的列是y的副本。如果省略y，则假定它与x相同。

如果给出了可选的z输入，或者请求了zz，那么输出将是一个完整的3-D网格。如果省略了z，并且请求了zz，则假定它与y相同。

meshgrid最常用于生成将要绘制的2-D或3-D函数的输入。下面的示例创建了“sombrero”函数的曲面图。

f = @(x,y) sin (sqrt (x.^2 + y.^2)) ./ sqrt (x.^2 + y.^2);

range = linspace (-8, 8, 41);

[X, Y] = meshgrid (range, range);

Z = f (X, Y);

surf (X, Y, Z);

编程注意:网格只能生成2-D或3-D网格。ndgrid函数将通过N-D生成1-D网格。然而，这些函数并不是完全等价的。如果x是一个长度为M的向量，y是一个长度为N的向量，那么meshgrid将产生一个输出网格，它是NxM。对于相同的输入，ndgrid将产生一个MxN(转置)的输出。一些核心函数期望网格输入，而另一些期望和网格输入。检查相关函数的文档，以确定正确的输入格式。

参见:网格，网格，轮廓，冲浪。

: [y1, y2, …, yn] = ndgrid (x1, x2, …, xn)

: [y1, y2, …, yn] = ndgrid (x)

给定n个向量x1，…，xn, ndgrid返回n个维数为n的数组。

第i个输出参数的元素包含向量xi在不同于第i维的所有维度上重复的元素。调用只有一个输入参数x的ndgrid相当于调用所有n个输入参数都等于x的ndgrid:

[y1, y2, …, yn] = ndgrid (x, …, x)

编程注意:ndgrid与函数meshgrid非常相似，除了前两个维度与meshgrid相比是调换的。一些核心函数期望网格输入，而另一些期望和网格输入。检查相关函数的文档，以确定正确的输入格式。

参见:meshgrid。

: plot3 (x, y, z)

: plot3 (x, y, z, prop, value, …)

: plot3 (x, y, z, fmt)

: plot3 (x, cplx)

: plot3 (cplx)

: plot3 (hax, …)

: h = plot3 (…)

生成三维图形。

许多不同的参数组合是可能的。最简单的形式是

plot3 (x, y, z)

其中参数被取为要在三维空间中绘制的点的顶点。如果所有参数都是相同长度的向量，则绘制一条连续的线。如果所有参数都是矩阵，则的每一列都被视为单独的一行。不会尝试对参数进行转置以使行数匹配。

如果只给出两个参数，则为

plot3 (x, cplx)

第二个参数的实部和虚部分别用作y和z坐标。

如果只给出一个参数，as

plot3 (cplx)

参数的实部和虚部用作y和z值，并根据它们的索引绘制它们。

论据也可以以三人为一组给出

plot3 (x1, y1, z1, x2, y2, z2, …)

其中，三个参数的每一组都被视为三维中的一个单独的行或一组行。

若要绘制多个单参数或双参数组，请使用空格式字符串分隔每个组，如

plot3 (x1, c1, "", c2, "", …)

可以指定多个属性值对，这将影响plot3绘制的线对象。如果提供了fmt参数，它将以与plot相同的方式格式化行对象。完整的属性列表记录在Line properties中。

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。

可选的返回值h是创建的绘图的图形句柄。

例子:

z = [0:0.05:5];

plot3 (cos (2\*pi\*z), sin (2\*pi\*z), z, ";helix;");

plot3 (z, exp (2i\*pi\*z), ";complex sinusoid;");

参见:ezplot3, plot。

: view (azimuth, elevation)

: view ([azimuth elevation])

: view ([x y z])

: view (2)

: view (3)

: view (hax, …)

: [azimuth, elevation] = view ()

查询或设置当前轴的视点。

参数方位角和仰角可以作为两个参数或作为2元素向量给出。视点也可以用笛卡尔坐标x、y和z来指定。

调用视图(2)将视点设置为方位角= 0和仰角= 90，这是二维图形的默认值。

调用视图(3)将视点设置为方位角= -37.5和仰角= 30，这是3- d图形的默认值。

如果第一个参数hax是一个轴句柄，那么操作这个轴，而不是gca返回的当前轴。

如果没有输入，则返回当前的方位角和高度。

: camlookat ()

: camlookat (h)

: camlookat (handle\_list)

: camlookat (hax)

移动相机并调整其属性以查看对象。

当输入为手柄h时，将摄像机设置为指向h的边界框的中心。调整摄像机的位置，使边界框大致填充视场。

此命令修复相机的观看方向(camtarget() - campos())，相机向上矢量(见camup)和视角(见camva)。摄像机目标(参见camtarget)和摄像机位置(参见campos)发生了变化。

如果参数是一个列表handle\_list，那么计算所有对象的单一边界框，然后如上所述调整相机。

如果参数是一个轴对象hax，则轴的子节点被用作handle\_list。在没有输入的情况下调用时，它使用当前轴(参见gca)。

参见:camorbit, camzoom, camroll。

: p = campos ()

: campos ([x y z])

: mode = campos ("mode")

: campos (mode)

: campos (hax, …)

获取或设置摄像机位置。

默认相机位置是根据场景自动确定的。例如，要获取相机位置:

hf = figure();

peaks()

p = campos ()

⇒ p =

-27.394 -35.701 64.079

然后我们可以在z轴上移动摄像机:

campos (p + [0 0 10])

campos ()

⇒ ans =

-27.394 -35.701 74.079

做了这个改变，相机位置模式现在是手动的:

campos ("mode")

⇒ manual

我们可以将其设置回自动:

campos ("auto")

campos ()

⇒ ans =

-27.394 -35.701 64.079

close (hf)

默认情况下，这些命令影响当前轴;或者，可以通过可选参数hax指定轴。

参见:camup, camtarget, camva。

: camorbit (theta, phi)

: camorbit (theta, phi, coorsys)

: camorbit (theta, phi, coorsys, dir)

: camorbit (theta, phi, "data")

: camorbit (theta, phi, "data", "z")

: camorbit (theta, phi, "data", "x")

: camorbit (theta, phi, "data", "y")

: camorbit (theta, phi, "data", [x y z])

: camorbit (theta, phi, "camera")

: camorbit (hax, …)

围绕目标上下左右旋转相机。

将相机向上移动1度，向右移动1度，就好像它在围绕目标的轨道上一样。例子:

sphere ()

camorbit (30, 20)

这些旋转以相机目标为中心(参见camtarget)。首先，相机的位置是向上或向下倾斜，通过旋转它phi度围绕一个垂直于观看方向的轴(特别是camtarget() - campos())和相机“向上矢量”(见camup)。例子:

camorbit (0, 20)

第二次旋转取决于坐标系坐标和方向输入。coorsys的默认值是"data"。在这种情况下，相机通过绕dir指定的轴旋转θ度来向左或向右偏航。dir的默认值是"z"，对应于向量[0,0,1]。例子:

camorbit (30, 0)

当coorsys设置为“camera”时，通过旋转与相机向上矢量平行的轴来左右移动相机(参见camup)。在这种情况下不应该指定输入目录。例子:

camorbit (30, 0, "camera")

(注意:旋转phi不受“camera”的影响。)

camorbit命令修改两个相机属性:campos和camup。

默认情况下，此命令影响当前轴;或者，可以通过可选参数hax指定轴。

参见:camzoom, camroll, camlook。

: camroll (theta)

: camroll (hax, theta)

转动相机。

将相机顺时针转动θ度。例如，下面的命令将使相机顺时针旋转30度(向右);这将导致场景看起来向左滚动30度:

peaks ()

camroll (30)

将相机向后转动:

camroll (-30)

恢复默认相机胶卷的命令如下:

camup ("auto")

默认情况下，这些命令影响当前轴;或者，可以通过可选参数hax指定轴。

参见:camzoom, camorbit, camlook, camup。

: t = camtarget ()

: camtarget ([x y z])

: mode = camtarget ("mode")

: camtarget (mode)

: camtarget (hax, …) ¶

获取或设置相机指向的位置。

相机目标是相机所指向的空间中的一个点。通常是根据场景自动确定的:

hf = figure();

sphere (36)

v = camtarget ()

⇒ v =

0 0 0

我们可以把摄像机转向一个新的目标:

camtarget ([1 1 1])

camtarget ()

⇒ 1 1 1

这样，相机的目标模式为手动:

camtarget ("mode")

⇒ manual

这意味着，例如，向场景中添加新对象不会重新瞄准相机:

hold on;

peaks ()

camtarget ()

⇒ 1 1 1

我们可以将其重置为自动:

camtarget ("auto")

camtarget ()

⇒ 0 0 0.76426

close (hf)

默认情况下，这些命令影响当前轴;或者，可以通过可选参数hax指定轴。

参见:campos, camup, camva。

: up = camup ()

: camup ([x y z])

: mode = camup ("mode")

: camup (mode)

: camup (hax, …)

获取或设置相机矢量。

默认情况下，相机的方向使“上”对应于正z轴:

hf = figure ();

sphere (36)

v = camup ()

⇒ v =

0 0 1

指定一个新的“向上矢量”来滚动相机并将模式设置为手动:

camup ([1 1 0])

camup ()

⇒ 1 1 0

camup ("mode")

⇒ manual

修改上矢量不会修改相机目标(参见相机目标)。因此，相机向上向量可能与相机视图的方向不正交:

camup ([1 2 3])

dot (camup (), camtarget () - campos ())

⇒ 6...

结果是，向上矢量的“向后拉”并不会改变相机视图(因为这需要改变目标)。因此，设置矢量通常只用于滚动相机。更直观的命令是camroll。

最后，我们可以将向上向量重置为自动模式:

camup ("auto")

camup ()

⇒ 0 0 1

close (hf)

默认情况下，这些命令影响当前轴;或者，可以通过可选参数hax指定轴。

参见:campos, camtarget, camva。

: a = camva ()

: camva (a)

: mode = camva ("mode")

: camva (mode)

: camva (hax, …)

获取或设置相机视角。

摄像机有一个视角，它决定了可以看到多少。默认情况下是:

hf = figure();

sphere (36)

a = camva ()

⇒ a = 10.340

为了获得更广的视角，我们可以将视角扩大一倍。这也会将模式设置为手动:

camva (2\*a)

camva ("mode")

⇒ manual

我们可以将其设置回自动:

camva ("auto")

camva ("mode")

⇒ auto

camva ()

⇒ ans = 10.340

close (hf)

默认情况下，这些命令影响当前轴;或者，可以通过可选参数hax指定轴。

参见:campos, camtarget, camup。

: camzoom (zf)

: camzoom (hax, zf)

把相机放大或缩小。

大于1的zf值“放大”，使场景看起来被放大:

hf = figure ();

sphere (36)

camzoom (1.2)

小于1的值“缩小”，这样相机可以看到更多的场景:

camzoom (0.5)

从技术上讲，变焦会影响“视角”。以下命令重置为默认缩放:

camva ("auto")

close (hf)

默认情况下,这些命令会影响当前的axis;或者,一个axis可以由可选的参数hax指定。

参见:camroll,cam的轨道,camlookat,camva。

: slice (x, y, z, v, sx, sy, sz)

: slice (x, y, z, v, xi, yi, zi)

: slice (v, sx, sy, sz)

: slice (v, xi, yi, zi)

: slice (…, method)

: slice (hax, …)

: h = slice (…)

绘制三维数据/标量场切片。

三维数组v的每个元素在参数x、y和z给出的位置上表示一个标量值。参数x、y和z要么是与“网格”格式的数组v大小相同的三维数组，要么是矢量。参数xi等遵循与x等类似的格式，它们表示使用interp3对数组vi进行插值的点。向量sx, sy和sz包含各自轴的正交切片的点。

如果省略x, y, z，则假设它们为x = 1:size (v, 2)， y = 1:size (v, 1)和z = 1:size (v, 3)。

方法是:

"nearest"

返回最近的邻居。

"linear"

从最近邻的线性插值。

"cubic"

从四个最近邻的三次插值(尚未实现)。

"spline"

三次样条插值-平滑的一阶和二阶导数在整个曲线。

默认方法是“linear”。

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。

可选的返回值h是创建的表面对象的图形句柄。

例子:

[x, y, z] = meshgrid (linspace (-8, 8, 32));

v = sin (sqrt (x.^2 + y.^2 + z.^2)) ./ (sqrt (x.^2 + y.^2 + z.^2));

slice (x, y, z, v, [], 0, []);

[xi, yi] = meshgrid (linspace (-7, 7));

zi = xi + yi;

slice (x, y, z, v, xi, yi, zi);

参见:interp3, surface, pcolor。

: ribbon (y)

: ribbon (x, y)

: ribbon (x, y, width)

: ribbon (hax, …)

: h = ribbon (…)

为y与x的列绘制带状图。

如果省略x，则假定一个包含行号的向量(1:行(Y))。或者，x也可以是具有与y的行相同数量的元素的向量，在这种情况下，对y的每一列使用相同的x。

可选参数width指定单个ribbon的宽度(默认为0.75)。

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。

可选的返回值h是表示每个色带的表面对象的图形句柄向量。

参见:水面、瀑布。

: shading (type)

: shading (hax, type)

设置贴片或表面图形对象的阴影。

type的有效参数是

"flat"

带有不可见边缘的单色斑块。

"faceted"

带有黑色边缘的单色斑块。

"interp"

patch顶点之间的颜色被插值，patch边缘是不可见的。

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:填充，网格，补丁，pcolor，冲浪，表面，隐藏，照明。

: scatter3 (x, y, z)

: scatter3 (x, y, z, s)

: scatter3 (x, y, z, s, c)

: scatter3 (…, style)

: scatter3 (…, "filled")

: scatter3 (…, prop, val)

: scatter3 (hax, …)

: h = scatter3 (…)

绘制三维散点图。

在由向量x、y和z中的坐标定义的每个点上绘制一个标记。

标记的大小由s决定，s可以是一个标量，也可以是与x、y和z长度相同的向量。如果s没有给出，或者是一个空矩阵，则使用默认值8点。

标记的颜色由c决定，c可以是定义固定颜色的字符串;一个3元素向量，表示颜色的红、绿、蓝成分;一个与x长度相同的向量，给出当前颜色图的缩放索引;或定义每个标记的RGB颜色的Nx3矩阵。

可以使用style参数更改要使用的标记，这是一个以与plot命令相同的方式定义标记的字符串。如果没有指定标记，则默认为“0”或圆形。如果给出了参数" filling "，则标记被填充。

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。

可选的返回值h是表示点的散射对象的图形句柄。

[x, y, z] = peaks (20);

scatter3 (x(:), y(:), z(:), [], z(:));

编程注意:属性的完整列表记录在Scatter properties。

参见:scatter, patch, plot。

: waterfall (x, y, z)

: waterfall (z)

: waterfall (…, c)

: waterfall (…, prop, val, …)

: waterfall (hax, …)

: h = waterfall (…)

绘制一个3d瀑布图。

瀑布图类似于网格图，除了只显示z (x值)行的网格线。

线框网格使用矩形绘制。矩形[x, y]的顶点通常是网格的输出。在x-y平面上的二维矩形区域上。Z决定了每个顶点平面以上的高度。如果只给出一个z矩阵，则在网格上绘制x = 1:列(z)， y = 1:行(z)。因此，z的列对应不同的x值，z的行对应不同的y值。

网格的颜色是通过线性缩放z值来计算的，以适应当前颜色图的范围。使用颜色和/或更改颜色映射来控制外观。

网格的颜色可以通过提供颜色矩阵c来独立于z指定。

任何属性/值对都直接传递给底层表面对象。完整的属性列表记录在表面属性。

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。

可选的返回值h是创建的表面对象的图形句柄。

另见:网格，网格，网格，轮廓，冲浪，表面，丝带，网格，隐藏，阴影，颜色图，出租车。

**15.2.2.1纵横比**

对于三维绘图，可以为带有daspect的数据设置宽高比，也可以为带有paspect的绘图框设置宽高比。请参阅轴配置，以控制绘图的x、y和z限制。

: data\_aspect\_ratio = daspect ()

: daspect (data\_aspect\_ratio)

: daspect (mode)

: data\_aspect\_ratio\_mode = daspect ("mode")

: daspect (hax, …)

查询或设置当前轴的数据纵横比。

纵横比是一个规范化的3元素向量，表示x、y和z轴限制的跨度。

daspect (mode)

设置当前轴的数据纵横比模式。模式为“自动”或“手动”。

daspect ("mode")

返回当前轴的数据纵横比模式。

daspect (hax, …)

操作手柄x中的轴，而不是当前的轴。

参见:axis, pbaspect, xlim, ylim, zlim。

: plot\_box\_aspect\_ratio = pbaspect ( )

: pbaspect (plot\_box\_aspect\_ratio)

: pbaspect (mode)

: plot\_box\_aspect\_ratio\_mode = pbaspect ("mode")

: pbaspect (hax, …)

查询或设置当前轴的绘图框宽高比。

纵横比是一个规范化的3元素向量，表示x、y和z轴的呈现长度。

pbaspect(mode)

设置当前轴的绘图框长宽比模式。模式为“自动”或“手动”。

pbaspect ("mode")

返回当前轴的绘图框长宽比模式。

pbaspect (hax, …)

操作手柄x中的轴，而不是当前的轴。

参见:axis, daspect, xlim, ylim, zlim。

15.2.2.2三维函数绘图

: ezplot3 (fx, fy, fz)

: ezplot3 (…, dom)

: ezplot3 (…, n)

: ezplot3 (…, "animate")

: ezplot3 (hax, …)

: h = ezplot3 (…)

在三维空间绘制一条参数化定义的曲线。

Fx、fy和fz是字符串、内联函数或函数句柄，带有一个定义函数的参数。默认情况下，图在0 <= t <= 2\*pi的域上，有500个点。

如果dom是一个双元素向量，它表示t的最小值和最大值。

N是定义用于绘制函数的点的数量的标量。

如果选择“animate”选项，则绘图将以comet3的样式动画化。

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。

可选的返回值h是创建的绘图的图形句柄。

fx = @(t) cos (t);

fy = @(t) sin (t);

fz = @(t) t;

ezplot3 (fx, fy, fz, [0, 10\*pi], 100);

参见:plot3, comet3, ezplot, ezmesh, ezsurf。

: ezmesh (f)

: ezmesh (fx, fy, fz)

: ezmesh (…, dom)

: ezmesh (…, n)

: ezmesh (…, "circ")

: ezmesh (hax, …)

: h = ezmesh (…)

绘制由函数定义的网格。

F是一个字符串、内联函数或函数句柄，带有定义函数的两个参数。默认情况下，图在网格域-2\*pi <= x | y <= 2\*pi上，每个维度上有60个点。

如果传递了三个函数，则绘制参数化定义的函数[fx(s, t)， fy(s, t)， fz(s, t)]。

如果dom是一个双元素向量，它表示x和y的最小值和最大值。如果dom是一个四元素向量，那么最小值和最大值是[xmin xmax ymin ymax]。

N是定义在每个维度中使用的点的数量的标量。

如果给出了参数“circ”，则该函数将绘制在以域dom中间为中心的磁盘上。

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。

可选的返回值h是创建的表面对象的图形句柄。

例1:两个参数的函数

f = @(x,y) sqrt (abs (x .\* y)) ./ (1 + x.^2 + y.^2);

ezmesh (f, [-3, 3]);

例2:参数定义函数

fx = @(s,t) cos (s) .\* cos (t);

fy = @(s,t) sin (s) .\* cos (t);

fz = @(s,t) sin (t);

ezmesh (fx, fy, fz, [-pi, pi, -pi/2, pi/2], 20);

另见:网格，ezmeshc, ezplot, ezsurf, ezsurfc，隐藏。

: ezmeshc (f)

: ezmeshc (fx, fy, fz)

: ezmeshc (…, dom)

: ezmeshc (…, n)

: ezmeshc (…, "circ")

: ezmeshc (hax, …)

: h = ezmeshc (…)

绘制由函数定义的网格和等高线。

F是一个字符串、内联函数或函数句柄，带有定义函数的两个参数。默认情况下，图在网格域-2\*pi <= x | y <= 2\*pi上，每个维度上有60个点。

如果传递了三个函数，则绘制参数化定义的函数[fx(s, t)， fy(s, t)， fz(s, t)]。

如果dom是一个双元素向量，它表示x和y的最小值和最大值。如果dom是一个四元素向量，那么最小值和最大值是[xmin xmax ymin ymax]。

N是定义在每个维度中使用的点的数量的标量。

如果给出了参数“circ”，则该函数将绘制在以域dom中间为中心的磁盘上。

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。

可选的返回值h是一个2元素向量，具有用于创建的网格图的图形句柄和用于创建的等高线图的第二个句柄。

示例:2参数函数

f = @(x,y) sqrt (abs (x .\* y)) ./ (1 + x.^2 + y.^2);

ezmeshc (f, [-3, 3]);

参见：meshc, ezmesh, ezplot, ezsurf, ezsurfc, hidden。

: ezsurf (f)

: ezsurf (fx, fy, fz)

: ezsurf (…, dom)

: ezsurf (…, n)

: ezsurf (…, "circ")

: ezsurf (hax, …)

: h = ezsurf (…)

绘制由函数定义的曲面。

F是一个字符串、内联函数或函数句柄，带有定义函数的两个参数。默认情况下，图在网格域-2\*pi <= x | y <= 2\*pi上，每个维度上有60个点。

如果传递了三个函数，则绘制参数化定义的函数[fx(s, t)， fy(s, t)， fz(s, t)]。

如果dom是一个双元素向量，它表示x和y的最小值和最大值。如果dom是一个四元素向量，那么最小值和最大值是[xmin xmax ymin ymax]。

N是定义在每个维度中使用的点的数量的标量。

如果给出了参数“circ”，则该函数将绘制在以域dom中间为中心的磁盘上。

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。

可选的返回值h是创建的表面对象的图形句柄。

例1:两个参数的函数

f = @(x,y) sqrt (abs (x .\* y)) ./ (1 + x.^2 + y.^2);

ezsurf (f, [-3, 3]);

例2:参数定义函数

fx = @(s,t) cos (s) .\* cos (t);

fy = @(s,t) sin (s) .\* cos (t);

fz = @(s,t) sin (t);

ezsurf (fx, fy, fz, [-pi, pi, -pi/2, pi/2], 20);

参见：冲浪，ezsurfc, ezplot, ezmesh, ezmeshc，着色。

: ezsurfc (f)

: ezsurfc (fx, fy, fz)

: ezsurfc (…, dom)

: ezsurfc (…, n)

: ezsurfc (…, "circ")

: ezsurfc (hax, …)

: h = ezsurfc (…)

绘制由函数定义的曲面和等高线。

F是一个字符串、内联函数或函数句柄，带有定义函数的两个参数。默认情况下，图在网格域-2\*pi <= x | y <= 2\*pi上，每个维度上有60个点。

如果传递了三个函数，则绘制参数化定义的函数[fx(s, t)， fy(s, t)， fz(s, t)]。

如果dom是一个双元素向量，它表示x和y的最小值和最大值。如果dom是一个四元素向量，那么最小值和最大值是[xmin xmax ymin ymax]。

N是定义在每个维度中使用的点的数量的标量。

如果给出了参数“circ”，则该函数将绘制在以域dom中间为中心的磁盘上。

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。

可选的返回值h是一个2元素向量，其中一个图形句柄用于创建的表面图，另一个句柄用于创建的等高线图。

例子:

f = @(x,y) sqrt (abs (x .\* y)) ./ (1 + x.^2 + y.^2);

ezsurfc (f, [-3, 3]);

参见：冲浪，ezsurf, ezplot, ezmesh, ezmeshc，着色。

**15.2.2.3三维几何形状**

: cylinder

: cylinder (r)

: cylinder (r, n)

: cylinder (hax, …)

: [x, y, z] = cylinder (…)

绘制一个三维单位圆柱体。

可选输入r是一个矢量，指定沿单位z轴的半径。默认值为[1 1]，表示在Z == 0和Z == 1处的半径为1。

可选输入n决定圆柱体圆周周围的面数。缺省值为20。

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。

如果请求输出，则圆柱体以网格格式返回三个矩阵，这样surf (x, y, z)生成一个单位圆柱体。

例子:

[x, y, z] = cylinder (10:-1:0, 50);

surf (x, y, z);

title ("a cone");

参见:椭球，矩形，球体。

: sphere ()

: sphere (n)

: sphere (hax, …)

: [x, y, z] = sphere (…)

绘制一个三维单位球体。

可选输入n决定球体周长周围的面数。缺省值为20。

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。

如果输出被请求，球面返回三个网格格式的矩阵，使得surf (x, y, z)生成一个单位球面。

例子:

[x, y, z] = sphere (40);

surf (3\*x, 3\*y, 3\*z);

axis equal;

title ("sphere of radius 3");

参见:圆柱体、椭球体、矩形。

: ellipsoid (xc, yc, zc, xr, yr, zr)

: ellipsoid (…, n)

: ellipsoid (hax, …)

: [x, y, z] = ellipsoid (…)

绘制一个三维椭球体。

输入xc, yc, zc指定椭球体的中心。输入xr, yr, zr表示半长轴长度。

可选输入n决定圆柱体圆周周围的面数。缺省值为20。

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。

如果输出被请求，椭球体返回三个网格格式的矩阵，这样surf (x, y, z)生成椭球体。

参见:圆柱体，矩形，球体。

**15.2.3情节注解**

可以向现有绘图添加标题、轴标签、图例和任意文本。例如:

x = -10:0.1:10;

plot (x, sin (x));

title ("sin(x) for x = -10:0.1:10");

xlabel ("x");

ylabel ("sin (x)");

text (pi, 0.7, "arbitrary text");

legend ("sin (x)");

函数grid和box也可用于向绘图中添加网格和边框线。默认情况下，网格是关闭的，边线是打开的。

最后，可以使用注释功能添加箭头、文本和矩形或椭圆框，以突出显示绘图的某些部分。这些对象绘制在一个看不见的坐标轴上，在其他坐标轴上。

: title (string)

: title (string, prop, val, …)

: title (hax, …)

: h = title (…)

指定用作当前轴标题的字符串。

可以使用一个可选的属性/值对列表来更改所创建的标题文本对象的外观。

如果第一个参数hax是一个轴或图例句柄，那么为这个对象添加一个标题，而不是gca返回的当前轴。

可选的返回值h是创建的文本对象的图形句柄。

参见:xlabel, ylabel, zlabel, text。

: legend ()

: legend command

: legend (str1, str2, …)

: legend (charmat)

: legend ({cellstr})

: legend (…, property, value, …)

: legend (hobjs, …)

: legend ("command")

: legend (hax, …)

: legend (hleg, …)

: hleg = legend (…)

使用指定字符串作为标签显示当前轴的图例。

图例条目可以指定为单个字符串参数、字符数组或字符串单元数组。当标签名称可能与图例属性或命令参数混淆时，应该通过将标签指定为字符串的单元格数组来保护标签。

如果第一个参数hax是一个轴句柄，那么向这个轴添加一个图例，而不是gca返回的当前轴。

如果第一个参数hleg是图例句柄，则操作该图例，而不是当前轴的图例。

图例标签与坐标轴的子轴相关联;第一个标签分配给在坐标轴上绘制的第一个对象，第二个标签分配给下一个绘制的对象，依此类推。要标记特定的数据对象，而不标记所有对象，请在输入hobjs中提供它们的图形句柄。

使用命令可以进行以下自定义:

"show"

在情节上显示图例

"hide"

隐藏剧情上的传说

"toggle"

在“隐藏”和“显示”之间切换

"boxon"

在图例周围显示一个框(默认)

"boxoff"

隐藏框周围的传奇

"right"

将标签文本放在键的右侧(默认)

"left"

在键的左边放置标签文本

"off"

删除图例对象

图例函数创建一个图形对象，该对象具有可以通过get/set操作的各种属性。或者，可以在调用图例时通过包含属性/值对直接设置属性。如果使用此调用形式，则必须将标签指定为字符串的单元格数组。图形对象属性详细记录在图形对象属性。

以下是支持的图例属性的子集:

autoupdate: "off" | {"on"}

控制在向对等轴添加(或从对等轴删除)对象时是否自动更新图例项的数量。例如:

## Create a single plot with its legend.

figure ();

plot (1:10);

legend ("Slope 1");

## Add another plot and specify its displayname so that

## the legend is correctly updated.

hold on;

plot ((1:10) \* 2, "displayname", "Slope 2");

## Stop automatic updates for further plots.

legend ("autoupdate", "off");

plot ((1:10) \* 3);

box: "off" | {"on"}

控制图例是否有周围框。

location: "best" | "bestoutside" |

"east" | "eastoutside" | "none" | "north" | {"northeast"} | "northeastoutside" | "northoutside" | "northwest"| "northwestoutside" | "south" | "southeast" | "southeastoutside" | "southoutside" | "southwest" | "southwestoutside" | "west" | "westoutside" 控制图例的位置。

numcolumns: scalar interger, def. 1

控制图例项布局中使用的列数。例如:

figure ();

plot (rand (30));

legend ("numcolumns", 3);

设置numcolumns还强制将numcolumnmode属性设置为“manual”。

orientation: "horizontal" | {"vertical"}

控制图例项是垂直排列(按列排列)还是水平排列(按行排列)。

string: string | cell array of strings

图例项的标签列表。例如:

figure ();

plot (rand (20));

## Let legend choose names automatically

hl = legend ();

## Selectively change some names

str = get (hl, "string");

str(1:5:end) = "Garbage";

set (hl, "string", str);

textcolor: colorspec, def. [0 0 0]

控制图例项的文本字符串的颜色。

支持的图例特定属性的完整列表可以在图例属性中找到。

图例是作为附加的轴对象实现的，标记属性设置为“legend”。可以使用set直接操作图例对象的属性。

可选的输出值hleg是图例对象的句柄。

实现注意:图例标签文本要么在对图例的调用中提供，要么从图形对象的DisplayName属性中获取。只有数据对象，如线、块和面，具有此属性，而轴、图形等不具有此属性，因此它们永远不会出现在图例中。如果没有标签或DisplayName属性可用，那么标签文本就是简单的"data1"， "data2"，…，"dataN"。

图例FontSize属性最初设置为它所附加的轴FontSize的90%。如果有必要，可以使用set来覆盖它。

: text (x, y, string)

: text (x, y, z, string)

: text (…, prop, val, …)

: text (hax, …)

: h = text (…)

创建一个文本对象，文本字符串位于当前轴上的x, y， (z)位置。

如果x, y， (z)是向量，则可以指定多个位置。可以使用字符矩阵或字符串单元数组指定多个字符串。

可选的属性/值对可用于控制文本的外观。

如果第一个参数hax是一个轴句柄，那么将文本添加到该轴，而不是gca返回的当前轴。

可选的返回值h是创建的文本对象的图形句柄向量。

示例1:通过3种不同的方法实现多行文本

text (0.5, 0.8, {"Line 1", "Line 2"})

text (0.5, 0.6, ["Line 1"; "Line 2"])

text (0.5, 0.4, "Line 1\nLine 2")

示例2:文本在多个位置

text ([0.2, 0.2], [0.8, 0.6], "Same text at two locations")

text ([0.4, 0.4], [0.8, 0.6], {"Point 1 Text", "Point 2 text"})

text ([0.6, 0.6], [0.8, 0.6], {{"Point 1 Line 1", "Point 1 Line 2},

"Point 2 text"})

示例2:使用文本属性调整外观

ht = text (0.5, 0.5, "Hello World", "fontsize", 20);

set (ht, "color", "red");

编程注意:完整的属性列表记录在文本属性中。

单元格数组中的任何数字项都将使用sprintf ("%g")转换为文本。为了更精确地控制外观，在调用text之前使用num2str、sprintf等将任何数字项转换为字符串。

参见:gtext, title, xlabel, ylabel, zlabel。

: xlabel (string)

: xlabel (string, property, val, …)

: xlabel (hax, …)

: h = xlabel (…)

指定用于标记当前轴的x轴的字符串。

可选的属性/值对列表可用于更改所创建文本标签的属性。

文本对象属性的完整列表记录在文本属性中。

如果第一个参数hax是一个轴句柄，那么操作这个轴，而不是gca返回的当前轴。

可选的返回值h是创建的文本对象的图形句柄。

参见:ylabel, zlabel，日期，标题，文本。

: ylabel (string)

: ylabel (string, property, val, …)

: ylabel (hax, …)

: h = ylabel (…)

指定用于标记当前轴的 y 轴的字符串。

如果指定了 hax，则标记 hax 定义的轴。

属性/值对的可选列表可用于更改所创建文本标签的属性。

文本对象属性的完整列表记录在文本属性中。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

可选返回值 h 是所创建文本对象的图形句柄。

参见: xlabel, zlabel, datetick, title, text.

: zlabel (string)

: zlabel (string, property, val, …)

: zlabel (hax, …)

: h = zlabel (…)

指定用于标记当前轴的 z 轴的字符串。

属性/值对的可选列表可用于更改所创建文本标签的属性。

文本对象属性的完整列表记录在文本属性中。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

可选返回值 h 是所创建文本对象的图形句柄。

参见: xlabel, ylabel, datetick, title, text.

: clabel (c, h)

: clabel (c, h, v)

: clabel (c, h, "manual")

: clabel (c)

: clabel (…, prop, val, …)

: hlabels = clabel (…)

向等值线图的等值线添加标签。

等值线级别由等值线矩阵 c 指定，该矩阵由 contour、contourc、contourf 和 contour3 返回。等值线标签将旋转以匹配本地线方向，并以线为中心。标签沿等值线的位置是随机选择的。

如果参数 h 是等值线组对象的句柄，则标记此图，而不是 gca 返回的当前轴中的图。

默认情况下，所有等值线都标注。但是，要标记的等值线可以由矢量 v 指定。如果给出了“手动”参数，则可以使用鼠标选择要标记的轮廓。

可以提供作为文本对象有效属性的其他属性/值对，并将其传递给基础文本对象。此外，等值线组属性“LabelSpacing”可用，该属性用于确定要指定的等值线上标注之间的间距。默认值为 144 磅，即 2 英寸。

可选返回值 hlabels 是表示每个标签的文本对象的图形句柄的矢量。文本对象的“userdata”属性包含等值线标签的数值。

文本对象属性的完整列表记录在文本属性中。

[c, h] = contour (peaks (), -4 : 6);

clabel (c, h, -4:2:6, "fontsize", 12);

参见: contour, contourf, contour3, meshc, surfc, text.

: box

: box on

: box off

: box (hax, …)

控制轴边框的显示。

参数可以是“on”或“off”。如果省略它，则切换当前框状态。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

参见: axis, grid.

: grid

: grid on

: grid off

: grid minor

: grid minor on

: grid minor off

: grid (hax, …)

控制打印网格线的显示。

函数状态输入可以是“on”或“off”。如果省略，则切换当前网格状态。

当第一个参数为“次要”时，所有后续命令都将修改次要网格而不是主要网格。

如果第一个参数 hax 是轴句柄，则在此轴上操作，而不是 gca 返回的当前轴。

要控制单个轴的网格线，请使用 set 函数。例如：

set (gca, "ygrid", "on");

参见: axis, box.

: colorbar

: colorbar (…, loc)

: colorbar (delete\_option)

: colorbar (hcb, …)

: colorbar (hax, …)

: colorbar (…, "peer", hax, …)

: colorbar (…, "location", loc, …)

: colorbar (…, prop, val, …)

: h = colorbar (…)

将颜色条添加到当前轴。

颜色条显示当前颜色图以及数字规则，以便可以解释颜色比例。

可选的输入 loc 确定颜色条的位置。如果存在，它必须是 colorbar 的最后一个参数。 loc 的有效值为

"EastOutside"

将颜色条放置在右侧绘图之外。这是默认设置。

"East"

将颜色条放置在右侧绘图内。

"WestOutside"

将颜色条放置在图的左侧。

"West"

将颜色条放置在绘图的左侧。

"NorthOutside"

将颜色条放置在绘图上方。

"North"

将颜色条放置在图的顶部。

"SouthOutside"

将颜色条放置在图下方。

"South"

将颜色条放在图的底部。

要从图中删除颜色条，请使用以下任一关键字作为删除选项：“off”、“delete”、“hide”。

如果第一个参数 hax 是坐标区句柄，则颜色条将添加到该坐标区，而不是 gca 返回的当前坐标区。或者，如果给出参数“peer”，则以下参数将被视为要在其中添加颜色条的轴句柄。 “peer”调用语法将来可能会被删除，因此不建议使用。

如果第一个参数 hcb 是 colorbar 对象的句柄，则直接对该 colorbar 进行操作。

其他属性/值对直接传递到基础坐标区对象。属性的完整列表记录在 Axes Properties 中。

可选的返回值 h 是所创建的颜色条对象的图形句柄。

实现注意事项：将颜色条创建为附加轴对象，并将“tag”属性设置为“colorbar”。创建的对象具有额外的属性“location”，它控制颜色条的位置。

参见: colormap.

: annotation (type)

: annotation ("line", x, y)

: annotation ("arrow", x, y)

: annotation ("doublearrow", x, y)

: annotation ("textarrow", x, y)

: annotation ("textbox", pos)

: annotation ("rectangle", pos)

: annotation ("ellipse", pos)

: annotation (…, prop, val)

: annotation (hf, …)

: h = annotation (…)

绘制注释以强调图形的某些部分。

您可以通过仅指定注释的类型来构建默认注释。

否则，您可以选择注释类型，然后使用基于线的注释的 x 和 y 坐标或其他位置矢量 pos 设置其位置。在任何一种情况下，坐标都是使用注释对象的“单位”属性来解释的。默认为“标准化”，这意味着图形的左下角坐标为“[0 0]”，右上角坐标为“[1 1]”。

如果第一个参数 hf 是图窗句柄，则绘制到该图窗中，而不是 gcf 返回的当前图窗。

可以以 prop/val 对的形式提供更多参数来自定义注释外观。

可选的返回值 h 是所创建的注释对象的图形句柄。这可以与 set 函数一起使用来自定义现有的注释对象。

所有注释对象共享两个属性：

* “units”：解释坐标的单位。

它的值可能是“厘米”之一 | “人物”| “英寸”| “{标准化}”| “像素”| “点”。

* “position”：一个四元素矢量[x0 y0 width height]。

该矢量指定注释对象的原点坐标 (x0,y0)、其宽度和高度。宽度和高度可能为负值，具体取决于对象的方向。

有效的注释类型及其具体属性如下所述：

"line"

构造一条线。 x 和 y 必须是二元素矢量，指定线两端的 x 和 y 坐标。

可以使用“linewidth”、“linestyle”和“color”属性来自定义线条，就像线条对象一样。

"arrow"

构造一个箭头。矢量 x 和 y 中的第二个点指定箭头坐标。

除了线条属性之外，还可以使用“headlength”、“headwidth”和“headstyle”属性来自定义箭头。 “headstyle”属性支持的值为：[“diamond”|“headstyle”] “椭圆”| “简单”| “矩形”| “vback1”| “{vback2}”| “vback3”]

"doublearrow"

构造一个双箭头。矢量 x 和 y 指定箭头坐标。

线条和箭头可以像箭头注释一样进行自定义，但有些属性名称是重复的：“head1length”/“head2length”、“head1width”/“head2width”等。索引 1 标记第一个箭头的属性x 和 y 坐标中的点。

textarrow"

构造一个箭头，在箭头的另一端带有文本标签。

使用“string”属性更改文本字符串。可以像箭头注释一样自定义线条和箭头，并且可以使用与文本图形对象相同的属性来自定义文本。但请注意，某些文本属性名称以“text”为前缀，以将其与箭头属性区分开：“textbackgroundcolor”、“textcolor”、“textedgecolor”、“textlinewidth”、“textmargin”、“textrotation”。

"textbox"

构建一个里面有文本的盒子。 pos 指定注释的“位置”属性。

使用“string”属性更改文本字符串。您可以使用“backgroundcolor”、“edgecolor”、“linestyle”和“linewidth”属性来自定义框背景颜色和边缘外观。还可以使用一组有限的文本对象属性；除了“font…”属性之外，您还可以使用“horizo​​ntalalignment”和“verticalalignment”来定位框中的文本。

最后，“fitboxtotext”属性控制框的实际范围。如果“打开”（默认），则框限制将适合文本范围。

"rectangle"

构造一个矩形。 pos 指定注释的 "position" 属性。

您可以使用 "facecolor"、"color"、"linestyle" 和 "linewidth" 属性来自定义矩形背景颜色和边缘外观。

"ellipse"

构造一个椭圆。 pos 指定注释的 "position" 属性。

请参阅 "rectangle" 注释进行自定义。

参见: xlabel, ylabel, zlabel, title, text, gtext, legend, colorbar.

**15.2.4一页多图**

八度音阶可以在单个图形中显示多个图形。最简单的方法是使用subplot函数将绘图区域划分为一系列以整数为索引的子绘图窗口。例如,

subplot (2, 1, 1)

fplot (@sin, [-10, 10]);

subplot (2, 1, 2)

fplot (@cos, [-10, 10]);

创建具有两个独立轴的图形，一个显示正弦波，另一个显示余弦波。对subplot的第一次调用将图划分为两个绘图区域(两行一列)，并使第一个绘图区域处于活动状态。由subplot创建的plot区域网格按行-主顺序编号(从左到右，从上到下)。绘制完正弦波后，对subplot的下一次调用会激活第二个subplot区域，但不会重新划分图形。

: subplot (rows, cols, index)

: subplot (rows, cols, index, hax)

: subplot (rcn)

: subplot (hax)

: subplot (…, "align")

: subplot (…, "replace")

: subplot ("position", pos)

: subplot (…, prop, val, …)

: hax = subplot (…)

建立一个以cols子窗口为单位的绘图网格，并将当前绘图轴(gca)设置为index给出的位置。

如果在(rows, cols, index)参数后提供轴句柄hax，则将相应的轴转换为子图。

如果只提供了一个数字参数，那么它必须是一个三位数的值，指定数字1中的行数、数字2中的列数和数字3中的绘图索引。

地块指数按行排列;首先，对一行中的所有列进行编号，然后填充下一行。

例如，具有2x3网格的绘图将具有如下运行的绘图索引:

+-----+-----+-----+

| 1 | 2 | 3 |

+-----+-----+-----+

| 4 | 5 | 6 |

+-----+-----+-----+

Index也可以是一个向量。在这种情况下，新的轴将包围指定的网格位置。第一个演示演示了这一点:

demo ("subplot", 1)

要激活的subplot的索引也可以由它的轴句柄hax指定，该轴句柄从上一个subplot命令返回。

如果提供了“align”选项，则子窗口的绘图框将对齐，但这可能没有给轴标记或标签留下空间。

如果给出了“replace”选项，则子图轴将被重置，而不仅仅是切换当前轴以绘制到所请求的子图。

“position”属性可用于精确定位当前图中的子图轴。选项pos是一个4元素向量[x, y, width, height]，它决定了轴的位置和大小。pos中的值在[0,1]范围内归一化。

任何属性/值对都直接传递给底层轴对象。完整的属性列表记录在Axes properties。

任何先前存在的将被新创建的轴(部分)覆盖的轴将被删除。

如果请求输出轴，subplot返回子图的轴句柄。这对于使用set修改子图的属性非常有用。

在某些情况下，subplot可能无法识别可以重用的轴，并可能替换它们。如果subplot轴需要重复引用，请考虑事先创建并存储它们的轴句柄，而不是为同一位置重复调用subplot。

例子:

x = 1:10;

y = rand (16, 10);

for i\_plot = 1:4

hax(i\_plot) = subplot (2, 2, i\_plot);

hold (hax(i\_plot), "on");

grid (hax(i\_plot), "on");

endfor

for i\_loop = 1:2

for i\_plot = 1:4

iy = (i\_loop - 1)\*4 + i\_plot;

plotyy (hax(i\_plot), x,y(iy,:), x,y(iy+1,:));

endfor

endfor

参见:axes, plot, gca, set。

**15.2.5多地块窗口**

您可以使用图形函数打开多个绘图窗口。例如,

figure (1);

fplot (@sin, [-10, 10]);

figure (2);

fplot (@cos, [-10, 10]);

创建两个图形，第一个显示正弦波，第二个显示余弦波。数字必须是正整数。

: figure

: figure n

: figure (n)

: figure (…, "property", value, …)

: h = figure (…)

创建一个用于绘图的新图形窗口。

如果没有指定参数，则创建一个包含下一个可用数字的新图形。

如果使用整数n调用，并且不存在该编号的图形，则创建具有指定编号的新图形。如果图形已经存在，则使其可见并成为用于绘图的当前图形。

可以为图形对象指定多个属性值对，但它们必须成对出现。

可选的返回值h是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。

参见:axes, gcf, shg, clf, close。

**15.2.6绘图对象的操作**

: pan

: pan on

: pan off

: pan xon

: pan yon

: pan (hfig, option)

控制图形界面中图形的交互平移方式。

给定选项“开”或“关”，设置交互式平移模式开或关。

在没有参数的情况下，打开或关闭当前平移模式。

给定选项“xon”或“yon”，仅对x轴或y轴启用平移模式。

如果第一个参数hfig是一个数字，则对给定的数字进行操作，而不是对gcf返回的当前数字进行操作。

参见:rotate3d，缩放。

: rotate (h, direction, alpha)

: rotate (…, origin)

将绘图对象h围绕具有方向和原点原点的直线旋转alpha度。

origin的默认值是h的父轴对象的中心。

如果h是句柄的向量，它们必须都有相同的父轴对象。

可以旋转的图形对象包括线、面、补丁和图像。

: rotate3d

: rotate3d on

: rotate3d off

: rotate3d (hfig, option)

在GUI中控制图形的交互式三维旋转模式。

给出“开”或“关”选项，设置交互旋转模式为开或关。

在没有参数的情况下，打开或关闭当前的旋转模式。

如果第一个参数hfig是一个数字，则对给定的数字进行操作，而不是对gcf返回的当前数字进行操作。

参见:平移，缩放。

: zoom

: zoom (factor)

: zoom on

: zoom off

: zoom xon

: zoom yon

: zoom out

: zoom reset

: zoom (hfig, option)

在GUI中缩放当前轴对象或控制图形的交互式缩放模式。

给定一个大于零的数值参数，按给定的因子进行缩放。如果缩放因子大于1，则放大绘图。如果因子小于1，则缩小。如果缩放因子是两个或三个元素的矢量，则这些元素分别指定x、y和z轴的缩放因子。

给出“开”或“关”选项，将交互式缩放模式设置为开或关。

在没有参数的情况下，打开或关闭当前缩放模式。

给定选项“xon”或“yon”，仅对x轴或y轴启用缩放模式。

给定“out”选项，缩放到初始缩放设置。

给定“重置”选项，存储当前的缩放设置，以便缩小将返回到该缩放级别。

如果第一个参数hfig是一个数字，则对给定的数字进行操作，而不是对gcf返回的当前数字进行操作。

参见:pan, rotate3d。

**15.2.7对绘图窗口的操作**

默认情况下，Octave在打印提示符或等待输入时刷新绘图窗口。drawnow函数用于更新绘图窗口。

: drawnow ()

: drawnow ("expose")

: drawnow (term, file, debug\_file)

更新图形窗口及其子窗口。

刷新事件队列，并执行生成的任何回调。

使用可选参数“expose”，只更新图形对象，不处理其他事件或回调。

drawnow的第三种调用形式是用于调试的，没有文档记录。

参见:refresh。

只有修改过的数字才会更新。刷新功能还可以用于更新当前图形，即使它没有被修改。

: refresh ()

: refresh (h)

刷新图形，强制重新绘制。

当不带参数调用时，将重新绘制当前图形。否则，重绘图形手柄为h的图形。

参见:drawnow。

通常，像plot或mesh这样的高级绘图函数调用newplot来确定目标轴的状态是否应该初始化(默认值)，或者是否应该在之前的绘图上绘制后续绘图。要使两个绘图相互重叠，请使用hold功能或手动更改坐标轴的nextplot属性。例如,

hold on;

x = -10:0.1:10;

plot (x, sin (x));

plot (x, cos (x));

hold off;

在同一轴上显示正弦波和余弦波。如果保持状态为关闭，像这样的连续绘图命令将只显示最后的绘图。

: newplot ()

: newplot (hfig)

: newplot (hax)

: hax = newplot (…)

准备图形引擎生成一个新的情节。

此函数在所有高级绘图函数的开头调用。在用户程序中通常不需要它。newplot查询当前图形和轴的“NextPlot”字段，以确定要做什么。

| **Figure NextPlot** | **Action** |
| --- | --- |
| *"new"* | Create a new figure and make it the current figure. |
| *"add"* (default) | Add new graphic objects to the current figure. |
| *"replacechildren"* | Delete child objects whose HandleVisibility is set to *"on"*. Set NextPlot property to *"add"*. This typically clears a figure, but leaves in place hidden objects such as menubars. This is equivalent to *clf*. |
| *"replace"* | Delete all child objects of the figure and reset all figure properties to their defaults. However, the following four properties are not reset: Position, Units, PaperPosition, PaperUnits. This is equivalent to *clf reset*. |

| **Axes NextPlot** | **Action** |
| --- | --- |
| *"add"* | Add new graphic objects to the current axes. This is equivalent to *hold on*. |
| *"replacechildren"* | Delete child objects whose HandleVisibility is set to *"on"*, but leave axes properties unmodified. This typically clears a plot, but preserves special settings such as log scaling for axes. This is equivalent to *cla*. |
| *"replace"* (default) | Delete all child objects of the axes and reset all axes properties to their defaults. However, the following properties are not reset: Position, Units. This is equivalent to *cla reset*. |

如果给出了可选输入hfig或hax，则准备指定的图形或轴，而不是当前的图形和轴。

可选的返回值hax是创建的轴对象(不是图形)的图形句柄。

注意:调用newplot可能会改变当前图形和当前轴。

: hold

: hold on

: hold off

: hold (hax, …)

切换或设置绘图引擎的“保持”状态，该状态决定是否将新的图形对象添加到绘图中或替换现有对象。

hold on

保留绘图数据和设置，以便在单个图形上显示后续绘图命令。线的颜色和线的样式是先进的每一个新的情节添加。

hold all (deprecated)

相当于hold on

hold off

恢复默认图形设置，在每个新的绘图命令之前清除图形并重置轴属性。(默认)。

hold

切换当前保持状态。

当给定额外的参数hax时，将修改该轴的保持状态，而不是gca返回的当前轴。

使用ishold函数查询当前保持状态。

参见:ishold, cla, clf, newplot。

: tf = ishold

: tf = ishold (hax)

: tf = ishold (hfig)

如果下一个绘图将被添加到当前绘图中，则返回true，如果在绘制下一个绘图之前将清除绘图设备，则返回false。

如果第一个参数是轴句柄hax或图形句柄hfig，那么操作这个图而不是当前的图。

参见:hold, newplot。

要清除当前图形，请调用clf函数。要清除当前轴，请调用类函数。要将当前图形移到窗口堆栈的顶部，请调用shg函数。要删除图形对象，请对其索引调用delete。要关闭图形窗口，请调用close函数。

: clf

: clf reset

: clf (hfig)

: clf (hfig, "reset")

: h = clf (…)

清除当前图形窗口。

clf通过删除具有可见句柄(HandleVisibility = "on")的子图形对象来操作。

如果指定了可选参数“reset”，则删除所有子对象，包括那些具有隐藏句柄的子对象，并将所有图形属性重置为默认值。但是，以下属性不会被重置:Position, Units, PaperPosition, PaperUnits。

如果第一个参数hfig是一个图形句柄，那么操作这个图形而不是gcf返回的当前图形。

可选的返回值h是被清除的图形窗口的图形句柄。

参见:class, close, delete, reset。

: cla

: cla reset

: cla (hax)

: cla (hax, "reset")

清除当前或指定的(hax)轴对象。

class通过删除具有可见句柄的子图形对象来操作(HandleVisibility = "on")。这通常会清除任何可视对象的轴，但保留轴限制、打勾标记和标签、相机视图等。此外，通过将轴属性ColorOrderIndex、LinestyleOrderIndex更改为1，可以重置线的自动着色和样式。

如果指定了可选参数“reset”，则删除所有子对象，包括那些具有隐藏句柄的子对象，并将所有轴属性重置为默认值。但是，以下属性不会被重置:位置，单位。

如果第一个参数hax是一个轴句柄，那么操作这个轴，而不是gca返回的当前轴。

参见:clf, delete, reset。

: shg

显示图形窗口。

此函数使当前图形可见，并将其置于所有其他绘图窗口的顶部。

编程注意:shg相当于figure (gcf)，假设存在一个当前figure。

参见:figure, drawnow, gcf。

: delete file

: delete file1 file2 …

: delete (file)

: delete (file1, file2, …)

: delete (handle)

删除指定的文件或图形句柄。

文件可能包含全球通模式，如“\*”。可以在同一个函数调用中指定要删除的多个文件。

句柄可以是要删除的图形句柄的标量或向量。

编程说明:删除图形对象是在不清除整个图形的情况下从绘图中删除特征的正确方法。

参见:clf, cla, unlink, rmdir。

: close

: close (h)

: close figname

: close all

: close all hidden

: close all force

: status = close (…)

关闭图形窗口。

在没有参数的情况下调用时，关闭当前图形。这相当于close (gcf)。如果输入h是图形句柄或图形句柄向量，则关闭h中的每个图形。也可以通过name figname指定要关闭的图形，该名称与所有图形的“name”属性相匹配。

如果给出参数"all"，则关闭所有具有可见句柄(HandleVisibility = "on")的图形。

如果给出附加参数“hidden”，则所有图形(包括隐藏的图形)都关闭。

如果给出了附加参数“force”，则即使更改了“closerequestfcn”以防止关闭窗口，也会关闭图形。

如果请求可选输出状态，那么如果图形窗口已成功关闭，则Octave返回1。

实现说明:close的操作是将句柄h设置为当前图形，然后调用该图形的“closerequestfcn”属性指定的函数。缺省情况下，使用closeeq函数。调用的函数可能会延迟或中止删除图形。要删除一个数字而不执行任何回调函数，请使用delete。当编写回调函数来关闭窗口时，不要使用close来避免递归。

参见:close, delete。

: closereq ()

关闭当前图形并删除与之关联的所有图形对象。

默认情况下，新绘图图的“closerequestfcn”属性指向此函数。

参见:关闭，删除。

**15.2.8使用“interpreter”属性**

文本(如标题，标签，图例项)和轴对象分别具有“interpreter”和“ticklabelinterpreter”属性。它决定文本中特殊控制序列的呈现方式。

解释器属性可以有三个值:"none"， "tex"， "latex"。

**15.2.8.1 "none"解释器**

如果将解释器设置为“none”，则不会发生特殊的呈现—显示的文本是指定文本的逐字副本。

**15.2.8.2 "tex"解释器**

“tex”解释器在呈现文本时实现了tex功能的子集。这允许插入特殊的字形，如希腊字符或数学符号。使用反斜杠(\)字符后跟代码插入特殊字符，如表15.1所示。

除了特殊的符号外，还可以通过使用代码在字符串中更改文本的格式

\bf Bold font

\it Italic font

\sl Oblique Font

\rm Normal font

这些代码可以与{和}字符一起使用，以限制对字符串的一部分的更改。例如,

xlabel ('{\bf H} = a {\bf V}')

其中字符“a”不会以粗体显示。注意，为了避免让Octave解释字符串中的反斜杠字符，字符串本身应该放在单引号中。

还可以在文本中更改字体名和大小

\fontname{fontname} Specify the font to use

\fontsize{size} Specify the size of the font to use

文本的颜色也可以使用字符串(例如，“红色”)或红色-绿色-蓝色(RGB)规范的数字(例如，[1 0 0]，也是红色)进行内联更改。

\color{color}将颜色指定为字符串

\color[rgb]{R G B}用数字指定颜色

最后，可以用'^'和'\_'字符控制上标和下标。如果'^'或'\_'后面跟一个{字符，那么被{}对包围的所有块都是上标或下标。如果没有{}对，则只更改紧接在'^'或'\_'后面的字符。

|  |
| --- |
| Greek Lowercase Letters |
|  | \alpha | \beta | \gamma |
|  | \delta | \epsilon | \zeta |
|  | \eta | \theta | \vartheta |
|  | \iota | \kappa | \lambda |
|  | \mu | \nu | \xi |
|  | \o | \pi | \varpi |
|  | \rho | \sigma | \varsigma |
|  | \tau | \upsilon | \phi |
|  | \chi | \psi | \omega |
| Greek Uppercase Letters |  |  |  |
|  | \Gamma | \Delta | \Theta |
|  | \Lambda | \Xi | \Pi |
|  | \Sigma | \Upsilon | \Phi |
|  | \Psi | \Omega |  |
| Misc Symbols Type Ord |  |  |  |
|  | \aleph | \wp | \Re |
|  | \Im | \partial | \infty |
|  | \prime | \nabla | \surd |
|  | \angle | \forall | \exists |
|  | \neg | \clubsuit | \diamondsuit |
|  | \heartsuit | \spadesuit |  |
| “Large” Operators |  |  |  |
|  | \int |  |  |
| Binary Operators |  |  |  |
|  | \pm | \cdot | \times |
|  | \ast | \circ | \bullet |
|  | \div | \cap | \cup |
|  | \vee | \wedge | \oplus |
|  | \otimes | \oslash |  |
| Relations |  |  |  |
|  | \leq | \subset | \subseteq |
|  | \in | \geq | \supset |
|  | \supseteq | \ni | \mid |
|  | \equiv | \sim | \approx |
|  | \cong | \propto | \perp |
| Arrows |  |  |  |
|  | \leftarrow | \Leftarrow | \rightarrow |
|  | \Rightarrow | \leftrightarrow | \uparrow |
|  | \downarrow |  |  |
| Openings and Closings |  |  |  |
|  | \lfloor | \langle | \lceil |
|  | \rfloor | \rangle | \rceil |
| Alternate Names |  |  |  |
|  | \neq |  |  |
| Other |  |  |  |
|  | \ldots | \0 | \copyright |
|  | \deg |  |  |

表15.1:TeX模式中可用的特殊字符

警告:学位符号

关于\circ符号，TeX和MATLAB的一致性是不可能的。当TeX将此符号转换为Unicode 2218 (U+2218)时，MATLAB将其映射为Unicode 00B0 (U+00B0)。Octave选择遵循TeX规范，但增加了映射到度数符号(U+00B0)的额外符号\deg。

**15.2.8.3 "latex"解释器**

“latex”解释器只有在存在外部latex工具链时才能工作。需要三个二进制文件:latex、dvipng和dvisvgm。如果这些二进制文件已经安装，但不在路径上，仍然可以使用以下环境变量提供它们各自的路径:OCTAVE\_LATEX\_BINARY、OCTAVE\_DVIPNG\_BINARY和OCTAVE\_DVISVG\_BINARY。

注意，Octave在“latex”模式下不会解析或验证文本字符串——生成有效字符串是程序员的责任，其中可能包括应该在Math模式下出现的带有“$”字符的包装部分。例如，请参阅https://www.latex-project.org/help/documentation/获取有关LaTeX排版的文档。

出于调试目的，可以设置一个方便的环境变量OCTAVE\_LATEX\_DEBUG\_FLAG，以便在Octave无法让外部LaTeX引擎编译给定文本时触发更详细的输出。例如，"x^2"不是一个有效的LaTeX字符串，下面的示例应该失败

setenv ("OCTAVE\_LATEX\_DEBUG\_FLAG", "1")

x = 1:10;

plot (x, x.^2)

title ("x^2", "interpreter", "latex")

搜索终端输出，你会发现一些有用的信息，关于失败的来源:

...

No file default.aux.

! Missing $ inserted.

<inserted text>

$

l.6 x^

2

! Missing $ inserted.

...

如果在第一次文本渲染时没有找到可用的LaTeX工具链，则使用“LaTeX”解释器相当于“无”。

**15.2.9打印和保存绘图**

print命令允许您将绘图发送到打印机，并以各种格式保存绘图。例如,

print -dpsc

将当前图形打印到彩色PostScript打印机。而且,

print -deps foo.eps

将当前图形保存到名为foo.eps的封装PostScript文件中。

当前的图形工具包产生非常相似的图形显示，但在显示不寻常文本的能力和打印此类文本的能力方面有所不同。一般来说，“tex”解释器(默认)对于屏幕显示和打印来说都是性能最好的。然而，对于复杂文本公式的再现，首选“latex”解释器。当使用-painters渲染器(所有矢量格式的默认值)打印时，可以考虑两个选项:

使用-svgconvert选项允许呈现LaTeX公式。注意，字形呈现为路径，原始文本信息丢失。

使用-d\*latex\*设备之一生成.tex文件(加上支持的.eps或.pdf文件)，由外部latex引擎进一步处理。注意，print函数首先将所有字符串的解释器设置为"latex"，这意味着所有字符串必须是有效的latex字符串。

使用-dpdflatexstandalone选项显示文本打印功能的完整示例如下:

x = 0:0.01:3;

hf = figure ();

plot (x, erf (x));

hold on;

plot (x, x, "r");

axis ([0, 3, 0, 1]);

text (0.65, 0.6175, ...

['$\displaystyle\leftarrow x = {2 \over \sqrt{\pi}}' ...

'\int\_{0}^{x} e^{-t^2} dt = 0.6175$'],

"interpreter", "latex");

xlabel ("x");

ylabel ("erf (x)");

title ("erf (x) with text annotation");

print (hf, "plot15\_7", "-dpdflatexstandalone");

system ("pdflatex plot15\_7");

open plot15\_7.pdf

此示例的结果如图15.7所示

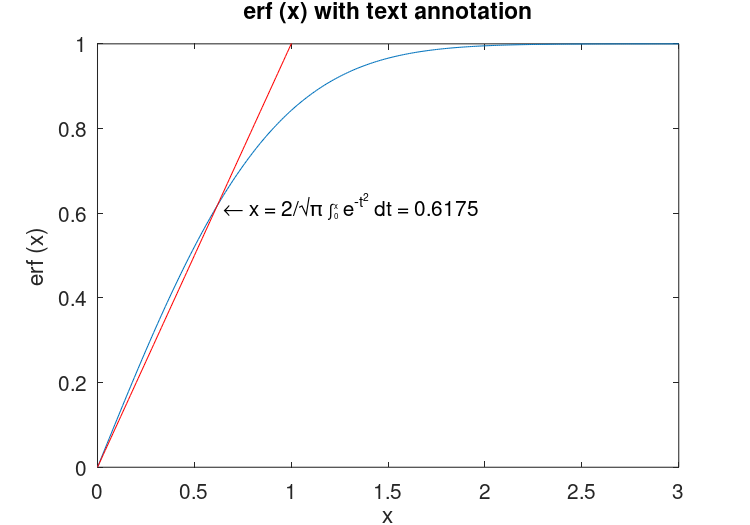


图15.7:使用-dpdflatexstandalone包含文本的示例

: print ()

: print (options)

: print (filename, options)

: print (hfig, …)

: RGB = print ("-RGBImage", …)

对要打印的图形进行格式化，然后将其保存到文件、发送到打印机或返回RGB图像。

Filename定义输出文件的名称。如果文件名没有后缀，则从指定的设备推断出一个并附加到文件名。当文件名和“-RGBImage”选项都不存在时，输出将被发送到打印机。各种选项和文件名参数可以以任何顺序给出，除了图形句柄参数hfig，如果它出现，则必须在第一个。

示例:使用PDF和JPEG格式打印到文件。

figure (1);

clf ();

surf (peaks);

print figure1.pdf # The extension specifies the format

print -djpg figure1 # Will produce "figure1.jpg" file

如果第一个参数是图形对象的句柄hfig，则它指定要打印的图形。默认情况下，打印gcf返回的当前数字。

对于分页格式(例如PostScript和PDF)的输出，页面大小由图形的papersize属性和paperunits属性指定。图形在页面上的位置和大小由图形的纸张位置属性指定。页面的方向由图形的纸张方向属性指定。

对于非页面格式—例如，像jpeg这样的图像格式—输出的宽度和高度由图形的paperposition(3:4)属性值指定。

print命令支持许多选项:

-fh

指定要打印的图形的手柄h。

示例:打印图1。

figure (1);

clf ();

surf (peaks);

figure (2);

print -f1 figure1.pdf

## Equivalent functional form:

print (1, "figure1.pdf")

-Pprinter

如果没有指定文件名，则设置发送绘图的打印机名称。

示例:使用PostScript格式打印到名为PS\_printer的打印机。

clf ();

surf (peaks);

print -dpswrite -PPS\_printer

-RGBImage

返回该图形的m × n × 3的RGB图像。图像的大小取决于格式化选项。这类似于获取情节的屏幕截图，但格式选项可能会更改，例如分辨率或单色/颜色。

示例:获取图形图像的像素。

clf ();

surf (peaks);

rgb = print ("-RGBImage");

-image | -opengl

-vector | -painters

指定是使用基于像素的渲染器(-image或-opengl)还是基于矢量的渲染器(-vector或-painters)。这相当于改变图形的“Renderer”属性。当图形“RendererMode”属性为“auto”(默认值)时，Octave将对栅格格式(如JPEG)使用“opengl”渲染器，对矢量格式(如PDF)使用“painter”渲染器。这些选项仅支持“qt”图形工具包。

-svgconvert (default)

-nosvgconvert

当使用-painters渲染器时，这将启用或禁用基于SVG的后端工具链，并具有增强的特性:

Font handling:

对于解释器"none"和"tex"，实际的字体被嵌入到输出文件中，允许以所有矢量格式打印任意字符和字体。

使用“latex”解释器的字符串使用路径对象呈现。这看起来不错，但请注意文本信息(字体，字符…)丢失了。

Output Simplification:

默认情况下，选项-painter使用三角形组件渲染补丁和表面对象。这可能导致在查看文件时出现反混叠现象。svgconvert选项重建多边形，以避免这些工件(特别是2-D图形)。

Transparency:

允许打印PDF格式的透明图形对象。对于PostScript格式，任何透明对象的存在都会导致输出光栅化。

警告:如果Octave是针对早于5.13的Qt版本构建的，-svgconvert可能会导致图像对象的不准确渲染。

-polymerge

-nopolymerge

-polymerge-all

当使用基于SVG的后端-svgconvert时，面被渲染为三角形。在某些情况下，一些观察者可能会在这些三角形共用一条边的地方显示细线。这些选项控制是否将所有共享边的三角形合并为多边形(-polymerge-all对于由许多三角形组成的图形(包括线标记)可能需要一些时间)，仅合并连续多边形(-polymerge)，或者根本不合并三角形(-no-polymerge)。默认情况下，只合并共享一条边的连续三角形，除非打印图形包含补丁或表面图形对象，在这种情况下，合并共享一条边的所有三角形。

-portrait

-landscape

指定打印输出图的方向。对于非打印输出，输出的宽高比对应于在指定方向上由“paperposition”属性定义的绘图面积。此选项相当于更改图形的“纸张方向”属性。

-fillpage

-bestfit

当使用基于页面的格式(PDF、PostScript、打印机)时，忽略“paperposition”属性，让绘图占据整个页面。option -fillpage将拉伸该情节，使其以0.25英寸的空白占据整个页面。选项-bestfit将扩展情节以占用尽可能多的页面空间，而不会扭曲情节的原始宽高比。

-color

-mono

彩色或单色输出。

-solid

-dashed

分别强制所有线为实线或虚线。

-noui

不要打印uicontrol对象，例如按钮，因为它们可能会覆盖绘图。这是默认行为，如果不使用外部屏幕捕获工具，就不可能在输出中包含uicontrol对象。

-rNUM

位图的分辨率(以每英寸点数为单位)。对于元文件和SVG，默认值都是屏幕分辨率;对于其他格式，默认值是150dpi。要指定屏幕分辨率，使用"- 0"。

例如:高分辨率光栅输出。

clf ();

surf (peaks (), "facelighting", "gouraud");

light ();

print ("-r600", "lit\_peaks.png");

-Sxsize,ysize

光栅格式(包括PNG、JPEG、PNG和SVG)的图大小(以像素为单位)。对于所有矢量格式，包括PDF, PS和EPS，绘图大小以点为单位指定。此选项相当于通过设置图形属性paperposition(3:4)来改变输出的宽度和高度。在使用print函数的命令形式时，必须引用xsize,ysize选项，以防止Octave解释器识别嵌入的逗号('，')。例如，写入“-S640,480”。

-tight

-loose

为EPS文件强制一个紧或松的边界框。默认值是紧的。

-preview

添加预览EPS文件。支持的格式有:

-interchange

提供交换预览。

-metafile

提供元文件预览。

-pict

提供图片预览。

-tiff

提供TIFF预览。

-append

将PostScript或PDF输出给同一类型的现有文件。

-Ffontname

-Ffontname:size

-F:size

对所有文本使用fontname和/或fontsize。对于某些设备，字体名称被忽略:fig等。

-ddevice

可用的输出格式由选项设备指定，并且是以下格式之一(带有'\*'标记的设备仅在Gnuplot工具包中可用):

矢量格式

svg

可伸缩矢量图形

pdf

pdfcrop

便携式文档格式。pdf设备格式化图形以便在纸上打印。周围页面的大小和图形在页面内的位置由纸张图形属性定义。

如果不想要周围的页面，可以使用pdf。

注意:使用-nosvgconvert选项，PDF继承了与PostScript相同的限制(有限的字体集和缺乏透明度)。

eps(2)

epsc(2)

封装PostScript(1级和2级，单声道和彩色)。

基于opengl的图形工具包总是生成PostScript 3.0。它们对文本的支持有限，除非使用-svgconvert选项(默认值)。限制包括只使用ASCII字符(例如，不使用希腊字母)和只支持三种基本PostScript字体:Helvetica(默认)、Times或Courier。任何其他字体将被Helvetica取代。

ps(2)

psc(2)

与eps相同，不同之处在于该图形是为在纸上打印而格式化的。周围页面的大小和图形在页面内的位置由纸张图形属性定义。

pslatex

epslatex

pdflatex

pslatexstandalone

epslatexstandalone

pdflatexstandalone

为绘图的文本部分生成一个LaTeX文件filename.tex，为其余图形生成一个文件filename.(ps|eps|pdf)。图形文件后缀“。ps|eps|pdf”由指定的设备类型决定。由“standalone”选项生成的LaTeX文件可以由LaTeX直接处理。不使用“standalone”选项生成的文件将包含在另一个LaTeX文档中。在这两种情况下，LaTeX文件都包含一个\includegraphics命令，以便在处理LaTeX文件时自动包含生成的图形文件。写入LaTeX文件的文本包含的字符串与图中指定的字符串完全相同。如果使用了TeX模式解释器的任何特殊字符，则必须在LaTeX处理之前编辑文件。具体来说，特殊字符必须用美元符号($…$)括起来，并且LaTeX可以识别的其他字符也可能需要编辑(例如，大括号)。“pdflatex”设备和任何“独立”格式都不能在Gnuplot工具包中使用。

epscairo\*

pdfcairo\*

epscairolatex\*

pdfcairolatex\*

epscairolatexstandalone\*

pdfcairolatexstandalone\*

使用Cairo渲染器生成输出。epscairo和pdfcairo设备是epsc设备的同义词。LaTeX变体为绘图的文本部分生成一个LaTeX文件filename.tex，为绘图的图形部分生成一个图像文件filename.(eps|pdf)。' standalone '变体的行为与上面描述的' epslatexstandalone '一样。

canvas\*

在HTML5画布上基于javascript的绘图，可在web浏览器中查看。

emf

meta

Microsoft Enhanced Metafile

fig

XFig。对于Gnuplot图形工具包，可以使用附加选项-textspecial或-textnormal(默认值)来控制是否应该为图中的文本设置特殊标志。

latex\*

eepic\*

乳胶图片环境和扩展图片环境。

tikz

tikzstandalone\*

使用PGF/TikZ格式生成LaTeX文件。基于opengl的工具包创建PGF文件，而Gnuplot创建TikZ文件。“tikzstandalone”设备生成一个LaTeX文档，其中包含TikZ文件。

光栅文件

png

便携式网络图像格式

jpg

jpeg

JPEG图像：一种常见的图像文件格式，全称为Joint Photographic Experts Group，通常用于存储和传输数字照片。

tif

tiff

tiffn

具有LZW压缩(tif, TIFF)或未压缩(tiffn)的TIFF图像。

gif

GIF图像

pbm

PBMplus

dumb\*

ASCII字符画

如果省略设备，则从文件扩展名中推断，或者如果没有文件名，则将其作为PostScript发送到打印机。

-dghostscript\_device

Ghostscript支持其他设备。一些例子是:

ljet2p

HP LaserJet IIP

pcx24b

24位彩色PCX文件格式

ppm

便携式像素地图文件格式

要获得可用格式和设备的完整列表，请键入system(“gs -h”)。

当Ghostscript输出发送到打印机时，大小由图形的“papersize”属性决定。当输出发送到文件时，大小由图形的“paperposition”属性定义的绘图框决定。

-Gghostscript\_command

指定调用Ghostscript的命令。Unix的默认值是“gs”，Windows的默认值是“gswin32c”。

-TextAlphaBits=n

-GraphicsAlphaBits=n

Octave能够通过使用Ghostscript生成各种打印机，位图和矢量格式的输出。对于位图和打印机输出，使用Ghostscript的TextAlphaBits和GraphicsAlphaBits选项应用抗锯齿。默认位数分别为4和1。N的取值为1、2或4。

-no-append-file-extension

使用此选项时，将逐字使用文档名。这意味着不会自动附加与文档格式匹配的文档扩展名。

参见:saveas, getframe, savefig, hgsave, orient, figure。

: saveas (h, filename)

: saveas (h, filename, fmt)

将图形对象h保存为图形格式fmt的文件文件名。

如果h是图形对象的句柄，则保存该图形对象。如果h是另一个图形对象的句柄，则保存包含该图形对象的图形。

可以使用打印接受的所有设备格式。常见的格式有:

ofig

八度图形文件格式(默认)

mfig

两个文件:Octave m-file filename。m包含代码，以打开八度音程图形文件filename.ofig

ps

PostScript

eps

被封装的 PostScript 格式（PostScript 是一种编程语言）

带有预视图象的 PostScript 格式（PostScript 是一种编程语言）

pdf

便携式文档格式：一种由Adobe公司开发的电子文档格式，可以在不同操作系统和设备上保持格式一致性。

jpg

JPEG图像：一种常见的图像文件格式，全称为Joint Photographic Experts Group，通常用于存储和传输数字照片。

png

便携式网络图形图像

emf

增强型图元文件

tif

TIFF图像，压缩

如果省略fmt是提取文件名的扩展。当没有扩展名时，默认格式为“ofig”。

clf ();

surf (peaks);

saveas (1, "figure1.png");

参见:print, savefig, hgsave, orient。

: orient (orientation)

: orient (hfig, orientation)

: orientation = orient ()

: orientation = orient (hfig)

查询或设置图形hfig的打印方向。

方向的有效值为"portrait"、"landscape"和"tall"。

“景观”选项改变方向，使地块宽度大于地块高度。“纸张位置”也进行了修改，使情节填满页面，同时留下0.25英寸的边框。

“tall”选项将方向设置为“纵向”，并在页面中填充情节，同时留下0.25英寸的边框。

“纵向”选项(默认)改变方向，使地块高度大于地块宽度。它还恢复默认的“paperposition”属性。

当不带参数调用时，返回当前的打印方向。

如果省略参数hfig，则对gcf返回的当前图形进行操作。

参见:打印，保存。

打印和保存在一个情节的工作已经完成并且输出必须是出版准备格式时使用。在中间阶段，最好保存图形对象及其所有相关信息，以便在Octave中轻松地更改颜色、轴限制、标记样式等。hgsave/hgload命令可用于保存和重新创建图形对象。

: hgsave (filename)

: hgsave (h, filename)

: hgsave (h, filename, fmt)

将图形句柄h以fmt格式保存到文件文件名中。

如果未指定，h是gcf返回的当前数字。

当filename没有扩展名时，默认的文件扩展名。ofig将被追加。

如果存在，fmt必须是下列之一:

* -binary, -float-binary
* -hdf5, -float-hdf5
* -V7, -v7, -7, -mat7-binary
* -V6, -v6, -6, -mat6-binary
* -text
* -zip, -z

默认格式为-binary以最小化存储空间。

编程注意事项:为最终出版物制作图形时，请使用打印或保存。当重要的是能够继续编辑一个图形作为一个Octave对象时，使用hgsave/hgload。

参见:hgload, hdl2struct, savefig, saveas, print。

: h = hgload (filename)

: [h, old\_prop] = hgload (filename, prop\_struct)

将文件名中的图形对象加载到图形句柄向量中。

如果filename没有扩展名，Octave将尝试查找带有或不带有默认扩展名.ofig的文件。

如果提供，结构prop\_struct的元素将用于覆盖存储在filename中的顶级对象的属性，并且filename中保存的值将存储在old\_prop中。Old\_prop是一个匹配h大小的单元格数组;在被覆盖之前，每个单元格包含现有属性名称和值的结构。

参见:openfig, hgsave, struct2hdl。

: openfig

: openfig (filename)

: openfig (…, copies)

: openfig (…, visibility)

: h = openfig (…)

从文件名中读取保存的图形窗口并返回图形句柄h。

默认情况下，文件名为“Untitled.fig”。如果未指定完整路径，则打开的文件将是加载路径中遇到的第一个文件。如果找不到文件名并且没有扩展名，则将搜索加载路径中扩展名为“”的第一个文件。图“或”。Ofig”，按这个顺序。

拷贝是一个可选的输入，指示是否应该创建一个新的图形(“新的”)或者是否可以重用一个现有的图形(“重用”)。如果"FileName"属性与指定的输入文件名匹配，则可以重用现有的图形。当一个图被重用时，它将成为活动图，并显示在其他图的顶部。如果图形在屏幕外，它将被重新定位到屏幕上。副本的默认值为“new”。

可见性是一个可选输入，指示是否显示图形(“可见”)或不显示(“不可见”)。当可见度被指定为openfig的输入时，它会覆盖存储在filename中的可见度设置。

参见:open, hgload, savefig, struct2hdl。

: savefig ()

: savefig (h)

: savefig (filename)

: savefig (h, filename)

: savefig (h, filename, "compact")

保存图形句柄h指定的图形窗口到文件filename。

如果未指定，h是gcf返回的当前数字。

如果未指定，则文件名设置为“Untitled.fig”。如果filename没有扩展名，则默认扩展名为"。将添加Fig”。

如果存在可选的第三个输入“compact”，则数据将被压缩以节省更多空间。

参见:hgsave, hdl2struct, openfig。

**15.2.10与情节交互**

用户可以使用ginput函数选择绘图上的点，或者使用鼠标使用gtext函数选择在绘图上放置文本的位置。

: [x, y, buttons] = ginput (n)

: [x, y, buttons] = ginput ()

返回当前图形窗口中鼠标单击和/或击键的位置和类型。

如果定义了n，那么在返回之前捕获n个事件。当n未定义时，ginput将循环，直到按下返回键RET。

返回值x、y是以当前轴为单位单击鼠标的坐标。对于左、中或右按钮，返回值按钮为1、2或3。如果按下某个键，则在button中返回ASCII值。

实现说明:ginput用于二维绘图。对于三维绘图，请参阅当前轴的当前点属性，该属性可以通过当前视图的知识转换为数据单元。

参见:gtext, waitforbuttonpress。

: b = waitforbuttonpress ()

等待在当前图形窗口上单击鼠标或按下键。

如果按下鼠标按钮，b的返回值为0，如果按下键，返回值为1。

参见:waitfor, ginput, kbhit。

: gtext (s)

: gtext ({s1, s2, …})

: gtext ({s1; s2; …})

: gtext (…, prop, val, …)

: h = gtext (…)

使用鼠标将文本放置在当前图形上。

字符串参数s可以是字符数组或字符串的单元格数组。如果s有多行，则在单击鼠标后使用每行创建一个单独的文本对象。例如:

在一次鼠标点击后放置一个字符串

gtext ("I clicked here")

鼠标点击两次后放置两个字符串

gtext ({"I clicked here"; "and there"})

鼠标点击两次后，放置两个字符串，每个字符串有两条线

gtext ({"I clicked", "here"; "and", "there"})

可选的属性/值对直接传递给底层文本对象。

文本对象属性的完整列表记录在文本属性中。

可选的返回值h保存了创建的文本对象的图形句柄。

参见:ginput, text。

更复杂的用户交互机制可以使用ui\*函数族获得，参见ui元素。

**15.2.11测试绘图函数**

函数sombrero和peaks提供了一种检查绘图工作的方法。在“Octave”提示符下输入“sombrero”或“peaks”，将显示一个三维图形。

: sombrero ()

: sombrero (n)

: z = sombrero (…)

: [x, y, z] = sombrero (…)

绘制熟悉的三维宽边帽函数。

绘制的函数是

z = sin (sqrt (x^2 + y^2)) / (sqrt (x^2 + y^2))

调用时不带返回参数，sombrero使用surf在网格[-8,8]上绘制上述函数的表面。

如果n是一个标量，则该图由n条网格线组成。n的默认值是41。

当使用输出参数调用时，返回在网格上求值的函数的数据。这随后可以用surf (x, y, z)绘制。

参见:峰值，网格，网格，冲浪。

: peaks ()

: peaks (n)

: peaks (x, y)

: z = peaks (…)

: [x, y, z] = peaks (…)

画一个有很多局部最大值和最小值的函数。

函数有这样的形式

f(x,y) = 3\*(1-x)^2\*exp(-x^2 - (y+1)^2) ...

- 10\*(x/5 - x^3 - y^5)\*exp(-x^2-y^2) ...

- 1/3\*exp(-(x+1)^2 - y^2)

调用时不带返回参数，peaks使用surf绘制上述函数的表面。

如果n是标量，则在n × n网格上绘制上述函数在[-3,3]范围内的值。n的默认值为49。

如果n是一个向量，那么它表示计算函数的网格值。如果指定了x和y，则在指定的顶点网格上计算函数值。

当使用输出参数调用时，返回在网格上求值的函数的数据。这随后可以用surf (x, y, z)绘制。

参见:宽边帽，网格，网格，冲浪。

**15.3图形数据结构**

**15.3.1图形结构简介**

图形函数使用类graphics\_handle的指针来定位控制可视化显示的数据结构。图形句柄可以指向许多不同的基本对象类型中的任何一种。这些对象就是图形数据结构本身。原始图形对象类型有:图形、轴、线、文本、补丁、散射、表面、文本、图像和光。

这些对象中的每一个都有一个同名的函数，这些函数中的每一个都返回一个指向相应类型对象的图形句柄。

此外，还有几个函数对图形对象的属性进行操作，并返回句柄。这包括但不限于以下函数:函数plot和plot3返回一个指向line类型对象的句柄。函数subplot返回一个指向类型为axes的对象的句柄。函数fill、fill3、trimesh和trisurf返回一个指向patch类型对象的句柄。函数scatter3返回一个散点类型对象的句柄。函数slice, surf, surfl, mesh, meshz, pcolor和waterfall都返回一个类型为surface的句柄。函数camlight返回一个light类型对象的句柄。函数area、bar、bar、contour、contourf、contour3、surf、meshc、errorbar、quiver、quiver3、stair、stem、stem3都返回一个复杂数据结构的句柄，如数据源中所述。

图形对象按层次结构排列:

1. 根对象由root返回(历史上，相当于句柄0)。换句话说，get (root)返回根对象的属性。

2. 根的下面是图形对象。

3. 在图形对象下面是轴或g组对象。

4. 在轴或g组对象下面是线、文本、patch、scatter、surface、image和light对象。

可以通过查询图形对象的“父”和“子”属性来遍历这个层次树。

图形句柄可以通过ishhandle函数与函数句柄(参见函数句柄)区别开来。如果ishhandle的参数是一个图形对象的句柄，则返回true。此外，图形或轴对象可以分别使用isfigure或isaxes进行测试。要测试特定类型的图形句柄，例如补丁或直线对象，请使用isgraphics。更具体的测试函数只有在参数既为图形句柄又为正确类型(图形、轴、指定对象类型)时才返回true。

get和set命令用于获取和设置图形对象的属性值。此外，还可以使用get命令获取属性名称。

例如，图形句柄h所指向的图形对象的属性“type”可以显示为:

get (h, "type")

属性及其当前值可以使用s = get (h)以结构形式获得，其中h是图形对象的句柄。如果只需要属性的名称和允许的值(仅针对无线电属性)，则可以使用set (h)。

因此，例如:

h = figure ();

get (h, "type")

⇒ ans = figure

set (h)

⇒

alphamap:

beingdeleted: [ {off} | on ]

busyaction: [ cancel | {queue} ]

buttondownfcn:

clipping: [ off | {on} ]

closerequestfcn:

color:

colormap:

createfcn:

currentaxes:

deletefcn:

dockcontrols: [ {off} | on ]

filename:

graphicssmoothing: [ off | {on} ]

handlevisibility: [ callback | off | {on} ]

...

得到和设置的使用进一步解释了。

: res = isprop (obj, "prop")

如果prop是对象obj的属性，则返回true。

Obj也可以是一个对象数组，在这种情况下res将是一个逻辑数组，指示每个句柄是否具有属性prop。

对于绘图，obj是图形对象的句柄。否则，obj应该是一个类的实例。isprop报告类是否定义了属性，但是访问权限或可见性限制(Hidden = true)可能会阻止程序员使用。

参见:get, set, properties, ismethod, isobject。

**15.3.2图形对象**

上面已经解释了图形对象的层次结构。参见图形结构介绍。这里描述了具体的对象，并讨论了这些对象中包含的属性。请记住，图形对象总是由句柄引用。

root

层次结构的顶层和所有图形对象的父级。使用root获取根图形对象的句柄。

figure

图形窗口。

axes

一组轴。该对象是图形对象的子对象，并且可以是线、文本、图像、补丁、表面或光对象的父对象。

line

二维或三维的直线

text

文本注释。

image

位图图像。

patch

一种填充多边形，目前仅限于二维。

surface

三维曲面

light

在小块和表面上用于照明效果的光物体。

**15.3.2.1创建图形对象**

你可以通过调用与对象同名的函数来创建任何图形对象原语;换句话说，图形、轴、线、文本、图像、补丁、表面和光函数。这些基本图形对象自动成为当前轴对象的子对象，就像保持在适当位置一样。另外，轴将自动成为当前图形对象的子对象，图形将成为根对象的子对象。

如果不需要这种自动连接功能，那么首先调用newplot来准备一个新的图形和坐标轴进行绘图是很重要的。或者，更简单的方法是调用高级图形例程，它将创建绘图，然后用低级图形对象填充它。不要用电话，用情节。或者用surf代替surface。或者使用fill或fill3代替patch。

: axes ()

: axes (property, value, …)

: axes (hpar, property, value, …)

: axes (hax)

: h = axes (…)

创建一个笛卡尔轴对象并返回一个句柄，或者将当前轴设置为hax。

调用时不带任何参数，或带属性/值对，将构造一个新轴。可选参数hpar是一个图形句柄，指定新轴的父节点，可以是图形、uipanel或uitab。

使用单个轴句柄参数hax调用，该函数使hax为当前轴(由gca返回)。它还使包含hax的图形为当前图形(由gcf返回)。最后，它重新堆叠父对象的children属性，以便该轴出现在列表中所有其他轴句柄之前。这将导致hax显示在任何其他轴对象的顶部(z顺序堆叠)。此外，它重新堆叠与hax关联的任何图例或颜色条对象，以便它们也可见。

编程注意:属性的完整列表记录在Axes properties。

参见:gca, set, get。

: line ()

: line (x, y)

: line (x, y, z)

: line ("xdata", x, "ydata", y)

: line ("xdata", x, "ydata", y, "zdata", z)

: line (…, property, value)

: line (hax, …)

: h = line (…)

从x和y(可能还有z)创建一个直线对象，并将其插入当前轴中。

在标准调用形式中，数据x、y和z可以是标量、向量或矩阵。在矩阵输入的情况下，line将尝试定向标量和向量，以便可以绘制结果。这要求向量的一个维度匹配矩阵的行数或列数。

在低级调用形式(性能提高50%)中，数据由名称(行("xdata"， x，…))指定，数据必须是向量。如果没有指定数据(line())，则x == y =[0,1]。

可以为line对象指定多个属性值对，但它们必须成对出现。

如果只使用属性/值对调用，则任何未指定的属性使用根对象上指定的默认值。

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。

可选的返回值h是创建的线条对象的图形句柄(或句柄向量)。

编程注意:完整的属性列表记录在Line properties中。

函数线与绘图的不同之处在于，在不首先清除绘图的情况下，将直线对象插入到当前轴中。

参见:图像，补丁，矩形，表面，文本。

: patch ()

: patch (x, y, c)

: patch (x, y, z, c)

: patch ("Faces", faces, "Vertices", verts, …)

: patch (…, "prop", val, …)

: patch (…, propstruct, …)

: patch (hax, …)

: h = patch (…)

在当前坐标轴上创建patch对象，顶点位置为(x, y)，颜色为c。

如果顶点是大小为MxN的矩阵，则每个多边形补丁有M个顶点，总共将创建N个多边形。如果某些多边形没有M个顶点，则使用NaN表示“无顶点”。如果z输入存在，则会创建3-D补丁。

颜色论证c可以有多种形式。要创建所有共享单一颜色的多边形，请使用字符串值(例如，“r”表示红色)，按轴缩放并索引到当前颜色图的标量值，或具有精确TrueColor的3元素RGB矢量。

如果c是一个长度为N的向量，那么第i个多边形的颜色将由根据坐标轴缩放条目c(i)决定，然后索引到当前颜色图。更复杂的着色情况需要直接操作补丁属性/值对。

不需要通过矩阵x和y来指定多边形，而是可以呈现一个唯一的顶点列表，然后是一个从这些顶点创建的多边形面列表。在这种情况下，“顶点”矩阵将是Nx2 (2-D补丁)或Nx3 (3-D补丁)。MxN“Faces”矩阵描述了M个具有N个顶点的多边形——每一行描述一个多边形，每一列条目是“顶点”矩阵的一个索引，用于识别一个顶点。patch对象可以通过直接传递属性/值对“顶点”/vert，“面”/ Faces作为输入来创建。

不使用属性/值对，任何属性都可以通过传递带有各自字段名的结构propstruct来设置。

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。

可选的返回值h是创建的补丁对象的图形句柄。

编程注意:

1. 完整的属性列表记录在修补进程属性中。有用的修补进程属性包括：“cdata”、“edgecolor”、“facecolor”、“faces”、“vertices”和“facevertexcdata”。
2. 指定面片几何形状的属性（“xdata”、“ydata”、“zdata”、“faces”、“vertices”）应在其他属性之前指定，以避免意外的绘图输出或错误。
3. 混合定义几何的 x-y-z 和面顶点形式可能会产生意外的几何结果。
4. 将“cdata”颜色定义与面顶点定义几何或“facecentercdata”颜色定义与 x-y-z 定义几何一起使用时，可能会出现意外的色块颜色结果。

参见:fill, get, set。

: surface (x, y, z, c)

: surface (x, y, z)

: surface (z, c)

: surface (z)

: surface (…, prop, val, …)

: surface (hax, …)

: h = surface (…)

从网格中给定矩阵x和y，创建一个表面图形对象，矩阵z值对应于表面的x和y坐标。

如果x和y是向量，那么一个典型的顶点是(x(j)， y(i)， z(i,j))。因此，z的列对应不同的x值，z的行对应不同的y值。如果只给出一个输入z，则取x为1:列(z)， y为1:行(z)。

任何属性/值输入对都被分配给表面对象。

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。

可选的返回值h是创建的表面对象的图形句柄。

编程注意:完整的属性列表记录在表面属性中。

参见:surf, mesh, patch, line。

: light ()

: light (…, "prop", val, …)

: light (hax, …)

: h = light (…)

在当前轴或轴hax上创建一个光对象。

当一个光照对象存在于一个轴对象中，并且一个补丁或表面对象的属性“EdgeLighting”或“facellighting”被设置为非“none”的值时，这些对象将被绘制为光照效果。支持的照明属性值是“none”(没有照明效果)，“flat”(对象的多面外观)和“gouraud”(顶点之间的照明效果的线性插值)。如果照明模式设置为“平”，则使用“FaceNormals”属性进行照明。对于“gouraud”，使用“VertexNormals”属性。

每个轴最多支持八个轻物体。(实现的依赖)

照明仅支持OpenGL图形工具包(即“fltk”和“qt”)。

光对象具有以下属性，可以改变绘图的外观。

“颜色”:光的颜色可以通过

rgb -矢量(例如，[1 0 0]表示红色)或字符串(例如，“r”表示红色)。默认颜色是白色([1 1 1])。

“位置”:光发出的方向

1 x3-vector。缺省方向为[1 0 1]。

“Style”:这个字符串定义了

光是来自无限大距离的光源(“infinite”)还是来自局部点源(“local”)。默认值是“infinite”。

如果第一个参数hax是一个轴句柄，那么将light对象添加到该轴，而不是gca返回的当前轴。

可选的返回值h是创建的光对象的图形句柄。

编程注意:完整的属性列表记录在Light properties中。

参见:照明，材料，贴片，表面。

**15.3.2.2句柄函数**

要确定一个变量是图形对象索引，还是轴或图形的索引，可以使用ishhandle、isgraphics、isaxes和isfigure函数。

: tf = ishghandle (h)

如果h是图形句柄则返回true，否则返回false。

H也可以是句柄的矩阵，在这种情况下，返回一个逻辑数组，当H的元素是图形句柄时为真，当它们不是时为假。

参见:isgraphics, isaxes, isfigure, handle。

: tf = isgraphics (h)

: tf = isgraphics (h, type)

如果h是图形句柄(类型为type)则返回true，否则返回false。

当没有指定类型时，该函数等价于ishhandle。

参见:ishhandle, handle, isaxes, isfigure。

: tf = ishandle (h)

如果h是图形或Java对象的句柄，则返回true，否则返回false。

h也可以是句柄矩阵，在这种情况下，返回一个逻辑数组，如果h的元素是图形或Java对象的句柄，则返回true，如果不是false。

编程注意:测试特定的对象类型通常更有用。要确定句柄是否属于图形对象，请使用ishhandle或isgraphics。要确定句柄是否属于Java对象，请使用isjava。

参见:ishhandle, isgraphics, isjava。

: tf = isaxes (h)

如果h是轴图形句柄则返回true，否则返回false。

如果h是一个矩阵，那么返回一个逻辑数组，当h的元素是轴图形句柄时为真，当它们不是时为假。

参见:isfigure, ishhandle, isgraphics。

: tf = isfigure (h)

如果h是图形句柄则返回true，否则返回false。

如果h是一个矩阵，那么返回一个逻辑数组，当h的元素是图形句柄时为真，当它们不是时为假。

参见isaxes, ishhandle, isgraphics。

函数gcf返回当前图形对象的索引，如果不存在则创建索引。类似地，gca返回当前的轴对象，如果不存在，则创建一个(及其父图形对象)。

: h = groot ()

返回根图形对象的句柄。

根图形对象是所有图形对象的最终父对象。

此外，根对象包含有关图形系统作为一个整体的信息，如屏幕大小。使用get (root)来查找可用的信息。

图形系统作为一个整体的默认值是通过设置以“Default”开头的根图形对象的属性来指定的。例如，要将所有文本对象的默认字体设置为FreeSans使用

set (groot, "DefaultTextFontName", "FreeSans")

默认属性可以通过使用set和特殊属性值“remove”来删除。要撤销上述默认字体分配，请使用

set (groot, "DefaultTextFontName", "remove")

编程注意:根图形对象由特殊句柄值0标识。在某些时候，这个惟一的值可能会改变，但是可以通过使用保证总是返回根图形对象的root使代码抵抗将来的改变。

参见:gcf, gca, get, set。

: h = gcf ()

返回当前图形的句柄。

当前图形是图形输出的默认目标。如果存在多个图形，gcf将返回最后创建的图形或用鼠标单击的最后一个图形。

如果当前图形不存在，则创建一个并返回其句柄。然后可以使用句柄来检查或设置图形的属性。例如,

fplot (@sin, [-10, 10]);

fig = gcf ();

set (fig, "numbertitle", "off", "name", "sin plot")

绘制正弦波，找到当前图形的句柄，然后重命名图形窗口以描述其内容。

注:如果当前图形不存在，不需要新建，需要查询根图形对象的“CurrentFigure”属性。

Get (root， "currentfigure");

参见:gca, gco, gcbf, gcbo, get, set。

: h = gca ()

返回当前轴对象的句柄。

当前轴是图形输出的默认目标。对于具有多个轴的图形，gca返回最后创建的轴或用鼠标单击的最后一个轴。

如果当前不存在轴对象，则创建一个并返回其句柄。然后可以使用手柄来检查或设置轴的属性。例如,

ax = gca ();

set (ax, "position", [0.5, 0.5, 0.5, 0.5]);

创建空轴对象，然后更改其在图形窗口中的位置和大小。

注意:要查找当前轴而不创建新的轴对象(如果它不存在)，请查询图形的“CurrentAxes”属性。

get (gcf, "currentaxes");

参见:gcf, gco, gcbf, gcbo, get, set。

: h = gco ()

: h = gco (hfig)

返回当前图形的当前对象的句柄，或者返回具有句柄hfig的图形的当前对象的句柄。

图形的当前对象是上次点击的对象。它存储在目标图形的“CurrentObject”属性中。

如果最后一次鼠标单击没有发生在图形的任何子对象上，则当前对象就是图形本身。

如果目标图中没有发生鼠标单击，则此函数返回一个空矩阵。

编程注意:这个函数返回的值不一定与gbo在回调执行期间返回的值相同。一个正在执行的回调可以被另一个回调中断，并且当前对象可能被更改。

参见:gcbo、gca、gcf、gcbf、get、set。

get和set函数可以用来检查和设置图形对象的属性。例如,

get (groot)

⇒ ans =

{

type = root

currentfigure = [](0x0)

children = [](0x0)

visible = on

…

}

返回一个包含根图形对象的所有属性的结构。与Octave中的所有函数一样，该结构是按值返回的，因此修改它不会修改内部根对象。要做到这一点，必须使用set函数。另外，请注意，在本例中，currentfigure属性为空，这表明没有当前图形窗口。

get函数也可用于查找单个属性的值。例如,

get (gca (), "xlim")

⇒ [ 0 1 ]

返回当前图形中当前轴对象的x轴范围。

要设置图形对象的属性，使用set函数。例如,

set (gca (), "xlim", [-10, 10]);

将当前图形中当前轴对象的x轴范围设置为'[- 10,10]'。

如果不带值参数调用set函数，也可以查询默认属性值。当只给出一个参数(图形句柄)时，则返回一个具有给定对象类型的所有属性默认值的结构。例如,

set (gca ())

返回包含轴对象默认属性值的结构。如果用两个参数(一个图形句柄和一个属性名)调用set，则只返回所请求属性的默认值。

: val = get (h)

: val = get (h, p)

从图形句柄h返回命名属性p的值。

如果省略p，则返回h的完整属性列表。

如果h是矢量，则返回一个单元格数组，其中分别包含属性值或列表。

参见:set。

: set (h, property, value, …)

: set (h, {properties}, {values})

: set (h, pv)

: value\_list = set (h, property)

: all\_value\_list = set (h)

为图形句柄(或图形句柄向量)设置命名属性值h。

提供属性名称和值的方法有三种:

作为逗号分隔的属性、值对列表

每个属性都是包含属性名称的字符串，每个值都是属性的适当类型的值。当h中有多个句柄时，每个句柄都被分配相同的值。例如:

h = plot ([0, 1]);

set (h, 'color', 'green');

作为包含属性名称的字符串属性和包含属性值的单元格数组的单元格数组。

在这种情况下，值的列数必须与属性中的元素数相匹配。第一列的值包含属性中第一个条目的值，等等。值的行数必须为1或匹配h的元素数。在第一种情况下，h中的每个句柄将被分配相同的值。在第二种情况下，h中的第一个句柄将被分配第一行值中的值，以此类推。例如:

h = plot ([0, 1; 1, 0]);

set (h, {'color'}, {'green'; 'red'});

作为结构pv

这与第一种情况相同，其中pv的字段名表示属性名，字段值给出属性值。与第一种情况一样，只能为一个属性设置一个值，该属性将应用于h中的所有句柄。例如:

h = plot ([0, 1]);

props.color = 'green';

set (h, props);

Set还用于查询指定属性的值列表。Clist = set (h， "property")将返回单元格列表列表中"property"的可能值列表。如果没有使用输出变量，则格式化列表并打印到屏幕上。

例如：

hf = figure ();

set (hf, "paperorientation")

⇒ [ landscape | {portrait} ]

显示纸张方向属性可以有两个值，默认值是“纵向”。

参见:get。

: parent = ancestor (h, type)

: parent = ancestor (h, type, "toplevel")

返回句柄对象h的第一个祖先，其类型匹配type，其中type是一个字符串。

如果type是字符串的单元格数组，则返回类型与任何给定类型字符串匹配的第一个父元素。

如果句柄对象h本身是type类型，则返回h。

如果"toplevel"作为第三个参数给出，则返回匹配条件的对象层次结构中最高的父级，而不是第一个(最接近的)父级。

参见:findobj, findall, allchild。

: h = allchild (handles)

查找图形对象的所有子对象，包括隐藏的子对象。

此函数类似于get (h， "children")，但也返回隐藏对象(HandleVisibility = "off")。

如果handle是一个标量，h就是一个向量。否则，h将是一个与句柄大小相同的单元矩阵，每个单元将包含一个句柄向量。

参见:findall, findobj, get, set。

: findfigs ()

找到当前不在屏幕上的所有可见图形，并将它们移动到屏幕上。

参见:allchild, figure, get, set。

可以使用打印和保存功能以多种图形格式打印或保存图形。但是，有时直接保存原始的Octave句柄图形可能是有用的，以便可以进行进一步的修改，例如修改标题或图例。

这可以通过以下函数来完成

fig\_struct = hdl2struct (gcf);

save myplot.fig -struct fig\_struct;

…

fig\_struct = load ("myplot.fig");

struct2hdl (fig\_struct);

: s = hdl2struct (h)

返回一个结构s，其字段描述对象及其子对象的属性，这些属性与句柄h相关联。

结构s的字段是“type”、“handle”、“properties”、“children”和“special”。

参见:struct2hdl, hgsave, findobj。

: h = struct2hdl (s)

: h = struct2hdl (s, p)

: h = struct2hdl (s, p, hilev)

从结构s构造一个图形句柄对象h。

该结构必须包含“handle”、“type”、“children”、“properties”和“special”字段。

如果指定了现有图形或轴的句柄p，则新对象将作为该对象的子对象创建。如果没有提供父句柄，则将使用根对象的默认值构造一个新图和必要的子图。

第三个布尔参数hilev可以用来指定函数是否应该保留监听器/回调，例如，对于图例或hggroups。默认为false。

参见:hdl2struct, hgload, findobj。

: hnew = copyobj (horig)

: hnew = copyobj (horig, hparent)

构造与句柄horing相关联的图形对象的副本，并将新句柄hnew返回给新对象。

如果指定了父句柄hparent(根、图、轴或组)，则复制的对象将作为hparent的子对象创建。

如果horig是句柄的向量，而hparent是一个标量，则vector hnew中的每个句柄都有其“Parent”属性设置为hparent。相反，如果horig是一个标量，hparent是一个向量，那么每个父对象都将收到horig的副本。如果horing和hparent都是具有相同元素数量的向量，那么hnew(i)将具有parent hparent(i)。

参见:struct2hdl, hdl2struct, findobj。

**15.3.3图形对象属性**

在本节中，将详细讨论图形对象属性，从根属性开始，一直到对象层次结构。有关特定图形对象的文档可以使用 doc 功能显示，例如，doc（“坐标区属性”）将显示坐标区属性。

可以通过编程方式检索单选（字符串）属性的允许值，也可以使用 set 函数的一个或两个参数调用形式显示。请参见set。

Octave 会接受以下文档中标记为未使用或未使用的任何属性，而不会出现错误。这些属性的值存储在对象中，但对对象没有影响。

默认属性值包含在 { } 中。

**15.3.3.1根属性**

**类别：**

回调执行 |命令窗口显示 |鼠标交互 |物体识别 |家长/子女 |指针信息 |屏幕信息 |闲置

**回调执行**

callbackobject (read-only): graphics handle, def. [](0x0)

当前对象的图形句柄,其回调正在执行。

**命令窗口显示**

commandwindowsize (read-only): two-element vector, def. [0 0]

新创建的命令窗口中显示的列数和行数。

fixedwidthfontname: string, def. "Courier"

当“fontname”属性设置为“FixedWidth”时将用于图形对象的固定宽度字体的名称。

**鼠标交互**

contextmenu: graphics handle, def. [](0x0)

Graphics handle of the uicontextmenu object that is currently associated to this root object.

**物体识别**

currentfigure: graphics handle, def. [](0x0)

图形处理当前数字。

tag: string, def. ""

用户定义的字符串,用于标记图形对象。

type (read-only): string

图形对象的类名,类型总是"root"。

userdata: Any Octave data, def. [](0x0)

用户定义的数据,以与图形对象关联。

**家长/子女**

children (read-only): vector of graphics handles, def. [](0x1)

图形处理根的子女。

handlevisibility: "callback" | "off" | {"on"}

根对象句柄始终可见。将此设置更改为“回调”或“关闭”不起作用。

parent: graphics handle, def. [](0x0)

根对象没有父图形对象,父对象总是空的。

showhiddenhandles: {"off"} | "on"

如果 showhiddenhandles 为“on”，则所有图形对象句柄在其父项的子级列表中都可见，而不考虑其 handlevisibility 属性的值。

**指针信息**

pointerlocation: two-element vector, def. [0 0]

尚未为根对象实现全局指针位置跟踪。PointerLocation 未使用。根对象的指针分配将始终为 [0， 0]。

pointerwindow (read-only): graphics handle, def. 0

尚未为根对象实现指针窗口跟踪。PointerWindow 未使用。根对象的 PointerWindow 值将始终为 0。

**屏幕信息**

monitorpositions (read-only): four-element vector

报告连接显示器的宽度和高度。注意：Octave 仅部分实现 monitorpositions。只有有关主监视器的信息存储在监视器位置中，这与存储在“screensize”属性中的信息相同。

screendepth (read-only): double

显示器的每像素比特颜色深度。

screenpixelsperinch (read-only): double

主显示器的屏幕分辨率，以每英寸像素为单位。

screensize (read-only): four-element vector

主显示器的大小表示为四元素矢量 [左、下、宽、高]。

units: "centimeters" | "characters" | "inches" | "normalized" | {"pixels"} | "points"

用于“monitorpositions”、“pointerlocation”和“screensize”属性的单位类型。

**闲置**

beingdeleted: {"off"} | "on"

beingdeleted 是未使用的。

busyaction: "cancel" | {"queue"}

busyaction 是未使用的。

buttondownfcn: string | function handle, def. [](0x0)

buttondownfcn是未使用的。

clipping: "off" | {"on"}

clipping 是未使用的。

createfcn: string | function handle, def. [](0x0)

createfcn 是未使用的。

deletefcn: string | function handle, def. [](0x0)

deletefcn 是未使用的。

hittest: "off" | {"on"}

hittest是未使用的。

interruptible: "off" | {"on"}

interruptible是未使用的。

pickableparts: "all" | "none" | {"visible"}

pickableparts是未使用的。

selected: {"off"} | "on"

selected是未使用的。

selectionhighlight: "off" | {"on"}

selectionhighlight是未使用的。

visible: "off" | {"on"}

visible是未使用的。

**15.3.3.2图属性**

图形对象的属性(见图):

**类别：**

外观 |回调执行 |创建/删除 |显示器 |键盘交互 |鼠标交互 |物体识别 |对象位置 |家长/子女 |打印/保存 |闲置

**外观**

alphamap: def. 64-by-1 double

尚未为图形对象实现透明度。AlphaMap 未使用。

color: colorspec, def. [1 1 1]

图案背景的颜色。请参阅色彩规格。

colormap: N-by-3 matrix, def. 64-by-3 double

包含当前轴的 RGB 颜色映射的矩阵。

graphicssmoothing: "off" | {"on"}

使用平滑技术来减少锯齿状线条的出现。

name: string, def. ""

要在图标题栏中显示的名称。该名称显示在由 numbertitle 属性确定的任何标题的右侧。

numbertitle: "off" | {"on"}

在图标题栏中显示“图”，后跟数字图句柄值。

**回调执行**

busyaction: "cancel" | {"queue"}

定义 Octave 在无法中断另一个对象的执行回调时如何处理此对象的回调属性的执行。仅当当前正在执行的回调对象的 interruptible 属性设置为“off”时，这才有意义。中断回调对象的 busyaction 属性指示中断回调是排队（“queue”（默认））还是丢弃（“cancel”）。请参阅回调部分。

interruptible: "off" | {"on"}

指定此对象的回调函数是否会被其他回调中断。默认情况下，interruptible 为 “on”，使用 drawnow、figure、waitfor、getframe 或 pause 函数的回调最终会中断。请参阅回调部分。

**创建/删除**

beingdeleted: {"off"} | "on"

指示函数已启动对象删除的属性。beingdeleted 设置为 true，直到对象不再存在。

closerequestfcn: string | function handle, def. "closereq"

删除图形时执行的函数。请参见 closereq 函数。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

createfcn: string | function handle, def. [](0x0)

创建图后立即执行回调函数。函数是通过使用根对象上的默认属性来设置的，例如，set （groot， “defaultfigurecreatefcn”， 'disp （“figure created！”）'）。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

创建图后立即执行回调函数。函数是通过使用根对象上的默认属性来设置的，例如，set （groot， “defaultfigurecreatefcn”， 'disp （“figure created！”）'）。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

**显示器**

visible: "off" | {"on"}

如果 visible 为"关闭",则图形不会显示在屏幕上。

windowstate: "fullscreen" | "maximized" | "minimized" | {"normal"}

尚未为图形对象实现窗口状态调整。windowState 未使用。

windowstyle: "docked" | "modal" | {"normal"}

图形的窗口样式。以下值之一：

正常

窗口可能未选中，其他窗口可能显示在窗口前面。

模 态

该窗口将保持在所有正常数字的顶部，直到它被关闭。

停靠

未实现。

更改可见图形的模式可能会导致图形关闭并重新打开。

**键盘交互**

keypressfcn: string | function handle, def. [](0x0)

当图形具有焦点时发生击键事件时执行的回调函数。可以使用函数的第二个参数“evt”检索按下的实际键。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

keyreleasefcn: string | function handle, def. [](0x0)

使用 keypressfcn，键盘回调函数。这些回调函数分别在按下/释放键时调用。这些函数使用两个输入参数调用。第一个参数保存调用图形的句柄。第二个参数包含一个事件结构，该结构具有以下成员：

字符：

密钥的 ASCII 值

钥匙：

键的小写值

修饰语：

一个元胞数组，其中包含表示用键按下的修饰符的字符串。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

windowkeypressfcn: string | function handle, def. [](0x0)

按下某个键并且图形具有焦点时执行的功能。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

windowkeyreleasefcn: string | function handle, def. [](0x0)

松开键且图形具有焦点时执行的函数。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

**鼠标交互**

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形侦听器函数的信息，请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

当前与此图示对象关联的 uicontextmenu 对象的图形句柄。

currentpoint (read-only): two-element vector, def. [0; 0]

一个 1×2 矢量，用于保存鼠标事件发生时鼠标指针所在点的坐标。X 和 Y 坐标以图示的单位属性定义，其原点位于绘图区域的左下角。

设置当前点的事件是

按下鼠标按钮

始终

释放鼠标按钮

只有定义了图形的回调 windowbuttonupfcn。

按下鼠标按钮(拖动)时移动指针

只有在定义了图形的回调窗口 windowbuttonmotionfcn。

hittest: "off" | {"on"}

指定 figure 是处理鼠标事件还是将其传递给对象的祖先。启用后，对象可以通过评估“buttondownfcn”、显示 uicontextmenu 并最终成为根“currentobject”来响应鼠标单击。仅当对象可以接受鼠标单击时，此属性才相关，鼠标单击由“pickableparts”属性确定。请参见pickableparts 属性。

menubar: {"figure"} | "none"

控制图形顶部图形菜单栏的显示。

pointer: {"arrow"} | "botl" | "botr" | "bottom" | "circle" | "cross" | "crosshair" | "custom" | "fleur" | "hand" | "ibeam" | "left" | "right" | "top" | "topl" | "topr" | "watch"

与图窗画布关联的鼠标指针形状的名称。当指针为“自定义”时，形状由 pointershapecdata 属性确定。

当图形处于缩放、平移或旋转模式时，指针不起作用。在这种情况下，Octave 会自动使用适合该模式的指针形状。

pointershapecdata: 16-by-16 or 32-by-32 Matrix, def. 16-by-16 double

定义自定义指针的 m-by-m 矩阵。每个元素定义一个像素，元素 （1,1） 表示左上角的像素。值 1 为黑色，值为 2 为白色，所有其他值均呈现为透明。

pointershapehotspot: two-element vector, def. [1 1]

对于自定义指针，只有 pointershapehotspot 定义用作指针位置的 pointershapecdata 中像素的行和列。

resize: "off" | {"on"}

通过使用鼠标拖动窗口边框和边角来控制是否可以调整图形的大小。当调整大小为“关闭”时，鼠标交互将被禁用，但仍可以通过更改其“位置”属性来调整图形的大小。

resizefcn: string | function handle, def. [](0x0)

resizefcn 已弃用,改用 sizechangedfcn。

selected: {"off"} | "on"

属性表示该数字是否被选中。

selectionhighlight: "off" | {"on"}

如果 selectionhighlight 为“on”，则图形的选择状态将直观地突出显示。

selectiontype: "alt" | "extend" | {"normal"} | "open"

最近一次鼠标单击的选择类型。

SelectionType 可能采用不同的值，具体取决于所使用的鼠标按钮和键盘修饰符的组合：

正常：

单击鼠标左键。

替代：

右键单击或 Ctrl+左键单击。

扩展：

Shift+左键单击、中键单击或组合左键单击和右键单击。

打开：

双击左键。

sizechangedfcn: string | function handle, def. [](0x0)

更改图窗大小时触发回调。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

toolbar: {"auto"} | "figure" | "none"

控制工具栏（沿菜单栏底部）和状态栏的显示。设置为“auto”时，显示基于“menubar”属性的值。

windowbuttondownfcn: string | function handle, def. [](0x0)

参见 windowbuttonupfcn 属性。

windowbuttonmotionfcn: string | function handle, def. [](0x0)

参见 windowbuttonupfcn 属性。

windowbuttonupfcn: string | function handle, def. [](0x0)

使用 windowbuttondownfcn 和 windowbuttonmotionfcn，鼠标回调函数。分别在按下、拖动或释放鼠标按钮时调用这些回调函数。执行这些回调函数时，currentpoint 属性将保存游标的当前坐标。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

windowscrollwheelfcn: string | function handle, def. [](0x0)

当用户在此图形上操作鼠标滚轮时执行的函数。该函数使用两个输入参数调用。第一个参数保存调用图形的句柄。第二个参数包含一个事件结构，该结构具有以下成员：

VerticalScrollCount：

滚轮步数，向下滚动时通常为 1，向上滚动时通常为 -1。

VerticalScrollAmount：

轮步应滚动的行数。此值始终为 3。

EventName:

事件名称为“WindowScrollWheel”。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

**物体识别**

currentaxes: graphics handle, def. [](0x0)

处理当前轴的图形对象。

currentcharacter (read-only): def. ""

尚未对图形对象实现对最后一个按下的键的跟踪。currentcharacter未使用。

currentobject (read-only): graphics handle, def. [](0x0)

处理图中最近活动的图形对象。

integerhandle: "off" | {"on"}

分配下一个最低的未使用整数作为图号。

nextplot: {"add"} | "new" | "replace" | "replacechildren"

NextPlot 由高级绘图函数用于决定如何处理图中已存在的轴。请参阅 newplot 函数。

number (read-only): double

当前数字的编号。

tag: string, def. ""

用户定义的字符串,用于标记图形对象。

type (read-only): string

图形对象的类名,类型总是"图形"。

userdata: Any Octave data, def. [](0x0)

用户定义的数据,以与图形对象关联。

**对象位置**

dockcontrols: "off" | {"on"}

尚未为图形对象实现交互式图形对接。DockControls 未使用。

innerposition: four-element vector, def. [300 200 560 420]

"内置"属性与"位置"属性相同。

outerposition: four-element vector, def. [-1 -1 -1 -1]

指定图形的位置和大小，包括顶部菜单栏和底部状态栏。矢量的四个元素是左下角的坐标和图形的宽度和高度。请参阅单位属性。

position: four-element vector, def. [300 200 560 420]

指定图窗的位置和大小。矢量的四个元素是左下角的坐标和图形的宽度和高度。请参阅单位属性。

units: "centimeters" | "characters" | "inches" | "normalized" | {"pixels"} | "points"

用于计算位置和外部位置属性的单位。

**家长/子女**

children (read-only): vector of graphics handles, def. [](0x1)

图中人物的孩子的图形处理。

handlevisibility: "callback" | "off" | {"on"}

如果 handlevisibility 为“off”，则图形的句柄在其父级的“children”属性中不可见。

parent: graphics handle, def. 0

父图形对象的句柄。

**打印/保存**

filename: string, def. ""

保存情节图时使用的文档名。

inverthardcopy: "off" | {"on"}

打印时将图形和轴的背景颜色替换为白色。

paperorientation: "landscape" | {"portrait"}

纸张尺寸和纸张位置属性的值取决于纸张方向。当纸张方向在“纵向”和“横向”之间切换时，纸张大小和纸张位置的水平和垂直值顺序相反。

paperposition: four-element vector, def. [1.3422 3.3191 5.8156 4.3617]

矢量 [左下角宽度高度] 定义图形在打印页面上的位置和大小（以纸张为单位）。位置 [左下角] 定义页面上图形的左下角，大小由 [宽度高度] 定义。对于未在纸张上隐式呈现的输出格式，宽度和高度定义图像的大小，并忽略位置信息。设置 paperposition 还会强制将 paperpositionmode 属性设置为“manual”。

paperpositionmode: {"auto"} | "manual"

如果 paperpositionmode 设置为“auto”，则会自动计算 paperposition 属性：打印的图形将与屏幕上的图形具有相同的大小，并将在输出页面上居中。将 paperpositionmode 设置为“auto”不会修改 paperposition 属性的值。

papersize: two-element vector, def. [8.5000 11.0000]

矢量 [宽度、高度] 定义用于打印的纸张尺寸。将 papersize 属性设置为与定义的 papertype 之一无关且与 paperoriented 的设置一致的值，会强制 papertype 属性为值“<custom>”。如果 papersize 设置为与支持的 papertype 关联的值，并且与 paperorientation 一致，则 papertype 值将修改为关联的值。

papertype: "<custom>" | "a" | "a0" | "a1" | "a2" | "a3" | "a4" | "a5" | "arch-a" | "arch-b" | "arch-c" | "arch-d" | "arch-e" | "b" | "b0" | "b1" | "b2" | "b3" | "b4" | "b5" | "c" | "d" | "e" | "tabloid" | "uslegal" | {"usletter"}

用于打印输出的纸张的名称。设置 papertype 也会更改纸张大小，同时保持与 paperorientation 属性的一致性。

paperunits: "centimeters" | {"inches"} | "normalized" | "points"

用于计算 paperposition 属性的单位。物理单位（例如，“英寸”）的转换取决于根对象的 screenpixelsperinch 属性。

renderer: {"opengl"} | "painters"

渲染器模式为“手动”时用于打印的渲染引擎。设置 renderer 还会强制将 renderermode 属性设置为“manual”。

renderermode: {"auto"} | "manual"

控制用于打印的呈现引擎是自动选择还是由 renderer 属性指定。请参阅打印功能。

**闲置**

clipping: "off" | {"on"}

clipping未使用。

pickableparts (read-only): "all" | "none" | {"visible"}

pickableparts未使用。

**15.3.3.3轴属性**

轴对象的属性(参见轴):

**类别：**

自动子属性 |轴箱外观 |轴网格外观 |回调执行 |相机和视图控制 |颜色和透明度 |创建/删除 |显示器 |鼠标交互 |物体识别 |对象位置 |家长/子女 |文本外观

**自动子属性**

colororder: N-by-3 RGB matrix, def. 7-by-3 double

图形函数用于子对象着色的 RGB 值。

colororderindex: integer, def. 1

“colororder”属性中要由 Axes-child 对象使用的下一个颜色的索引。

linestyleorder: def. "-"

要按轴子对象顺序使用的线型列表，指定为线规范字符串的元胞数组。请注意，线条样式仅在循环浏览完整的“colororder”列表后递增。请参阅线型。

linestyleorderindex: whole number scalar, def. 1

“linestyleorder”属性中要由 Axes-child 对象使用的下一个线型的索引。

nextseriesindex (read-only): whole number scalar, def. 1

“colororder”和“linestyleorder”属性中的当前索引值，指示下一个子对象将使用的项

**轴箱外观**

box: {"off"} | "on"

控制轴是否有一个周围的盒子。

boxstyle: {"back"} | "full"

对于 3-D 轴，控制是绘制“完整”框还是仅绘制 3 个“后”轴。

color: colorspec, def. [1 1 1]

轴背景的颜色。请参阅色彩规格。

dataaspectratio: three-element vector, def. [1 1 1]

指定轴中显示的数据的相对高度和宽度。将 dataaspectratio 设置为 [1， 2] 会导致 x 轴上显示的一个单位的长度与 y 轴上 2 个单位的长度相同。请参见 daspect 函数。设置 dataaspectratio 还会强制将 dataaspectratiomode 属性设置为“manual”。

dataaspectratiomode: {"auto"} | "manual"

数据纵横比模式的当前状态，由“dataaspectratio”属性手动设置，或由 Octave 结合其他显示属性自动设置，以适应当前视图中的数据。

layer: {"bottom"} | "top"

控制轴是在子图形对象（被打印对象覆盖的刻度、标签等）下方绘制还是在上方绘制。

layout (read-only): def. [](0x0)

尚未为轴对象实现平铺和网格化图表布局。布局未使用。

linewidth: scalar, def. 0.5000

主要轴线的宽度。

tickdir: "both" | {"in"} | "none" | "out"

控制轴刻度线是“向内”投影到打印框还是“向外”投影。值“both”将画出刻度线。值“none”表示不会绘制刻度线，但仍会呈现刻度标记。设置 tickdir 还会强制将 tickdirmode 属性设置为“manual”。请注意，对于二维图，默认值为“in”，对于三维图，默认值为“out”。

tickdirmode: {"auto"} | "manual"

tickdir 模式的当前状态，由“tickdir”属性手动设置，或自动设置为当前视图的默认值。

ticklength: two-element vector, def. [0.010000 0.025000]

双元素矢量 [2Dlen 3Dlen] 指定刻度线相对于最长可见轴的长度。

toolbar (read-only): def. [](0x0)

尚未为 axes 对象实现 AxesToolbar 对象。工具栏未使用。

xaxis (read-only): def. [](0x0)

尚未为 axes 对象实现 Axes Ruler 对象。xaxis 未使用。

xaxislocation: {"bottom"} | "origin" | "top"

控制x轴位置。

xcolor: {colorspec} | "none", def. [0.1500 0.1500 0.1500]

x 轴的颜色。请参阅 colorspec。设置 xcolor 还会强制将 xcolormode 属性设置为“manual”。

xcolormode: {"auto"} | "manual"

确定应用于 x 轴网格线的颜色的设置的当前状态。如果设置为“auto”和/或“gridcolormode”属性设置为“manual”，则 x 轴网格颜色将由“gridcolor”属性定义。否则，x 轴网格颜色将由“xcolor”属性定义。

xdir: {"normal"} | "reverse"

x 轴的方向：“法线”在默认的 2D 和 3D 视图中从左到右。

xlim: two-element vector, def. [0 1]

双元素矢量 [xmin xmax] 指定 x 轴的限制。设置 xlim 还会强制将 xlimmode 属性设置为“manual”。请参见 xlim 函数。

xlimitmethod: "padded" | {"tickaligned"} | "tight"

当 xlimmode 属性为“auto”时，用于确定 x 轴限制的方法。默认值“tickaligned”使限制与最接近的刻度对齐。当值为“tight”时，将调整限制以将所有图形对象封闭在轴中，而当值为“ppaid”时，将在对象周围添加约 7% 的额外边距数据范围。请参阅轴功能。

xlimmode: {"auto"} | "manual"

x 轴极限选择方法的当前状态，可以使用“xlim”属性手动设置，也可以根据“xlimitmethod”属性自动设置为跨越绘制的数据。

xminortick: {"off"} | "on"

控制是否显示小x标记。

xtick: vector

x 刻度线的位置。设置 xtick 还会强制将 xtickmode 属性设置为“manual”。

xtickmode: {"auto"} | "manual"

用于确定 xtick 位置和间距是由 Octave 自动设置还是使用“xtick”属性手动设置的设置。

yaxis (read-only): def. [](0x0)

尚未为 axes 对象实现 Axes Ruler 对象。yaxis 未使用。

yaxislocation: {"left"} | "origin" | "right"

控制y轴位置。

ycolor: {colorspec} | "none", def. [0.1500 0.1500 0.1500]

y 轴的颜色。请参阅颜色规范。

ycolormode: {"auto"} | "manual"

确定应用于 y 轴网格线的颜色的设置的当前状态。如果设置为“auto”和/或“gridcolormode”属性设置为“manual”，则 y 轴网格颜色将由“gridcolor”属性定义。否则，y 轴网格颜色将由“ycolor”属性定义。

ydir: {"normal"} | "reverse"

y 轴方向：“正常”在 2D 中从下到上，在 3D 默认视图中从前到后。

ylim: two-element vector, def. [0 1]

双元素矢量 [ymin ymax] 指定 y 轴的极限。设置 ylim 还会强制将 ylimmode 属性设置为“manual”。请参见 ylim 函数。

ylimitmethod: "padded" | {"tickaligned"} | "tight"

当 xlimmode 属性为“auto”时，用于确定 y 轴限制的方法。默认值“tickaligned”使限制与最接近的刻度对齐。当值为“tight”时，将调整限制以将所有图形对象封闭在轴中，而当值为“ppaid”时，将在对象周围添加约 7% 的额外边距数据范围。请参阅轴功能。

ylimmode: {"auto"} | "manual"

y 轴极限选择方法的当前状态，可以使用“ylim”属性手动设置，也可以根据“ylimitmethod”属性自动设置为跨越绘制的数据。

ytick: vector

y 刻度线的位置。设置 ytick 还会强制将 ytickmode 属性设置为“manual”。

ytickmode: {"auto"} | "manual"

用于确定 ytick 位置和间距是由 Octave 自动设置还是使用“ytick”属性手动设置的设置。

zaxis (read-only): def. [](0x0)

尚未为 axes 对象实现 Axes Ruler 对象。Zaxis 未使用。

zcolor: {colorspec} | "none", def. [0.1500 0.1500 0.1500]

z 轴的颜色。请参阅colorspec。

zcolormode: {"auto"} | "manual"

确定应用于 z 轴网格线的颜色的设置的当前状态。如果设置为“auto”和/或“gridcolormode”属性设置为“manual”，则 z 轴网格颜色将由“gridcolor”属性定义。否则，z 轴网格颜色将由“zcolor”属性定义。

zdir: {"normal"} | "reverse"

y 轴的方向：“正常”在默认的 3D 视图中从下到上。

zlim: two-element vector, def. [0 1]

二元素矢量 [zmin zmax] 指定 z 轴的极限。设置 zlim 还会强制将 zlimmode 属性设置为“manual”。请参见 zlim 函数。

zlimitmethod: "padded" | {"tickaligned"} | "tight"

当 xlimmode 属性为“auto”时，用于确定 z 轴限制的方法。默认值“tickaligned”使限制与最接近的刻度对齐。当值为“tight”时，将调整限制以将所有图形对象封闭在轴中，而当值为“ppaid”时，将在对象周围添加约 7% 的额外边距数据范围。请参阅轴功能。

zlimmode: {"auto"} | "manual"

z 轴极限选择方法的当前状态，可以使用“zlim”属性手动设置，也可以根据“zlimitmethod”属性自动设置为跨越绘制的数据。

zminortick: {"off"} | "on"

控制是否显示较小的 z 勾标记。

ztick: vector

z 刻度线的位置。设置 ztick 还会强制将 ztickmode 属性设置为“manual”。

ztickmode: {"auto"} | "manual"

用于确定 ztick 位置和间距是由 Octave 自动设置还是使用“ztick”属性手动设置的设置。

**轴网格外观**

gridalpha: def. 0.1500

尚未为坐标区对象实现透明度。gridalpha 未使用。

gridalphamode: {"auto"} | "manual"

尚未为坐标区对象实现透明度。GridAlphamode 未使用。

gridcolor: {colorspec} | "none", def. [0.1500 0.1500 0.1500]

主要网格线的颜色。请参阅 colorspec。设置 gridcolor 还会强制将 gridcolormode 属性设置为“manual”。

gridcolormode: {"auto"} | "manual"

gridcolor 模式的当前状态，由“gridcolor”属性手动设置，或由 Octave 自动设置为默认值。

gridlinestyle: {"-"} | "--" | "-." | ":" | "none"

参见 Line Styles。

innerposition: four-element vector, def. [0.1300 0.1100 0.7750 0.8150]

"内置"属性与"位置"属性相同。

minorgridalpha: def. 0.2500

尚未为坐标区对象实现透明度。MinorgridAlpha 未使用。

minorgridalphamode: {"auto"} | "manual"

尚未为坐标区对象实现透明度。未使用minorgridalphamode。

minorgridcolor: {colorspec} | "none", def. [0.1000 0.1000 0.1000]

次要网格线的颜色。请参阅 colorspec。设置 minorgridcolor 还会强制将 minorgridcolormode 属性设置为“manual”。

minorgridcolormode: {"auto"} | "manual"

minorgridcolor 模式的当前状态，由“minorgridcolor”属性手动设置，或由 Octave 自动设置为默认值。

minorgridlinestyle: "-" | "--" | "-." | {":"} | "none"

参见 Line Styles。

xgrid: {"off"} | "on"

控制是否显示主要x网格线。

xminorgrid: {"off"} | "on"

控制是否显示小 x 网格线。

xscale: {"linear"} | "log"

将 x 轴设置为线性或对数比例。

ygrid: {"off"} | "on"

控制是否显示主要y网格线。

yminorgrid: {"off"} | "on"

控制是否显示小y网格线。

yminortick: {"off"} | "on"

检查是否显示小标记和勾号。

yscale: {"linear"} | "log"

将 y 轴设置为线性或对数比例。

zgrid: {"off"} | "on"

控制是否显示主要的z格线。

zminorgrid: {"off"} | "on"

控制是否显示较小的z网格线。

zscale: {"linear"} | "log"

将 z 轴设置为线性或对数比例。

**回调执行**

busyaction: "cancel" | {"queue"}

定义 Octave 在无法中断另一个对象的执行回调时如何处理此对象的回调属性的执行。仅当当前正在执行的回调对象的 interruptible 属性设置为“off”时，这才有意义。中断回调对象的 busyaction 属性指示中断回调是排队（“queue”（默认））还是丢弃（“cancel”）。请参阅回调部分。

interactions (read-only): def. [](0x0)

尚未为轴对象实现交互对象。交互未使用。

interruptible: "off" | {"on"}

指定此对象的回调函数是否会被其他回调中断。默认情况下，interruptible 为 “on”，使用 drawnow、figure、waitfor、getframe 或 pause 函数的回调最终会中断。请参阅回调部分。

**相机和视图控制**

cameraposition: three-element vector, def. [0.5000 0.5000 9.1603]

查看轴的相机位置坐标。设置 cameraposition 还会强制将 camerapositionmode 属性设置为“manual”。

camerapositionmode: {"auto"} | "manual"

相机位置属性的当前状态，无论是根据视图功能自动设置，还是使用“cameraposition”属性手动设置。

cameratarget: three-element vector, def. [0.5000 0.5000 0.5000]

查看摄像机瞄准点的坐标。设置 cameratarget 还会强制将 cameratargetmode 属性设置为“manual”。

cameratargetmode: {"auto"} | "manual"

相机目标属性的当前状态，可以使用“cameratarget”属性手动设置，也可以自动定位在轴图区域的中心。

cameraupvector: three-element vector, def. [0 1 0]

定义当前视图向上方向的 3 元素矢量。请注意，对于二维图，默认值为 [0 1 0]，对于三维图，默认值为 [0 0 1]。设置 cameraupvector 还会强制将 cameraupvectormode 属性设置为“manual”。

cameraupvectormode: {"auto"} | "manual"

相机向上矢量属性的当前状态，当“cameraupvector”属性用于将矢量从二维或三维默认值更改为手动时，设置为手动。

cameraviewangle: scalar, def. 6.6086

相机的视野定义为 0 到 180 度之间的角度。设置 cameraviewangle 还会强制将 cameraviewanglemode 属性设置为“manual”。

cameraviewanglemode: {"auto"} | "manual"

相机视角属性的当前状态，可以使用“cameraviewangle”属性手动设置，也可以通过 Octave 自动设置以包含所有可见对象。

projection: {"orthographic"} | "perspective"

尚未对轴对象实现正交/透视投影调整。投影未使用。

view: two-element vector, def. [0 90]

二元矢量 [方位角仰角] 指定三维图的视点。

**颜色和透明度**

alim: def. [0 1]

axes 对象尚未实现透明度,alim 未使用。

alimmode: {"auto"} | "manual"

尚未为坐标区对象实现透明度。Alimmode 未使用。

alphamap: def. [](0x0)

尚未为坐标区对象实现透明度。AlphaMap 未使用。

alphascale: {"linear"} | "log"

尚未为坐标区对象实现透明度。Alphascale 未使用。

ambientlightcolor: def. [1 1 1]

尚未为轴对象实现统一的背景轴照明。AmbientLightColor 未使用。

clim: two-element vector, def. [0 1]

定义具有 cdata 属性的轴子项的颜色轴的限制。设置 clim 还会强制将 climmode 属性设置为“manual”。

climmode: {"auto"} | "manual"

颜色限制模式的当前状态，可以由“clim”属性手动设置，也可以由 Octave 自动设置为轴子项的最小和最大 cdata 值。

colormap: N-by-3 matrix, def. 64-by-3 double

包含此轴对象的 RGB 颜色映射的矩阵。

colorscale: {"linear"} | "log"

尚未为坐标区对象实现自动线性/对数颜色缩放。色阶未使用。

**创建/删除**

beingdeleted: {"off"} | "on"

指示函数已启动对象删除的属性。beingdeleted 设置为 true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

allback 函数在创建轴后立即执行。函数是通过使用根对象上的默认属性来设置的，例如，set （groot， “defaultaxescreatefcn”， 'disp （“axes created！”）'）。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除坐标之前立即执行的回调函数。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

**显示**

clipping: "off" | {"on"}

轴子对象尚未实现轴子剪切控制,剪切未使用。

clippingstyle: {"3dbox"} | "rectangle"

Axes-children 裁剪控制尚未针对 Axes 对象实现,裁剪样式未使用。

visible: "off" | {"on"}

如果 visible 为"关闭",则轴不会在屏幕上呈现。

**鼠标交互**

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形监听器功能的信息,请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

当前与此轴对象关联的 uicontextmenu 对象的图形句柄。

currentpoint: 2-by-3 matrix, def. 2-by-3 double

矩阵 [xf, yf, zf; xb, yb, zb] 保存鼠标按键按下时鼠标指针所在点的坐标(以轴数据单位为单位)。如果定义了鼠标回调函数,则 currentpoint 保存鼠标按键按下时的指针坐标。对于三维绘图,返回矩阵的第一行指定最接近当前相机位置的点,第二行指定最远的点。这两个点形成一条垂直于屏幕的线。

hittest: "off" | {"on"}

指定axes是否处理鼠标事件或将其传递给对象的祖先。启用时,对象可以通过评估"buttondownfcn"来响应鼠标单击,显示uicontextmenu,并最终成为根"当前对象"。此属性仅在对象可以接受鼠标单击时才相关,这由"pickableparts"属性确定。请参阅pickableparts属性。

mousewheelzoom: scalar in the range (0, 1), def. 0.5000

轴的分数限制了每个车轮运动的变焦。

pickableparts: "all" | "none" | {"visible"}

指定轴是否接受鼠标点击。默认情况下,pickableparts 是"可见"的,只有轴的可见部分或其子节点才可能对鼠标点击做出反应。当 pickableparts 是"所有"时,可见和不可见部分(或子节点)都可能对鼠标点击做出反应。当 pickableparts 是"无"时,对对象的鼠标点击将被忽略并传输到该对象下方的任何对象。当一个对象被配置为接受鼠标点击时,"hittest" 属性将决定它们如何被处理。参见 hittest 属性。

selected: {"off"} | "on"

属性指示是否选择此轴。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为"on",则轴的选择状态在视觉上被突出显示。

**物体识别**

nextplot: "add" | {"replace"} | "replacechildren"

nextplot 被高级绘图函数用来决定如何处理已经存在于轴上的图形对象。参见 newplot 函数。nextplot 的状态通常使用 hold 函数来控制。参见 hold 函数。

tag: string, def. ""

用户定义的字符串,用于标记图形对象。

title: graphics handle

图形处理标题文本对象。

type (read-only): string

图形对象类型的类名始终为"axes"。

userdata: Any Octave data, def. [](0x0)

用户定义的数据,以与图形对象关联。

**位置对象**

outerposition: four-element vector, def. [0 0 1 1]

指定绘图的位置，包括标题、轴和图例。矢量的四个元素是左下角的坐标以及绘图的宽度和高度，单位归一化为绘图窗口的宽度和高度。例如，[0.2， 0.3， 0.4， 0.5] 将轴的左下角设置为 （0.2， 0.3），宽度和高度分别为 0.4 和 0.5。请参阅 position 属性。

plotboxaspectratio: def. [1 1 1]

请参见pbaspect 函数。设置 plotboxaspectratio 还会强制将 plotboxaspectratiomode 属性设置为“manual”。

plotboxaspectratiomode: {"auto"} | "manual"

绘图框纵横比模式的当前状态，可以由“dataaspectratio”属性手动设置，也可以由 Octave 结合其他显示属性自动设置，以适应当前视图中的数据。

position: four-element vector, def. [0.1300 0.1100 0.7750 0.8150]

指定打印的位置，不包括标题、轴和图例。矢量的四个元素是左下角的坐标以及绘图的宽度和高度，单位归一化为绘图窗口的宽度和高度。例如，[0.2， 0.3， 0.4， 0.5] 将轴的左下角设置为 （0.2， 0.3），宽度和高度分别为 0.4 和 0.5。请参阅 outerposition 属性。

positionconstraint: "innerposition" | {"outerposition"}

指定当轴注释范围更改时，哪个“内位”或“外位”属性优先。请参阅“innerposition”属性和“outerposition”属性。

units: "centimeters" | "characters" | "inches" | {"normalized"} | "pixels" | "points"

用于解释“position”、“outerposition”和“tightinset”属性的单位。

**家长/子女**

children (read-only): vector of graphics handles, def. [](0x1)

轴子项的图形控点。

handlevisibility: "callback" | "off" | {"on"}

如果 handlevisibility 为“off”，则轴的句柄在其父级的“children”属性中不可见。

parent: graphics handle

父图形对象的句柄。

sortmethod: "childorder" | {"depth"}

尚未为轴对象实现子显示顺序控制。Sort方法未使用。

**文本外观**

fontangle: "italic" | {"normal"}

控制字体是斜体还是普通。

fontname: string, def. "\*"

用于文本呈现的字体名称。设置此属性时，文本呈现引擎将在系统中搜索匹配的字体。如果未找到任何文本，则使用默认的无衬线字体（与默认的“\*”值相同）呈现文本。

编程说明：在不本机使用 FontConfig 的系统（除 Linux 之外的所有系统）上，字体缓存是在安装 Octave 时构建的。安装新字体后，您需要手动运行系统（“fc-cache -fv”）。

fontsize: scalar, def. 10

用于文本呈现的字体大小。请参见fontunits 属性。设置 fontsize 还会强制将 fontsizemode 属性设置为“manual”。

fontsizemode: {"auto"} | "manual"

fontsize 模式的当前状态，可以由“fontsize”属性手动设置，也可以由 Octave 自动设置以保持可读性。

fontsmoothing: "off" | {"on"}

控制与轴关联的任何文本是否消除锯齿。

fontunits: "centimeters" | "inches" | "normalized" | "pixels" | {"points"}

用于解释“fontsize”属性的单位。

fontweight: "bold" | {"normal"}

控制用于文本呈现的基本字体的变体。

labelfontsizemultiplier: def. 1.1000

x/y/zlabel 字体大小与刻度标签字体大小之间的比率。

legend (read-only): def. [](0x0)

尚未为坐标区对象实现图例属性控制。图例未使用。使用图例函数设置图例属性。

ticklabelinterpreter: "latex" | "none" | {"tex"}

控制 x/y/zticklabel 属性的解释方式。请参见“interpreter”属性的使用。

tightinset (read-only): four-element vector

包含标签和标题批注的轴周围的 [左下、右上] 边距的大小。

titlefontsizemultiplier: positive scalar, def. 1.1000

标题字体大小与刻度标签字体大小之间的比率。

titlefontweight: {"bold"} | "normal"

用于轴标题的基本字体的控制变体。

xlabel: graphics handle

x 标签文本对象的图形句柄。

xticklabel: string | cell array of strings, def. 6-by-1 cell

x 刻度线的标签。设置 xticklabel 还会强制将 xticklabelmode 属性设置为“manual”。

xticklabelmode: {"auto"} | "manual"

用于确定 xtick 标签是由 Octave 自动设置还是使用“xticklabel”属性手动设置的设置。

xticklabelrotation: def. 0

尚未为轴对象实现轴标签旋转。xticklabelrotation 未使用。

ylabel: graphics handle

y 标签文本对象的图形句柄。

yticklabel: string | cell array of strings, def. 6-by-1 cell

y 刻度线的标签。设置 yticklabel 还会强制将 yticklabelmode 属性设置为“manual”。

yticklabelmode: {"auto"} | "manual"

用于确定 ytick 标签是由 Octave 自动设置还是使用“yticklabel”属性手动设置的设置。

yticklabelrotation: def. 0

尚未为轴对象实现轴标签旋转。yticklabelrotation 未使用。

zlabel: graphics handle

z 标签文本对象的图形句柄。

zticklabel: string | cell array of strings, def. 6-by-1 cell

z 刻度线的标签。设置 zticklabel 还会强制将 zticklabelmode 属性设置为“manual”。

zticklabelmode: {"auto"} | "manual"

用于确定 ztick 标签是由 Octave 自动设置还是使用“zticklabel”属性手动设置的设置。

zticklabelrotation: def. 0

尚未为轴对象实现轴标签旋转。zticklabelRotation 未使用。

**15.3.3.4图例属性**

图例对象属性(见**图例**):

**类别：**

回调执行 |布局 |图例外观 |物体识别 |对象位置 |文本外观

**回调执行**

itemhitfcn: def. [](0x0)

回调函数，在单击图例项时执行。请参阅回调部分。

回调函数必须具有以下原型 fcn （hlegend， evnt） ，其中 hlegend 是图例对象句柄，evnt 是具有以下字段的结构：

Peer

单击项所关联的绘图对象的句柄。

Region

可以是“图标”或“标签”，具体取决于单击项目的哪个部分。

SelectionType

“normal”、“extend”、“open”或“alt”之一。请参见图“selectiontype”。

Source

图例对象的句柄。

EventName

名称为“ItemHit”。

**布局**

autoupdate: "off" | {"on"}

控制在对等轴上添加（或删除）对象时是否自动更新图例项的数量。例如：

## Create a single plot with its legend.

figure ();

plot (1:10);

legend ("Slope 1");

## Add another plot and specify its displayname so that

## the legend is correctly updated.

hold on;

plot ((1:10) \* 2, "displayname", "Slope 2");

## Stop automatic updates for further plots.

legend ("autoupdate", "off");

plot ((1:10) \* 3);

numcolumns: scalar integer, def. 1

控制图例项布局中使用的列数。例如：

figure ();

plot (rand (30));

legend ("numcolumns", 3);

设置 numcolumns 还会强制将 numcolumnsmode 属性设置为“manual”。

orientation: "horizontal" | {"vertical"}

控制图例项是垂直排列（按列排列）还是水平排列（按行排列）。

**图例外观**

box: "off" | {"on"}

控制图例是否有周围的框。

color: colorspec, def. [1 1 1]

图例背景的颜色。请参阅 colorspec。

edgecolor: colorspec, def. [0.1500 0.1500 0.1500]

控制图例轮廓的颜色。

**物体识别**

title: graphics handle

标题文本对象的图形句柄。

**对象位置**

location: "best" | "bestoutside" | "east" | "eastoutside" | "none" | "north" | {"northeast"} | "northeastoutside" | "northoutside" | "northwest" | "northwestoutside" | "south" | "southeast" | "southeastoutside" | "southoutside" | "southwest" | "southwestoutside" | "west" | "westoutside"

控制图例的位置。

position: four-element vector

指定图例的位置，不包括其标题。矢量的四个元素是左下角的坐标和图例的宽度和高度。更改此属性还会将“location”切换为“none”。

units: "centimeters" | "characters" | "inches" | {"normalized"} | "pixels" | "points"

用于解释“position”属性的单位。

**文本外观**

fontangle: "italic" | {"normal"}

控制字体是斜体还是普通。

fontname: string, def. "\*"

用于文本呈现的字体名称。设置此属性时，文本呈现引擎将在系统中搜索匹配的字体。如果未找到任何文本，则使用默认的无衬线字体（与默认的“\*”值相同）呈现文本。

编程说明：在不本机使用 FontConfig 的系统（除 Linux 之外的所有系统）上，字体缓存是在安装 Octave 时构建的。安装新字体后，您需要手动运行系统（“fc-cache -fv”）。

fontsize: scalar, def. 9

用于文本呈现的字体大小。请参见fontunits 属性。设置 fontsize 还会强制将 fontsizemode 属性设置为“manual”。

fontunits: "centimeters" | "inches" | "normalized" | "pixels" | {"points"}

用于解释“fontsize”属性的单位。

fontweight: "bold" | {"normal"}

控制用于文本呈现的基本字体的变体。

string: string | cell array of strings

图例项的标签列表。例如：

figure ();

plot (rand (20));

## Let legend choose names automatically

hl = legend ();

## Selectively change some names

str = get (hl, "string");

str(1:5:end) = "Garbage";

set (hl, "string", str);

textcolor: colorspec, def. [0 0 0]

控制图例项的文本字符串的颜色。

**15.3.3.5行属性**

线条对象的属性(见线条):

**类别：**

回调执行 |坐标数据 |创建/删除 |显示 |图例选项 |线条外观 |标记外观 |鼠标交互 |物体识别 |家长/子女

**回调执行**

busyaction: "cancel" | {"queue"}

定义 Octave 在无法中断另一个对象的执行回调时如何处理此对象的回调属性的执行。仅当当前正在执行的回调对象的 interruptible 属性设置为“off”时，这才有意义。中断回调对象的 busyaction 属性指示中断回调是排队（“queue”（默认））还是丢弃（“cancel”）。请参阅回调部分。

interruptible: "off" | {"on"}

指定此对象的回调函数是否会被其他回调中断。默认情况下，interruptible 为 “on”，使用 drawnow、figure、waitfor、getframe 或 pause 函数的回调最终会中断。请参阅回调部分。

**坐标数据**

xdata: vector, def. [0 1]

要绘制的 x 数据的矢量。

xdatasource: string, def. ""

当前基本工作区中要用作 x 数据的矢量的名称。

ydata: vector, def. [0 1]

要绘制的 y 数据的矢量。

ydatasource: string, def. ""

当前基本工作区中要用作 y 数据的矢量的名称。

zdata: vector, def. [](0x0)

当前基本工作区中要用作 y 数据的矢量的名称。

zdatasource: string, def. ""

当前基本工作区中要用作 z 数据的矢量的名称。

**创建/删除**

beingdeleted: {"off"} | "on"

指示函数已启动对象删除的属性。beingdeleted 设置为 true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

回调函数在创建行后立即执行。函数是通过使用根对象上的默认属性来设置的，例如，set （groot， “defaultlinecreatefcn”， 'disp （“line created！”）'）。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

回调函数在删除行之前立即执行。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

**显示**

clipping: "off" | {"on"}

如果剪裁为“打开”，则线将在其父轴限制内进行剪裁。

visible: "off" | {"on"}

如果 visible 为“off”，则不会在屏幕上呈现该行。

**图例选项**

displayname: string | cell array of strings, def. ""

与此行对应的图例条目的文本。

**线条外观**

color: colorspec, def. [0 0 0]

线条对象的颜色。请参阅 colorspec。

linejoin: "chamfer" | "miter" | {"round"}

控制线段交汇点的形状。此属性当前仅影响打印输出。

linestyle: {"-"} | "--" | "-." | ":" | "none"

请参阅线型。

linewidth: scalar, def. 0.5000

线对象的宽度（以磅为单位）。

**标记外观**

marker: "\*" | "+" | "." | "<" | ">" | "^" | "\_" | "d" | "diamond" | "h" | "hexagram" | {"none"} | "o" | "p" | "pentagram" | "s" | "square" | "v" | "x" | "|"

每个数据点的标记形状。请参阅标记样式。

markeredgecolor: {"auto"} | "none"

标记边缘的颜色。当设置为“自动”时，标记边缘的颜色与线条相同。如果设置为“none”，则不显示标记边缘。此属性也可以设置为任何颜色。请参阅 colorspec。

markerfacecolor: "auto" | {"none"}

标记面部的颜色。当设置为“自动”时，标记面的颜色与线条相同。如果设置为“无”，则不显示标记面。此属性也可以设置为任何颜色。请参阅 colorspec。

markersize: scalar, def. 6

以磅为单位的标记的大小。

**鼠标交互**

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形侦听器函数的信息，请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

当前与此行对象关联的 uicontextmenu 对象的图形句柄。

hittest: "off" | {"on"}

指定 line 是处理鼠标事件还是将其传递给对象的祖先。启用后，对象可以通过评估“buttondownfcn”、显示 uicontextmenu 并最终成为根“currentobject”来响应鼠标单击。仅当对象可以接受鼠标单击时，此属性才相关，鼠标单击由“pickableparts”属性确定。请参见pickableparts 属性。

pickableparts: "all" | "none" | {"visible"}

指定线路是否接受鼠标点击。默认情况下，拾取部件是“可见的”，只有行的可见部分或其子项才能对鼠标单击做出反应。当可拾取部件为“全部”时，可见和不可见部件（或子部件）都可能对鼠标单击做出反应。当 pickableparts 为“无”时，鼠标单击对象将被忽略并传输到该对象下方的任何对象。当对象配置为接受鼠标单击时，“hittest”属性将确定如何处理它们。请参阅 hittest 属性。

selected: {"off"} | "on"

属性指示是否选择此行。

selectionhighlight: "off" | {"on"}

如果 selectionhighlight 为“on”，则该行的选择状态将直观地突出显示。

**物体识别**

tag: string, def. ""

用于标记图形对象的用户定义字符串。

type (read-only): string

图形对象的类名。type 始终为“line”。

userdata: Any Octave data, def. [](0x0)

要与图形对象关联的用户定义数据。

**家长/子女**

children (read-only): vector of graphics handles, def. [](0x1)

尚未为线条对象实现线条的子对象。儿童未使用。

handlevisibility: "callback" | "off" | {"on"}

如果 handlevisibility 为“off”，则行的句柄在其父级的“children”属性中不可见。

parent: graphics handle

父图形对象的句柄。

**15.3.3.6文本属性**

文本对象的属性(参见文本):

**类别：**

回调执行 |创建/删除 |显示 |鼠标交互 |物体识别 |对象位置 |家长/子女 |文本外观 |文本框外观

**回调执行**

busyaction: "cancel" | {"queue"}

定义 Octave 在无法中断另一个对象的执行回调时如何处理此对象的回调属性的执行。仅当当前正在执行的回调对象的 interruptible 属性设置为“off”时，这才有意义。中断回调对象的 busyaction 属性指示中断回调是排队（“queue”（默认））还是丢弃（“cancel”）。请参阅回调部分。

interruptible: "off" | {"on"}

指定此对象的回调函数是否会被其他回调中断。默认情况下，interruptible 为 “on”，使用 drawnow、figure、waitfor、getframe 或 pause 函数的回调最终会中断。请参阅回调部分。

**创建/删除**

beingdeleted: {"off"} | "on"

指示函数已启动对象删除的属性。beingdeleted 设置为 true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

创建文本后立即执行回调函数。函数是通过使用根对象上的默认属性来设置的，例如，set （groot， “defaulttextcreatefcn”， 'disp （“text created！”）'）。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除文本之前立即执行回调函数。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

**显示**

clipping: "off" | {"on"}

如果剪裁为“打开”，则文本将在其父轴限制内进行剪裁。

visible: "off" | {"on"}

如果 visible 为“off”，则文本不会呈现在屏幕上。

**鼠标交互**

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形侦听器函数的信息，请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

当前与此文本对象关联的 uicontextmenu 对象的图形句柄。

hittest: "off" | {"on"}

指定文本是处理鼠标事件还是将其传递给对象的祖先。启用后，对象可以通过评估“buttondownfcn”、显示 uicontextmenu 并最终成为根“currentobject”来响应鼠标单击。仅当对象可以接受鼠标单击时，此属性才相关，鼠标单击由“pickableparts”属性确定。请参见pickableparts 属性。

pickableparts: "all" | "none" | {"visible"}

指定文本是否接受鼠标单击。默认情况下，拾取部分是“可见的”，只有文本的可见部分或其子部分才能对鼠标单击做出反应。当可拾取部件为“全部”时，可见和不可见部件（或子部件）都可能对鼠标单击做出反应。当 pickableparts 为“无”时，鼠标单击对象将被忽略并传输到该对象下方的任何对象。当对象配置为接受鼠标单击时，“hittest”属性将确定如何处理它们。请参阅 hittest 属性。

selected: {"off"} | "on"

属性指示是否选中此文本。

selectionhighlight: "off" | {"on"}

如果 selectionhighlight 为“on”，则文本的选择状态将直观地突出显示。

**物体识别**

tag: string, def. ""

用于标记图形对象的用户定义字符串。

type (read-only): string

图形对象的类名。type 始终为“text”。

userdata: Any Octave data, def. [](0x0)

要与图形对象关联的用户定义数据。

**物体位置**

extent (read-only): four-element vector

矢量 [x0 y0 width height] 指示文本字符串的大小和位置。

horizontalalignment: "center" | {"left"} | "right"

指定“position”属性设置的点相对于文本的水平位置。

position: three-element vector, def. [0 0 0]

矢量 [X0 Y0 Z0]，其中 X0、Y0 和 Z0 表示由垂直对齐和水平对齐定义的文本锚点的位置。

rotation: scalar, def. 0

显示文本的旋转角度，以度为单位。

units: "centimeters" | {"data"} | "inches" | "normalized" | "pixels" | "points"

设置应用于“位置”和“范围”属性的测量单位或方法。默认选项“data”使用与图中绘制的数据相同的单位和限制。“规范化”选项将无单位的 0 到 1 刻度应用于沿显示数据的每个轴的限制。

verticalalignment: "baseline" | "bottom" | "cap" | {"middle"} | "top"

指定“position”属性设置的点相对于文本的垂直位置。请注意，“top”和“bottom”与文本框的边缘对齐，而“cap”和“baseline”是指文本本身的边缘。

**家长/子女**

children (read-only): vector of graphics handles, def. [](0x1)

文本对象没有子对象。儿童未使用。

handlevisibility: "callback" | "off" | {"on"}

如果 handlevisibility 为“off”，则文本的句柄在其父级的“children”属性中不可见。

parent: graphics handle

父图形对象的句柄。

**文本外观**

color: colorspec, def. [0 0 0]

文本的颜色。请参阅 colorspec。

editing: {"off"} | "on"

尚未为文本对象实现交互式文本编辑。编辑未使用。

fontangle: "italic" | {"normal"}

控制字体是斜体还是普通。

fontname: string, def. "\*"

用于文本呈现的字体名称。设置此属性时，文本呈现引擎将在系统中搜索匹配的字体。如果未找到任何文本，则使用默认的无衬线字体（与默认的“\*”值相同）呈现文本。

编程说明：在不本机使用 FontConfig 的系统（除 Linux 之外的所有系统）上，字体缓存是在安装 Octave 时构建的。安装新字体后，您需要手动运行系统（“fc-cache -fv”）。

fontsize: scalar, def. 10

用于文本呈现的字体大小。请参见fontunits 属性。

fontsmoothing: "off" | {"on"}

控制渲染文本时是否使用抗锯齿。

fontunits: "centimeters" | "inches" | "normalized" | "pixels" | {"points"}

用于解释“fontsize”属性的单位。

fontweight: "bold" | {"normal"}

控制用于文本呈现的基本字体的变体。

interpreter: "latex" | "none" | {"tex"}

控制“string”属性的解释方式。请参见“interpreter”属性的使用。

string: string, def. ""

文本对象字符串内容。

**文本框外观**

backgroundcolor: colorspec, def. "none"

背景区域的颜色。请参阅 colorspec。

edgecolor: colorspec, def. "none"

背景区域轮廓的颜色。请参阅 colorspec。

linestyle: {"-"} | "--" | "-." | ":" | "none"

文本框轮廓的样式。请参阅线型。

linewidth: scalar, def. 0.5000

文本框轮廓的宽度。

margin: scalar, def. 3

背景区域和文本边框之间的边距。该值当前被解释为像素，而不考虑“fontunits”属性。

**15.3.3.7镜像属性**

图像对象的属性(见图):

类别：

回调执行 |创建/删除 |显示器 |图像数据 |鼠标交互 |物体识别 |家长/子女

回调执行

busyaction: "cancel" | {"queue"}

定义 Octave 在无法中断另一个对象的执行回调时如何处理此对象的回调属性的执行。仅当当前正在执行的回调对象的 interruptible 属性设置为“off”时，这才有意义。中断回调对象的 busyaction 属性指示中断回调是排队（“queue”（默认））还是丢弃（“cancel”）。请参阅回调部分。

interruptible: "off" | {"on"}

指定此对象的回调函数是否会被其他回调中断。默认情况下，interruptible 为 “on”，使用 drawnow、figure、waitfor、getframe 或 pause 函数的回调最终会中断。请参阅回调部分。

创建/删除

beingdeleted: {"off"} | "on"

指示函数已启动对象删除的属性。beingdeleted 设置为 true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

创建映像后立即执行回调函数。函数是通过使用根对象上的默认属性来设置的，例如，set （groot， “defaultimagecreatefcn”， 'disp （“image created！”）'）。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除图像之前立即执行的回调函数。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

显示

clipping: "off" | {"on"}

如果剪裁为“打开”，则图像将在其父轴限制内进行剪裁。

visible: "off" | {"on"}

如果可见为“关闭”，则图像不会呈现在屏幕上。

图像数据

alphadata: scalar | matrix, def. 1

尚未为图像对象实现透明度。Alphadata 未使用。

alphadatamapping: "direct" | {"none"} | "scaled"

尚未为图像对象实现透明度。AlphaDataMapping 未使用。

cdata: array, def. 64-by-64 double

图像对象的颜色数据。数据存储为 2D 矩阵，其中每个元素的值根据当前颜色图确定该像素的颜色，或者存储为 3D 数组，其中第三个维度包含每个像素的单独红色、蓝色和绿色分量。对于 RGB 数组，映射到最小和最大颜色值的值取决于“cdata”的类。对于该整数类，浮点数和逻辑值的范围从 0 到 1，而整数值的范围从 intmin 到 intmax。

cdatamapping: {"direct"} | "scaled"

设置将数据从“cdata”属性映射到当前颜色图的方法。“直接”映射使用“cdata”值作为当前颜色图的索引来选择颜色。“缩放”映射将“cdata”值缩放到“clim”轴属性中指定的范围。

xdata: two-element vector, def. [1 64]

双元素矢量 [xfirst xlast] 指定图像第一列和最后一列中心的 x 坐标。

将 xdata 设置为空矩阵 （[]） 将恢复默认值 [1 columns（image）]。

ydata: two-element vector, def. [1 64]

双元素矢量 [yfirst ylast] 指定图像第一行和最后一行中心的 y 坐标。

将 ydata 设置为空矩阵 （[]） 将恢复默认值 [1 rows（image）]。

鼠标交互

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形侦听器函数的信息，请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

当前与此图像对象关联的 uicontextmenu 对象的图形句柄。

hittest: "off" | {"on"}

指定图像是处理鼠标事件还是将其传递给对象的祖先。启用后，对象可以通过评估“buttondownfcn”、显示 uicontextmenu 并最终成为根“currentobject”来响应鼠标单击。仅当对象可以接受鼠标单击时，此属性才相关，鼠标单击由“pickableparts”属性确定。请参见pickableparts 属性。

pickableparts: "all" | "none" | {"visible"}

指定图像是否接受鼠标单击。默认情况下，拾取部分为“可见”部分，只有图像的可见部分或其子部分才能对鼠标单击做出反应。当可拾取部件为“全部”时，可见和不可见部件（或子部件）都可能对鼠标单击做出反应。当 pickableparts 为“无”时，鼠标单击对象将被忽略并传输到该对象下方的任何对象。当对象配置为接受鼠标单击时，“hittest”属性将确定如何处理它们。请参阅 hittest 属性。

selected: {"off"} | "on"

属性指示是否选中此图像。

selectionhighlight: "off" | {"on"}

如果 selectionhighlight 为“on”，则图像的选择状态在视觉上突出显示。

物体识别

tag: string, def. ""

用于标记图形对象的用户定义字符串。

type (read-only): string

图形对象的类名。type 始终为“image”。

userdata: Any Octave data, def. [](0x0)

要与图形对象关联的用户定义数据。

家长/子女

children (read-only): vector of graphics handles, def. [](0x1)

尚未为图像对象实现 Images 的子对象。儿童未使用。

handlevisibility: "callback" | "off" | {"on"}

如果 handlevisibility 为“off”，则图像的句柄在其父级的“children”属性中不可见。

parent: graphics handle

父图形对象的句柄。

**15.3.3.8补丁属性**

补丁对象的属性(参见patch):

类别：

回调执行 |颜色和透明度 |坐标数据 |创建/删除 |显示器 |图例选项 |照明 |标记外观 |鼠标交互 |物体识别 |外形外观 |家长/子女

回调执行

busyaction: "cancel" | {"queue"}

定义 Octave 在无法中断另一个对象的执行回调时如何处理此对象的回调属性的执行。仅当当前正在执行的回调对象的 interruptible 属性设置为“off”时，这才有意义。中断回调对象的 busyaction 属性指示中断回调是排队（“queue”（默认））还是丢弃（“cancel”）。请参阅回调部分。

interruptible: "off" | {"on"}

指定此对象的回调函数是否会被其他回调中断。默认情况下，interruptible 为 “on”，使用 drawnow、figure、waitfor、getframe 或 pause 函数的回调最终会中断。请参阅回调部分。

Color and Transparency

alphadatamapping: "direct" | "none" | {"scaled"}

尚未为修补进程对象实现透明度。AlphaDataMapping 未使用。

cdata: scalar | matrix | array, def. [](0x0)

定义面片对象颜色相对于其 x/y/z 坐标数据的数据。可以使用当前颜色图中的索引或RGB三元组来定义色块颜色，其中RGB颜色沿第三维定义。这些颜色可以分别为整个面片对象、单个面或单个顶点定义，并由“cdata”的形状确定，如下所示：

如果“cdata”是当前颜色图的标量索引或 1×1×3 RGB 三元组，则它定义所有面和边的颜色。

如果 patch 对象有 N 个面，并且“cdata”是 1×N 的颜色图索引矢量或 1×N×3 RGB 数组，则它定义每个面的颜色。

如果 patch 对象有 N 个面，每个面有 M 个顶点，并且 cdata 是颜色图索引的 M×N 矩阵或 M×N×3 RGB 数组，则它定义每个顶点的颜色。（“cdata”的形状应与“xdata”、“ydata”和“zdata”的形状匹配。

cdatamapping: "direct" | {"scaled"}

设置将数据从“cdata”或“cdata”属性映射到当前颜色图的方法。“直接”映射使用“cdata”或“facevertexcdata”值作为当前颜色图的索引来选择颜色。“缩放”映射将“cdata”或“facevertexcdata”值缩放到“clim”坐标属性中指定的范围。

facealpha: scalar | "flat" | "interp", def. 1

面片对象面的透明度级别。目前仅支持双精度值，其中值为 0 表示完全透明，值为 1 表示没有透明度的实体面。将属性设置为“flat”或“interp”会导致无法呈现人脸。此外，人脸不会从后到前进行排序，这可能会导致在渲染分层透明人脸时出现意外结果。

facecolor: {colorspec} | "none" | "flat" | "interp", def. [0 0 0]

面片对象的面的颜色，指定为有效颜色规范或“无”、“平面”或“interp”之一。“flat” 和 “interp” 将使用存储在 “cdata” 或 “facevertexcdata” 属性中的颜色值数据为每个面设置单一颜色，或在面的顶点上插值颜色。请参阅 colorspec。

facelighting: {"flat"} | "gouraud" | "none" | "phong"

当设置为“none”以外的值时，将使用光影效果绘制对象的面。支持的值为“none”（无照明效果）、“flat”（多面外观）和“gouraud”（顶点之间照明效果的线性插值）。“phong”已弃用，其效果与“gouraud”相同。

facevertexalphadata: def. [](0x0)

尚未为面片对象实现面顶透明度控制。FaceVertexalphaData 未使用。

facevertexcdata: scalar | matrix, def. [](0x0)

定义面片对象颜色相对于其面顶点数据的数据。可以使用当前颜色图中的索引或RGB三元组来定义斑块颜色，其中RGB颜色在“facevertexcdata”行中定义。这些颜色可以为整个面片对象、单个面或单个顶点单独定义，并由“facevertexcdata”的形状确定，如下所示：

如果 facevertexcdata 是当前颜色图的标量索引或 1×3 RGB 三元组，则它定义所有面和边的颜色。

如果 patch 对象具有 N 个面，并且 facevertexcdata 是索引的 N×1 列矢量或 N×3 RGB 矩阵，则它定义 N 个面中每个面的颜色。

如果 patch 对象具有 M 个顶点，并且 facevertexcdata 是索引的 M×1 列矢量或 M×3 RGB 矩阵，则它定义每个顶点的颜色。

坐标数据

faces: vector | matrix, def. [1 2 3]

面片面连通性列表存储为 M x N 矩阵，每个 M 面由一行最多 N 个顶点定义，每个元素都包含存储在顶点属性中的顶点的行索引。顶点少于 N 的面使用 NaN 值填充空行元素。

vertices: vector | matrix, def. 3-by-2 double

patch 顶点列表存储为 N x 3 矩阵，每行包含矢量的 x、y 和 z 坐标，并与 faces 属性一起用于定义 patch 结构。

xdata: vector | matrix, def. [0; 1; 0]

修补顶点 X 坐标。

ydata: vector | matrix, def. [1; 1; 0]

修补顶点 Y 坐标。

zdata: vector | matrix, def. [](0x0)

修补顶点 z 坐标。

创建/删除

beingdeleted: {"off"} | "on"

指示函数已启动对象删除的属性。beingdeleted 设置为 true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

创建补丁后立即执行回调函数。函数是通过使用 root 对象上的默认属性来设置的，例如 set （groot， “defaultpatchcreatefcn”， 'disp （“patch created！”）'）。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除补丁之前立即执行的回调函数。

有关如何编写图形侦听器函数的信息，请参阅回调部分。

显示

clipping: "off" | {"on"}

如果剪裁为“打开”，则裁剪贴片将在其父轴限制内进行剪裁。

visible: "off" | {"on"}

如果 visible 为“关闭”，则修补进程不会呈现在屏幕上。

图例选项

displayname: def. ""

与此修补进程对应的图例条目的文本。

照明

ambientstrength: scalar, def. 0.3000

环境光的强度。值介于 0.0 和 1.0 之间。

backfacelighting: "lit" | {"reverselit"} | "unlit"

“lit”：法线按原样用于照明。“reverselit”：法线始终朝向视点。“unlit”：法线指向远离视点的面不亮。

diffusestrength: scalar, def. 0.6000

漫反射的强度。值介于0.0(无漫反射)和1.0(完全漫反射)之间。

facenormals: def. [](0x0)

如果边缘光照或面光照属性设置为“平坦”，则面法线用于照亮边缘或面。设置facenormals也强制facnormalmode属性设置为“manual”。

facenormalsmode: {"auto"} | "manual"

如果将此属性设置为“auto”，则如果将边缘照明或面照明属性设置为“flat”，并且在同一轴上至少存在一个可见的光照对象，则会自动计算面大小。

specularcolorreflectance: scalar, def. 1

镜面颜色的反射率。值介于0.0(底面颜色)和1.0(光源颜色)之间。

specularexponent: scalar, def. 10

镜面反射的指数。该值越低，反射越扩散。

specularstrength: scalar, def. 0.9000

镜面反射强度。值介于0.0(无镜面反射)和1.0(全镜面反射)之间。

vertexnormals: def. [](0x0)

顶点法线用于照亮边缘或面，如果边缘照明或面照明属性设置为“gouraud”。设置vertexnormals也强制将vertexnormalmode属性设置为“manual”。

vertexnormalsmode: {"auto"} | "manual"

如果此属性设置为“auto”，则如果边缘照明或面照明属性设置为“gouraud”，并且在同一轴上至少存在一个可见的光照对象，则自动计算顶点法线。

标志的外观

marker: "\*" | "+" | "." | "<" | ">" | "^" | "\_" | "d" | "diamond" | "h" | "hexagram" | {"none"} | "o" | "p" | "pentagram" | "s" | "square" | "v" | "x" | "|"

参见行标记属性。

markeredgecolor: {"auto"} | "flat" | "none"

参见line markeredgcolor属性。

markerfacecolor: "auto" | "flat" | {"none"}

参见line markerfacecolor属性。

markersize: scalar, def. 6

参见line markersize属性。

鼠标交互

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形监听器函数的信息，请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

当前与此补丁对象关联的uicontextmenu对象的图形句柄。

hittest: "off" | {"on"}

指定patch是处理鼠标事件还是将其传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定补丁是否接受鼠标点击。默认情况下，pickableparts是“可见的”，只有补丁的可见部分或它的子部分可能对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性表示是否选择该补丁。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为“on”，则patch的选择状态将被高亮显示。

物体识别

tag: string, def. ""

一个用户定义的字符串，用于标记图形对象。

type (read-only): string

图形对象的类名。类型总是“patch”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

改善外形

edgealpha: scalar | matrix, def. 1

透明度尚未对补丁对象实现。Edgealpha未使用。

edgecolor: colorspec | "none" | "flat" | "interp", def. [0 0 0]

patch对象边缘的颜色，指定为有效的颜色规格或“none”、“flat”或“interp”中的一种。“flat”和“interp”将为每条边设置一个单一的颜色，或者使用存储在“cdata”中的颜色值数据在边缘的顶点之间插入一个颜色。看到colorspec。

edgelighting: "flat" | "gouraud" | {"none"} | "phong"

当设置为“none”以外的值时，对象的边缘将使用光影效果绘制。支持的值是“none”(没有照明效果)，“flat”(多面外观)和“gouraud”(顶点之间的照明效果的线性插值)。“phong”已弃用，其效果与“gouraud”相同。

linestyle: {"-"} | "--" | "-." | ":" | "none"

参见线条样式。

linewidth: scalar, def. 0.5000

参见line linewidth属性。

家长/子女

children (read-only): vector of graphics handles, def. [](0x1)

Patch对象的子对象尚未在Patch对象中实现。儿童是闲置的。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility是"off"，补丁的句柄在其父的"children"属性中不可见。

parent: graphics handle

父图形对象的句柄。

15.3.3.9散射属性

散射对象的属性(参见scatter):

类别:

回调执行|颜色数据|坐标数据|创建/删除|显示|图例选项|标记外观|鼠标交互|对象识别|父/子

回调执行

busyaction: "cancel" | {"queue"}

定义Octave在无法中断另一个对象的回调执行时如何处理该对象的回调属性的执行。只有当当前执行的回调对象的可中断属性设置为“off”时，这才有意义。中断回调对象的busyaction属性指示中断回调是排队(“queue”(默认))还是丢弃(“cancel”)。参见回调一节。

interruptible: "off" | {"on"}

指定此对象的回调函数是否可能被其他回调中断。默认情况下，可中断状态为“on”，使用drawnow、figure、waitfor、getframe或pause函数的回调函数最终会被中断。参见回调一节。

颜色数据

cdata: scalar | matrix, def. [0 0.4470 0.7410]

定义散射对象颜色的数据。

如果cdata是当前颜色映射或RGB三元组的标量索引，则它定义所有散点标记的颜色。

如果cdata是N × 1的索引向量或N × 3 (RGB)矩阵，则它定义了N个散点标记中每个散点标记的颜色。

cdatamode: {"auto"} | "manual"

如果cdatmode为"auto"，则cdata设置为与seriesindex对应的祖先轴的颜色顺序中的颜色。

cdatasource: string, def. ""

工作空间变量的名称，该变量包含将用于“cdata”属性的数据。使用refreshdata函数将数据传输到“cdata”中。

seriesindex: def. 1

在同一轴上的每个散射对象被分配一个递增的整数。这对应于在cdatmode设置为“auto”时使用的祖先轴颜色顺序的索引。

坐标数据

latitudedata: def. [](0x0)

地理坐标散点绘图尚未对散点对象实现。Latitudedata未使用。

latitudedatasource: def. ""

地理坐标散点绘图尚未对散点对象实现。Latitudedatasource未使用。

longitudedata: def. [](0x0)

地理坐标散点绘图尚未对散点对象实现。经度数据未被使用。

longitudedatasource: def. ""

地理坐标散点绘图尚未对散点对象实现。经度数据源未使用。

rdata: def. [](0x0)

对于散点对象，尚未实现用于散点绘图的极坐标。Rdata未使用。

rdatasource: def. ""

对于散点对象，尚未实现用于散点绘图的极坐标。Rdatasource未使用。

thetadata: def. [](0x0)

对于散点对象，尚未实现用于散点绘图的极坐标。数据未使用。

thetadatasource: def. ""

对于散点对象，尚未实现用于散点绘图的极坐标。数据源未使用。

xdata: vector, def. [](0x0)

带有散射物体x坐标的矢量。

xdatasource: string, def. ""

工作区变量的名称，该变量包含将用于“xdata”属性的数据。使用refreshdata函数将数据传输到“xdata”中。

ydata: vector, def. [](0x0)

矢量与散射对象的y坐标。

ydatasource: string, def. ""

工作空间变量的名称，该变量包含将用于“ydata”属性的数据。使用refreshdata函数将数据传输到“ydata”中。

zdata: [] | vector, def. [](0x0)

对于3D数据，矢量与散射对象的y坐标。

zdatasource: string, def. ""

工作区变量的名称，该变量包含将用于“zdata”属性的数据。使用refreshdata函数将数据传输到“zdata”中。

创建/删除

beingdeleted: {"off"} | "on"

属性，指示函数已开始删除对象。Beingdeleted被设置为true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

创建散射后立即执行的回调函数。函数通过使用根对象上的默认属性来设置，例如set (root， "defaultscattercreatefcn"， 'disp ("scatter created!")')。

有关如何编写图形监听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除scatter之前立即执行的回调函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

显示

clipping: "off" | {"on"}

如果裁剪处于“开启”状态，则散点在其父轴限制内被裁剪。

visible: "off" | {"on"}

如果“可见”为“关闭”，则散射不会在屏幕上渲染。

图例选项

annotation: def. [](0x0)

从分散对象内部进行图例外观切换尚未对分散对象实现。注释未使用。

displayname: def. ""

与此分散对象对应的图例条目的文本。

标志的外观

linewidth: scalar, def. 0.5000

标记线边缘的线宽。

marker: "\*" | "+" | "." | "<" | ">" | "^" | "\_" | "d" | "diamond" | "h" | "hexagram" | "none" | {"o"} | "p" | "pentagram" | "s" | "square" | "v" | "x" | "|"

参见行标记属性。

markeredgealpha: scalar, def. 1

标记面透明度级别，0表示完全透明，1表示不透明的实体面。请注意，标记不是从后到前排序的，这可能会导致渲染分层透明标记或与其他透明对象组合时出现意想不到的结果。

markeredgecolor: {"none"} | {"flat"} | colorspec

标记笔边缘的颜色。“none”表示边缘是透明的，“flat”表示使用cdata中的值。参见line markeredgcolor属性。

markerfacealpha: scalar, def. 1

标记面透明度级别，0表示完全透明，1表示不透明的实体面。请注意，标记不是从后到前排序的，这可能会导致渲染分层透明标记或与其他透明对象组合时出现意想不到的结果。

markerfacecolor: {{"none"}} | "flat" | "auto" | colorspec

记号笔表面的颜色。“none”表示面是透明的，“flat”表示使用cdata中的值，“auto”使用祖先轴的颜色属性。参见line markerfacecolor属性。

sizedata: [] | scalar | vector, def. [](0x0)

标记区域的大小。标量值适用于所有标记。如果cdata是一个N × 1的向量，它定义了N个散点标记中的每一个的颜色。

sizedatasource: def. ""

来自工作空间变量的数据尚未为分散对象实现。Sizedatasource未使用。

鼠标交互

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形监听器函数的信息，请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

当前与此散射对象关联的uicontextmenu对象的图形句柄。

datatiptemplate: def. [](0x0)

数据提示对象还没有为分散对象实现。Datatiptemplate未使用。

hittest: "off" | {"on"}

指定scatter是处理鼠标事件还是将其传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定scatter是否接受鼠标点击。默认情况下，pickableparts是“可见的”，并且只有散布的可见部分或它的子部分可能对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性指示是否选择此散射。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为“on”，则散点的选择状态将在视觉上突出显示。

物体识别

tag: string, def. ""

一个用户定义的字符串来标记图形对象。

type (read-only): string

图形对象的类名。类型总是“分散”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

亲生子女

children (read-only): vector of graphics handles, def. [](0x1)

分散对象的子对象尚未实现分散对象。孩子们没有用。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility为“off”，则散射的句柄在其父的“children”属性中不可见。

parent: graphics handle

父图形对象的句柄。

**15.3.3.10表面性质**

表面物体的性质(见表面):

**类别:**

回调执行|颜色和透明度|坐标数据|创建/删除|显示|人脸外观|图例选项|照明|标记外观|鼠标交互|对象识别|轮廓外观|父/子

回调执行

busyaction: "cancel" | {"queue"}

定义Octave在无法中断另一个对象的回调执行时如何处理该对象的回调属性的执行。只有当当前执行的回调对象的可中断属性设置为“off”时，这才有意义。中断回调对象的busyaction属性指示中断回调是排队(“queue”(默认))还是丢弃(“cancel”)。参见回调一节。

interruptible: "off" | {"on"}

指定此对象的回调函数是否可能被其他回调中断。默认情况下，可中断状态为“on”，使用drawnow、figure、waitfor、getframe或pause函数的回调函数最终会被中断。参见回调一节。

颜色和透明度

alphadata: scalar | matrix, def. 1

表面对象的透明度尚未实现。Alphadata未被使用。

alphadatamapping: "direct" | "none" | {"scaled"}

表面对象的透明度尚未实现。Alphadatamapping未被使用。

cdata: array, def. 3-by-3 double

表面顶点的颜色数据值。数据要么存储为与“zdata”大小相同的二维矩阵，其中每个元素的值根据当前颜色图确定顶点的颜色，要么存储为三维数组，其中第三维包含每个顶点的单独红色、蓝色和绿色组件。

cdatamapping: "direct" | {"scaled"}

设置将数据从“cdata”属性映射到当前颜色映射的方法。“直接”映射使用“cdata”值作为当前颜色映射的索引来选择颜色。“Scale”映射将“cdata”值缩放到“clim”坐标轴属性中指定的范围。

cdatasource: string, def. ""

工作空间变量的名称，该变量包含将用于“cdata”属性的数据。使用refreshdata函数将数据传输到“cdata”中。

坐标数据

xdata: matrix, def. [1 2 3]

x坐标的数据。

xdatasource: string, def. ""

工作区变量的名称，该变量包含将用于“xdata”属性的数据。使用refreshdata函数将数据传输到“xdata”中。

ydata: matrix, def. [1; 2; 3]

y坐标的数据。

ydatasource: string, def. ""

工作空间变量的名称，该变量包含将用于“ydata”属性的数据。使用refreshdata函数将数据传输到“ydata”中。

zdata: matrix, def. 3-by-3 double

z坐标的数据。

zdatasource: string, def. ""

工作区变量的名称，该变量包含将用于“zdata”属性的数据。使用refreshdata函数将数据传输到“zdata”中。

创建/删除

beingdeleted: {"off"} | "on"

属性，指示函数已开始删除对象。Beingdeleted被设置为true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

在创建表面后立即执行的回调函数。函数通过在根对象上使用默认属性来设置，例如set (root， "defaultsurfacecreatefcn"， 'disp ("surface created!")')。

有关如何编写图形监听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除表面之前立即执行的回调函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

显示

clipping: "off" | {"on"}

如果裁剪处于“开启”状态，则表面在其父轴限制内被裁剪。

visible: "off" | {"on"}

如果“可见”为“关闭”，则表面不会在屏幕上渲染。

面孔出现

facealpha: scalar | "flat" | "interp" | "texturemap", def. 1

表面物体表面的透明度级别。目前只支持双值，其中值0表示完全透明，值1表示不透明的实体面。将属性设置为“flat”，“interp”或“texturemap”会导致不渲染面。此外，这些面没有从后到前排序，这可能会导致渲染分层透明面时意想不到的结果。

facecolor: colorspec | "none" | {"flat"} | "interp"

表面对象的面颜色，指定为有效的颜色规格或“无”、“平坦”或“间隔”之一。“flat”和“interp”将为每个面设置一个单一的颜色，或者使用存储在“cdata”中的颜色值数据在面的顶点上插入一个颜色。看到colorspec。

facelighting: {"flat"} | "gouraud" | "none" | "phong"

当设置为“none”以外的值时，对象的面将使用光影效果绘制。支持的值是“none”(没有照明效果)，“flat”(多面外观)和“gouraud”(顶点之间的照明效果的线性插值)。“phong”已弃用，其效果与“gouraud”相同。

facenormals: def. [](0x0)

如果边缘光照或面光照属性设置为“平坦”，则面法线用于照亮边缘或面。设置facenormals也强制facnormalmode属性设置为“manual”。

facenormalsmode: {"auto"} | "manual"

如果将此属性设置为“auto”，则如果将边缘照明或面照明属性设置为“flat”，并且在同一轴上至少存在一个可见的光照对象，则会自动计算面大小。

图例选项

displayname: def. ""

与此图例表面对应的图例条目的文本。

照明

ambientstrength: scalar, def. 0.3000

环境光的强度。值介于0.0和1.0之间。

backfacelighting: "lit" | {"reverselit"} | "unlit"

"lit":法线与照明相同。“反转”:法线总是朝向视点。“未点亮”:法线指向远离视点的面是未点亮的。

diffusestrength: scalar, def. 0.6000

漫反射的强度。值介于0.0(无漫反射)和1.0(完全漫反射)之间。

specularcolorreflectance: scalar, def. 1

镜面颜色的反射率。值介于0.0(底面颜色)和1.0(光源颜色)之间。

specularexponent: scalar, def. 10

镜面反射的指数。该值越低，反射越扩散。

specularstrength: scalar, def. 0.9000

镜面反射强度。值介于0.0(无镜面反射)和1.0(全镜面反射)之间。

vertexnormals: def. [](0x0)

顶点法线用于照亮边缘或面，如果边缘照明或面照明属性设置为“gouraud”。设置vertexnormals也强制将vertexnormalmode属性设置为“manual”。

vertexnormalsmode: {"auto"} | "manual"

如果此属性设置为“auto”，则如果边缘照明或面照明属性设置为“gouraud”，并且在同一轴上至少存在一个可见的光照对象，则自动计算顶点法线。

标志的外观

marker: "\*" | "+" | "." | "<" | ">" | "^" | "\_" | "d" | "diamond" | "h" | "hexagram" | {"none"} | "o" | "p" | "pentagram" | "s" | "square" | "v" | "x" | "|"

参见标记样式。

markeredgecolor: {"auto"} | "flat" | "none"

参见line markeredgcolor属性。

markerfacecolor: "auto" | "flat" | {"none"}

参见line markerfacecolor属性。

markersize: scalar, def. 6

参见line markersize属性。

鼠标交互

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形监听器函数的信息，请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

当前与此表面对象关联的uicontextmenu对象的图形句柄。

hittest: "off" | {"on"}

指定surface是处理鼠标事件还是将其传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定surface是否接受鼠标点击。默认情况下，pickableparts是“可见的”，只有表面的可见部分或它的子部分可能对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性指示是否选择此表面。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为“on”，则表面的选择状态在视觉上高亮显示。

物体识别

tag: string, def. ""

一个用户定义的字符串，用于标记图形对象。

type (read-only): string

图形对象的类名。类型永远是“表面”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

改善外形

edgealpha: scalar, def. 1

表面对象的透明度尚未实现。Edgealpha未使用。

edgecolor: colorspec | "none" | "flat" | "interp", def. [0 0 0]

表面对象边缘的颜色，指定为有效的颜色规格或“无”、“平坦”或“间隔”中的一种。“flat”和“interp”将为每个边缘设置单一颜色，或者使用存储在“cdata”中的颜色值数据在两个相邻顶点之间插入颜色。看到colorspec。

edgelighting: "flat" | "gouraud" | {"none"} | "phong"

当设置为“none”以外的值时，对象的边缘将使用光影效果绘制。支持的值是“none”(没有照明效果)，“flat”(多面外观)和“gouraud”(顶点之间的照明效果的线性插值)。“phong”已弃用，其效果与“gouraud”相同。

linestyle: {"-"} | "--" | "-." | ":" | "none"

参见线条样式。

linewidth: scalar, def. 0.5000

参见line linewidth属性。

meshstyle: {"both"} | "column" | "row"

指定是否显示与曲面数据的行、列或两者相关联的边。

父母/孩子

children (read-only): vector of graphics handles, def. [](0x1)

表面的子对象还没有实现。儿童是闲置的。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility为"off"，则表面的句柄在其父的"children"属性中不可见。

parent: graphics handle

父图形对象的句柄。

**15.3.3.11 Light属性**

光对象的属性(见光):

**类别:**

回调执行|创建/删除|显示|照明|鼠标交互|对象识别|父/子

**回调执行**

busyaction: "cancel" | {"queue"}

定义Octave在无法中断另一个对象的回调执行时如何处理该对象的回调属性的执行。只有当当前执行的回调对象的可中断属性设置为“off”时，这才有意义。中断回调对象的busyaction属性指示中断回调是排队(“queue”(默认))还是丢弃(“cancel”)。参见回调一节。

interruptible: "off" | {"on"}

指定此对象的回调函数是否可能被其他回调中断。默认情况下，可中断状态为“on”，使用drawnow、figure、waitfor、getframe或pause函数的回调函数最终会被中断。参见回调一节。

创建/删除

beingdeleted: {"off"} | "on"

属性，指示函数已开始删除对象。Beingdeleted被设置为true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

创建光后立即执行的回调函数。函数通过在根对象上使用默认属性来设置，例如set (root， "defaultlightcreatefcn"， 'disp ("light created!")')。

有关如何编写图形监听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除light之前立即执行的回调函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

显示

clipping: "off" | {"on"}

如果裁剪处于“开启”状态，则光线在其父轴范围内被裁剪。

visible: "off" | {"on"}

如果“可见”为“关闭”，则不会在屏幕上渲染光线。

照明

color: colorspec, def. [1 1 1]

光源的颜色。看到colorspec。

position: def. [1 0 1]

光源位置。

style: {"infinite"} | "local"

这个字符串定义了光是从无限远的光源(“无限”)还是从局部点源(“本地”)发出的。

鼠标交互

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形监听器函数的信息，请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

当前与此light对象关联的uicontextmenu对象的图形句柄。

hittest: "off" | {"on"}

指定light是处理鼠标事件还是将其传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定light是否接受鼠标点击。默认情况下，pickableparts是“可见的”，只有灯光的可见部分或它的子部分可能对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性指示是否选择此灯。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为“on”，则灯光的选择状态在视觉上高亮显示。

物体识别

tag: string, def. ""

一个用户定义的字符串，用于标记图形对象。

type (read-only): string

图形对象的类名。类型总是“轻”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

父/子

children (read-only): vector of graphics handles, def. [](0x1)

光对象没有子对象。儿童是闲置的。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility为off，则该灯的句柄在其父的children属性中不可见。

parent: graphics handle

父图形对象的句柄。

**15.3.3.12 ui菜单属性**

**uimenu对象的属性(参见uimenu):**

**类别:**

外观|回调执行|创建/删除|显示|键盘交互|菜单选项|鼠标交互|对象识别|对象位置|父/子

外观

foregroundcolor: colorspec, def. [0 0 0]

此菜单项的文本颜色值。

separator: {"off"} | "on"

指示是否在当前菜单位置上方绘制分隔线的状态。

回调执行

busyaction: "cancel" | {"queue"}

定义Octave在无法中断另一个对象的回调执行时如何处理该对象的回调属性的执行。只有当当前执行的回调对象的可中断属性设置为“off”时，这才有意义。中断回调对象的busyaction属性指示中断回调是排队(“queue”(默认))还是丢弃(“cancel”)。参见回调一节。

interruptible: "off" | {"on"}

指定此对象的回调函数是否可能被其他回调中断。默认情况下，可中断状态为“on”，使用drawnow、figure、waitfor、getframe或pause函数的回调函数最终会被中断。参见回调一节。

menuselectedfcn: string | function handle, def. [](0x0)

函数，在执行此菜单项时调用。有关如何编写图形监听器函数的信息，请参阅回调部分。

创建/删除

beingdeleted: {"off"} | "on"

属性，指示函数已开始删除对象。Beingdeleted被设置为true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

创建ui菜单后立即执行的回调函数。函数通过使用根对象上的默认属性来设置，例如set (root， "defaultuimenucreatefcn"， 'disp ("uimenu created!")')。

有关如何编写图形监听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除ui菜单之前立即执行的回调函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

显示

clipping: "off" | {"on"}

如果裁剪为“on”，则菜单在其父轴限制内被裁剪。

visible: "off" | {"on"}

如果“可见”为“关闭”，则界面不呈现在屏幕上。

键盘交互

accelerator: character, def. ""

当与CTRL键同时按下时将执行该菜单项的字符(例如，“x”表示CTRL+x)。

菜单选项

checked: {"off"} | "on"

设置是否在此菜单项上显示标记。

enable: "off" | {"on"}

设置此菜单项是处于活动状态还是显示为灰色。

text: string, def. ""

这个菜单条目的文本。“&”字符可用于标记“加速键”

鼠标交互

buttondownfcn: string | function handle, def. [](0x0)

Buttondownfcn未使用。

contextmenu: graphics handle, def. [](0x0)

uicontextmenu对象的图形句柄，该对象当前与该uimenu对象相关联。

hittest: "off" | {"on"}

指定uimenu是处理鼠标事件还是将其传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定ui菜单是否接受鼠标点击。默认情况下，pickableparts是“可见的”，只有ui菜单的可见部分或它的子部分可能对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性指示是否选择此菜单。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为“on”，则ui菜单的选择状态将被高亮显示。

目标识别

tag: string, def. ""

一个用户定义的字符串，用于标记图形对象。

type (read-only): string

图形对象的类名。类型总是“ui菜单”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

对象的位置

position: scalar, def. 4

一个标量值，根据菜单的方向，包含从左侧或顶部开始的相对菜单位置。

父/子

children (read-only): vector of graphics handles, def. [](0x1)

ui菜单子节点的图形句柄。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility为"off"，则ui菜单的句柄在其父属性的"children"中不可见。

parent: graphics handle

父图形对象的句柄。

15.3.3.13 Uibuttongroup属性

uibuttongroup对象的属性(参见uibuttongroup):

类别:

外观|按钮组操作|回调执行|创建/删除|显示|鼠标交互|对象识别|对象位置|父/子|文本外观

外观

backgroundcolor: colorspec, def. [0.9400 0.9400 0.9400]

此按钮组的背景颜色值。

bordertype: "beveledin" | "beveledout" | {"etchedin"} | "etchedout" | "line" | "none"

设置是否在按钮组周围使用线边框。

borderwidth: whole number scalar, def. 1

以像素为单位的线边界宽度。

foregroundcolor: colorspec, def. [0 0 0]

此按钮组的标题文本的颜色值。

highlightcolor: colorspec, def. [1 1 1]

与此按钮组相邻的线的颜色值。

shadowcolor: colorspec, def. [0.7000 0.7000 0.7000]

环绕此按钮组的边框线的颜色值。

sizechangedfcn: string | function handle, def. [](0x0)

当按钮组大小改变时触发回调。

有关如何编写图形监听器函数的信息，请参阅回调部分。

按钮组操作

selectedobject: def. [](0x0)

按钮组中当前选定项的图形句柄。

回调执行

busyaction: "cancel" | {"queue"}

定义Octave在无法中断另一个对象的回调执行时如何处理该对象的回调属性的执行。只有当当前执行的回调对象的可中断属性设置为“off”时，这才有意义。中断回调对象的busyaction属性指示中断回调是排队(“queue”(默认))还是丢弃(“cancel”)。参见回调一节。

interruptible: "off" | {"on"}

指定此对象的回调函数是否可能被其他回调中断。默认情况下，可中断状态为“on”，使用drawnow、figure、waitfor、getframe或pause函数的回调函数最终会被中断。参见回调一节。

resizefcn: string | function handle, def. [](0x0)

Resizefcn已弃用。使用sizechangedfcn代替。

selectionchangedfcn: string | function handle, def. [](0x0)

当按钮组内所选项被更改时触发回调。

有关如何编写图形监听器函数的信息，请参阅回调部分。

创建/删除

beingdeleted: {"off"} | "on"

属性，指示函数已开始删除对象。Beingdeleted被设置为true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

在uibuttongroup创建后立即执行的回调函数。函数通过使用根对象上的默认属性来设置，例如set (root， "defaultuibuttongroupcreatefcn"， 'disp ("uibuttongroup created!")')。

有关如何编写图形监听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除uibuttongroup之前立即执行的回调函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

显示

clipping: "off" | {"on"}

如果clipping是“on”，uibuttongroup在它的父轴限制中被剪裁。

visible: "off" | {"on"}

如果visible为“off”，则uibuttongroup不呈现在屏幕上。

鼠标交互

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形监听器函数的信息，请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

当前与这个uibuttongroup对象相关联的uicontextmenu对象的图形句柄。

hittest: "off" | {"on"}

指定uibuttongroup是处理鼠标事件还是将它们传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定uibuttongroup是否接受鼠标点击。默认情况下，pickableparts是“可见的”，只有uibuttongroup或它的子元素的可见部分才会对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性指示是否选择此uibuttongroup。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为“on”，那么uibuttongroup的选择状态将被高亮显示。

目标识别

tag: string, def. ""

一个用户定义的字符串，用于标记图形对象。

type (read-only): string

图形对象的类名。类型总是“uibuttongroup”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

对象的位置

position: four-element vector, def. [0 0 1 1]

按钮组的大小表示为四元素向量[左，底，宽，高]。

units: "centimeters" | "characters" | "inches" | {"normalized"} | "pixels" | "points"

用来解释“位置”属性的测量单位。

父母/孩子

children (read-only): vector of graphics handles, def. [](0x1)

uibuttongroup子元素的图形句柄。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility是"off"， uibuttongroup的句柄在其父的"children"属性中是不可见的。

parent: graphics handle

父图形对象的句柄。

文本的外观

fontangle: "italic" | {"normal"}

控制字体是斜体还是普通。

fontname: string, def. "\*"

用于文本呈现的字体名称。设置此属性时，文本呈现引擎将在系统中搜索匹配的字体。如果没有找到，则使用默认的无衬线字体(与默认的“\*”值相同)呈现文本。

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”)。

fontsize: scalar, def. 10

用于文本呈现的字体大小。参见fontunits属性。

fontunits: "centimeters" | "inches" | "normalized" | "pixels" | {"points"}

用来解释"font - size"属性的单元。

fontweight: "bold" | {"normal"}

控制用于文本呈现的基本字体的变体。

title: string, def. ""

按钮组标题的文本。

titleposition: "centerbottom" | "centertop" | "leftbottom" | {"lefttop"} | "rightbottom" | "righttop"

按钮组中标题的相对位置。

**15.3.3.14 Uicontextmenu属性**

**uicontextmenu对象的属性(参见uicontextmenu):**

**类别:**

回调执行|创建/删除|显示|鼠标交互|对象识别|对象位置|父/子

**回调执行**

busyaction: "cancel" | {"queue"}

定义Octave在无法中断另一个对象的回调执行时如何处理该对象的回调属性的执行。只有当当前执行的回调对象的可中断属性设置为“off”时，这才有意义。中断回调对象的busyaction属性指示中断回调是排队(“queue”(默认))还是丢弃(“cancel”)。参见回调一节。

callback: string, def. [](0x0)

由有效的Octave表达式组成的字符串，该表达式将在选择此项时执行。

interruptible: "off" | {"on"}

指定此对象的回调函数是否可能被其他回调中断。默认情况下，可中断状态为“on”，使用drawnow、figure、waitfor、getframe或pause函数的回调函数最终会被中断。参见回调一节。

**创建/删除**

beingdeleted: {"off"} | "on"

属性，指示函数已开始删除对象。Beingdeleted被设置为true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

在uicontextmenu创建后立即执行的回调函数。函数通过使用根对象上的默认属性来设置，例如set (root， "defaultuicontextmenucreatefcn"， 'disp ("uicontextmenu created!")')。

有关如何编写图形监听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除uicontextmenu之前立即执行的回调函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

显示

clipping: "off" | {"on"}

如果裁剪为“on”，则uicontextmenu在其父轴限制内被裁剪。

visible: "off" | {"on"}

如果visible为“off”，则uicontextmenu不呈现在屏幕上。

鼠标交互

buttondownfcn: string | function handle, def. [](0x0)

Buttondownfcn未使用。

contextmenu: graphics handle, def. [](0x0)

当前与uicontextmenu对象关联的uicontextmenu对象的图形句柄。

hittest: "off" | {"on"}

指定uicontextmenu是处理鼠标事件还是将其传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定uicontextmenu是否接受鼠标点击。默认情况下，pickableparts是“可见的”，只有uicontextmenu或它的子菜单的可见部分才会对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性指示是否选中此uicontextmenu。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为“on”，则uicontextmenu的选择状态将被高亮显示。

目标识别

tag: string, def. ""

一个用户定义的字符串，用于标记图形对象。

type (read-only): string

图形对象的类名。类型总是“uicontextmenu”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

客体位置

position: def. [0 0]

手动设置uicontextmenu出现的位置尚未实现uicontextmenu对象。位置未使用。

父母/孩子

children (read-only): vector of graphics handles, def. [](0x1)

uicontextmenu子元素的图形句柄。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility为"off"，则uicontextmenu的句柄在其父属性的"children"中不可见。

parent: graphics handle

父图形对象的句柄。

**15.3.3.15 Uipanel属性**

uipanel对象属性(参见uipanel):

类别:

外观|回调执行|创建/删除|显示|鼠标交互|对象识别|对象位置|父/子|文本外观

外观

backgroundcolor: colorspec, def. [0.9400 0.9400 0.9400]

该面板背景的颜色值。

bordertype: "beveledin" | "beveledout" | {"etchedin"} | "etchedout" | "line" | "none"

设置是否在面板周围设置线边框。

borderwidth: whole number scalar, def. 1

以像素为单位的线边界宽度。

foregroundcolor: colorspec, def. [0 0 0]

此面板的标题文本的颜色值。

highlightcolor: colorspec, def. [1 1 1]

与该面板相邻的线的颜色值。

shadowcolor: colorspec, def. [0.7000 0.7000 0.7000]

围绕这个面板的边框线的线的颜色值。看到colorspec。

**回调执行**

busyaction: "cancel" | {"queue"}

定义Octave在无法中断另一个对象的回调执行时如何处理该对象的回调属性的执行。只有当当前执行的回调对象的可中断属性设置为“off”时，这才有意义。中断回调对象的busyaction属性指示中断回调是排队(“queue”(默认))还是丢弃(“cancel”)。参见回调一节。

interruptible: "off" | {"on"}

指定此对象的回调函数是否可能被其他回调中断。默认情况下，可中断状态为“on”，使用drawnow、figure、waitfor、getframe或pause函数的回调函数最终会被中断。参见回调一节。

resizefcn: string | function handle, def. [](0x0)

Resizefcn已弃用。使用sizechangedfcn代替。

sizechangedfcn: string | function handle, def. [](0x0)

当面板大小改变时触发回调。

有关如何编写图形监听器函数的信息，请参阅回调部分。

**创建/删除**

beingdeleted: {"off"} | "on"

属性，指示函数已开始删除对象。Beingdeleted被设置为true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

在uipanel创建后立即执行的回调函数。函数是通过在根对象上使用默认属性来设置的，例如set (root， "defaultuipanelcreatefcn"， 'disp ("uipanel created!")')。

有关如何编写图形监听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除uipanel之前立即执行的回调函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

**显示**

clipping: "off" | {"on"}

如果clip是“on”，uipanel在它的父轴限制内被clip。

visible: "off" | {"on"}

如果visible是“off”，uipanel不会在屏幕上渲染。

**鼠标交互**

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形监听器函数的信息，请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

当前与这个uipanel对象关联的uicontextmenu对象的图形句柄。

hittest: "off" | {"on"}

指定uipanel是处理鼠标事件还是将其传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定uipanel是否接受鼠标点击。默认情况下，pickableparts是“可见的”，只有uipanel或它的子面板的可见部分才会对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性指示该uipanel是否被选中。

selectionhighlight: "off" | {"on"}

如果selectionhighlight是“on”，那么uipanel的选择状态将被高亮显示。

**目标识别**

tag: string, def. ""

一个用户定义的字符串，用于标记图形对象。

type (read-only): string

图形对象的类名。类型总是“uipanel”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

**对象的位置**

position: four-element vector, def. [0 0 1 1]

面板的大小表示为四元素向量[左，底，宽，高]。

units: "centimeters" | "characters" | "inches" | {"normalized"} | "pixels" | "points"

用来解释“位置”属性的测量单位。

**父母/孩子**

children (read-only): vector of graphics handles, def. [](0x1)

uipanel子面板的图形句柄。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility是"off"， uipanel的句柄在其父的"children"属性中不可见。

parent: graphics handle

父图形对象的句柄。

文本的外观

fontangle: "italic" | {"normal"}

控制字体是斜体还是普通。

fontname: string, def. "\*"

用于文本呈现的字体名称。设置此属性时，文本呈现引擎将在系统中搜索匹配的字体。如果没有找到，则使用默认的无衬线字体(与默认的“\*”值相同)呈现文本。

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”)。

fontsize: scalar, def. 10

用于文本呈现的字体大小。参见fontunits属性。

fontunits: "centimeters" | "inches" | "normalized" | "pixels" | {"points"}

用来解释"font - size"属性的单元。

fontweight: "bold" | {"normal"}

控制用于文本呈现的基本字体的变体。

title: string, def. ""

面板标题的文本。

titleposition: "centerbottom" | "centertop" | "leftbottom" | {"lefttop"} | "rightbottom" | "righttop"

Relative position of the title within the panel.

标题在面板中的相对位置。

15.3.3.16 Uicontrol属性

uicontrol对象的属性(参见uicontrol):

类别:

外观|回调执行|控制选项|创建/删除|显示|鼠标交互|对象识别|对象位置|父/子|文本外观

外观

backgroundcolor: colorspec, def. [0.9400 0.9400 0.9400]

此控件对象的背景颜色值。

cdata: array, def. [](0x0)

用于表示控制对象的图像数据，存储为M × N × 3 RGB数组。

extent (read-only): four-element vector

与[0 0 width height]形式返回的uicontrol相关联的文本字符串的大小(前两个元素始终为零)。

对于多行字符串，返回的宽度和高度表示包含所有行的矩形的大小。

foregroundcolor: colorspec, def. [0 0 0]

此控件对象的文本的颜色值。看到colorspec。

style: "checkbox" | "edit" | "frame" | "listbox" | "popupmenu" | {"pushbutton"} | "radiobutton" | "slider" | "text" | "togglebutton"

创建的控件对象的类型。有关可用控件样式的完整描述，请参阅"uicontrol"函数

**回调执行**

busyaction: "cancel" | {"queue"}

定义Octave在无法中断另一个对象的回调执行时如何处理该对象的回调属性的执行。只有当当前执行的回调对象的可中断属性设置为“off”时，这才有意义。中断回调对象的busyaction属性指示中断回调是排队(“queue”(默认))还是丢弃(“cancel”)。参见回调一节。

callback: string, def. [](0x0)

由有效Octave表达式组成的字符串，该表达式将在激活此控件时执行。

interruptible: "off" | {"on"}

指定此对象的回调函数是否可能被其他回调中断。默认情况下，可中断状态为“on”，使用drawnow、figure、waitfor、getframe或pause函数的回调函数最终会被中断。参见回调一节。

keypressfcn: string | function handle, def. [](0x0)

当按下键并且控制对象有焦点时执行的函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

控制选项

enable: "inactive" | "off" | {"on"}

设置此控件对象是活动还是显示为灰色。

listboxtop: scalar, def. 1

将出现在“列表框”控件顶部的字符串选项的索引

max: scalar, def. 1

最大控制值，其对控件的影响取决于控件类型。对于“复选框”、“切换按钮”和“单选按钮”控件，当控件对象被选中时，将“max”值赋给“value”属性。对于“slider”控件，“max”定义了滑块的最大值。对于“edit”和“listbox”控件，如果Max - Min > 1，则该控件将分别允许多个行项或列表项选择。

min: scalar, def. 0

最小控制值，其对控件的影响取决于控件类型。对于“复选框”、“切换按钮”和“单选按钮”控件，当控件对象未被选中时，将“最小”值赋给“值”属性。对于“slider”控件，“min”定义了滑块的最小值。对于“edit”和“listbox”控件，如果Max - Min > 1，则该控件将分别允许多个行项或列表项选择。

sliderstep: two-element vector, def. [0.010000 0.100000]

相对于滑块的最小-最大跨度测量的分数步长，当用户单击对象时，滑块移动的步长。“sliderstep”被指定为由[minor major]组成的双元素向量，其中“minor”是单击滑块箭头的步长，“major”是在滑块条内单击的步长。

value: scalar, def. 0

与控件对象的当前状态相关联的数值，该值的含义取决于控件对象的“样式”。

创建/删除

beingdeleted: {"off"} | "on"

属性，指示函数已开始删除对象。Beingdeleted被设置为true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

在uicontrol创建后立即执行的回调函数。函数通过在根对象上使用默认属性来设置，例如set (root， "defaultuicontrolcreatefcn"， 'disp ("uicontrol created!")')。

有关如何编写图形监听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除uicontrol之前立即执行的回调函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

显示

clipping: "off" | {"on"}

如果裁剪为“on”，则uicontrol将在其父轴限制内被裁剪。

visible: "off" | {"on"}

如果visible为“off”，则该控件不会在屏幕上呈现。

**鼠标交互**

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形监听器函数的信息，请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

当前与uiconcontrol对象关联的uicontextmenu对象的图形句柄。

hittest: "off" | {"on"}

指定uiconcontrol是处理鼠标事件还是将其传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定uiconcontrol是否接受鼠标点击。默认情况下，pickableparts是“可见的”，只有uicontrol或它的子控件的可见部分才会对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性指示是否选择此控件。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为“on”，则uicontrol的选择状态将被高亮显示。

tooltipstring: string, def. ""

当鼠标指针悬停在控件对象上时出现在工具提示中的文本字符串。

目标识别

tag: string, def. ""

一个用户定义的字符串，用于标记图形对象。

type (read-only): string

图形对象的类名。类型总是“uicontrol”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

对象的位置

position: four-element vector, def. [0 0 80 30]

控件对象的大小，表示为四元素向量[左，底，宽，高]。

units: "centimeters" | "characters" | "inches" | "normalized" | {"pixels"} | "points"

用来解释“位置”属性的测量单位。

父母/孩子

children (read-only): vector of graphics handles, def. [](0x1)

控件子控件的图形句柄。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility为"off"，则uicontrol的句柄在其父的"children"属性中不可见。

parent: graphics handle

父图形对象的句柄。

文本的外观

fontangle: "italic" | {"normal"}

控制字体是斜体还是普通。

fontname: string, def. "\*"

用于文本呈现的字体名称。设置此属性时，文本呈现引擎将在系统中搜索匹配的字体。如果没有找到，则使用默认的无衬线字体(与默认的“\*”值相同)呈现文本。

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”)。

fontsize: scalar, def. 10

用于文本呈现的字体大小。参见fontunits属性。

fontunits: "centimeters" | "inches" | "normalized" | "pixels" | {"points"}

用来解释"font - size"属性的单元。

fontweight: "bold" | {"normal"}

控制用于文本呈现的基本字体的变体。

horizontalalignment: {"center"} | "left" | "right"

指定uicontrol对象内文本的水平对齐。

string: string, def. ""

与控件对象一起出现的文本。

verticalalignment: "bottom" | {"middle"} | "top"

指定文本在uicontrol对象中的垂直位置。

**15.3.3.17 Uitable属性**

uitable对象的属性(参见uitable):

类别:

外观|回调执行|创建/删除|显示|鼠标交互|对象识别|对象位置|父/子|表数据|表操作|文本外观

外观

backgroundcolor: colorspec, def. 2-by-3 double

指定为3元素RBG向量的表背景的颜色。如果backgroundcolor有多行，则在“rowstriping”属性开启时，颜色会重复循环。

foregroundcolor: colorspec, def. [0 0 0]

该表中数据文本的颜色。看到colorspec。

rowstriping: "off" | {"on"}

设置，以指示表背景颜色是否将为每一行使用不同的颜色。颜色是从重复图案中的“backgroundcolor”属性绘制的。

回调执行

busyaction: "cancel" | {"queue"}

定义Octave在无法中断另一个对象的回调执行时如何处理该对象的回调属性的执行。只有当当前执行的回调对象的可中断属性设置为“off”时，这才有意义。中断回调对象的busyaction属性指示中断回调是排队(“queue”(默认))还是丢弃(“cancel”)。参见回调一节。

celleditcallback: string, def. [](0x0)

由有效的Octave表达式组成的字符串，每当编辑表格单元格时将执行该表达式。

cellselectioncallback: string, def. [](0x0)

由有效的Octave表达式组成的字符串，每当选择表格单元格时将执行该表达式。

interruptible: "off" | {"on"}

指定此对象的回调函数是否可能被其他回调中断。默认情况下，可中断状态为“on”，使用drawnow、figure、waitfor、getframe或pause函数的回调函数最终会被中断。参见回调一节。

keypressfcn: string, def. [](0x0)

一个字符串，由一个有效的Octave表达式组成，只要在这个表对象中按下一个键，这个表达式就会执行。

keyreleasefcn: def. [](0x0)

由一个有效的Octave表达式组成的字符串，该表达式将在此表对象中释放键时执行。

**创建/删除**

beingdeleted: {"off"} | "on"

属性，指示函数已开始删除对象。Beingdeleted被设置为true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

创建后立即执行的回调函数。函数通过使用根对象上的默认属性来设置，例如set (root， "defaultuitablecreatefcn"， 'disp ("uitable created!")')。

有关如何编写图形监听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在uitable被删除之前执行的回调函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

**显示**

clipping: "off" | {"on"}

如果裁剪为“on”，则合适的对象在其父轴限制内被裁剪。

visible: "off" | {"on"}

如果visible为“off”，则不显示在屏幕上。

**鼠标交互**

buttondownfcn: string | function handle, def. [](0x0)

有关如何编写图形监听器函数的信息，请参阅回调部分。

contextmenu: graphics handle, def. [](0x0)

uicontextmenu对象的图形句柄，该对象当前与此合适的对象相关联。

hittest: "off" | {"on"}

指定是处理鼠标事件还是将其传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定是否接受鼠标点击。默认情况下，pickableparts是“可见的”，只有可显示的部分或它的子部分可能对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性指示是否选择此适合项。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为“on”，则显示该对象的选择状态。

tooltipstring: string, def. ""

当鼠标指针悬停在表对象上时出现在工具提示中的文本字符串。

目标识别

tag: string, def. ""

一个用户定义的字符串，用于标记图形对象。

type (read-only): string

图形对象的类名。类型总是“合适的”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

对象的位置

extent (read-only): four-element vector

表示表大小的元素向量。数组的前两个元素始终为零，而第三和第四个元素包含表的高度和宽度。

position: four-element vector, def. [20 20 300 300]

表的位置和大小。向量的四个元素是左下角的坐标和图形的宽度和高度。参见单位属性。

units: "centimeters" | "characters" | "inches" | "normalized" | {"pixels"} | "points"

用来解释“位置”属性的测量单位。

父母/孩子

children (read-only): vector of graphics handles, def. [](0x1)

适合子对象的图形句柄。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility为"off"，则该对象的句柄在其父对象的"children"属性中不可见。

parent: graphics handle

父图形对象的句柄。

表数据

columnformat: def. {}(0x0)

每列中数字数据的显示格式。有效的格式包括"char"、"logical"、"numeric"或format函数中的有效格式设置

columnname: def. "numbered"

列名指定为“编号”或1 x N单元格字符串向量，其中包含用于每个列标题的名称。

columnwidth: def. "auto"

设置确定每列的宽度，有效的选项包括:“auto”，“fit”，指定为“1x”，“2x”等的平均倍数，或1x N单元格向量，其中每个元素对应于N个表列中的一个，并包含上述任何选项或以像素指定的固定宽度。

data: matrix, def. [](0x0)

表中包含的数据指定为二维数字、逻辑或单元格数组。

rowname: def. "numbered"

行名指定为“编号”或N x 1单元格字符串向量，其中包含用于每个行标题的名称。

表操作

columneditable: logical row vector, def. [](0x0)

表示列是否可编辑的逻辑指示器。它由一个1 x N的逻辑值向量组成，其中true或false分别表示相应的列是可编辑的或不可编辑的，或者由一个空逻辑数组组成，表示没有列是不可编辑的。

enable: "off" | {"on"}

设置此表对象是活动还是显示为灰色。

rearrangeablecolumns: {"off"} | "on"

指示是否能够通过单击和拖动列标头来移动列。

文本的外观

fontangle: "italic" | {"normal"}

控制字体是斜体还是普通。

fontname: string, def. "\*"

用于文本呈现的字体名称。设置此属性时，文本呈现引擎将在系统中搜索匹配的字体。如果没有找到，则使用默认的无衬线字体(与默认的“\*”值相同)呈现文本。

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”)。

fontsize: scalar, def. 10

用于文本呈现的字体大小。参见fontunits属性。

fontunits: "centimeters" | "inches" | "normalized" | "pixels" | {"points"}

用来解释"font - size"属性的单元。

fontweight: "bold" | {"normal"}

控制用于文本呈现的基本字体的变体。

15.3.3.18工具栏属性

工具栏对象的属性(参见工具栏):

类别:

回调执行|创建/删除|显示|鼠标交互|对象识别|父/子

回调执行

busyaction: "cancel" | {"queue"}

定义Octave在无法中断另一个对象的回调执行时如何处理该对象的回调属性的执行。只有当当前执行的回调对象的可中断属性设置为“off”时，这才有意义。中断回调对象的busyaction属性指示中断回调是排队(“queue”(默认))还是丢弃(“cancel”)。参见回调一节。

interruptible: "off" | {"on"}

指定此对象的回调函数是否可能被其他回调中断。默认情况下，可中断状态为“on”，使用drawnow、figure、waitfor、getframe或pause函数的回调函数最终会被中断。参见回调一节。

创建/删除

beingdeleted: {"off"} | "on"

属性，指示函数已开始删除对象。Beingdeleted被设置为true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

在工具栏创建后立即执行的回调函数。函数通过使用根对象上的默认属性来设置，例如set (root， "defaultuitoolbarcreatefcn"， 'disp ("uitoolbar created!")')。

有关如何编写图形监听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在工具栏被删除之前立即执行的回调函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

显示

clipping: "off" | {"on"}

如果“打开”，则工具栏将在其父轴限制内被剪切。

visible: "off" | {"on"}

如果“可见”为“关闭”，则工具栏不会在屏幕上呈现。

鼠标交互

buttondownfcn: string | function handle, def. [](0x0)

buttondownfcn未使用。

contextmenu: graphics handle, def. [](0x0)

当前与此工具栏对象关联的uicontextmenu对象的图形句柄。

hittest: "off" | {"on"}

指定uitoolbar是处理鼠标事件还是将其传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定工具栏是否接受鼠标点击。默认情况下，pickableparts是“可见的”，只有工具栏的可见部分或它的子部分可能对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性指示是否选择此工具栏。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为“on”，则工具栏的选择状态将被可视化地高亮显示。

目标识别

tag: string, def. ""

一个用户定义的字符串，用于标记图形对象。

type (read-only): string

图形对象的类名。类型总是“uitoolbar”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

父母/孩子

children (read-only): vector of graphics handles, def. [](0x1)

工具栏子节点的图形句柄。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility为"off"，则工具栏的句柄在其父的"children"属性中不可见。

parent: graphics handle

父图形对象的句柄。

15.3.3.19 Uipushtool属性

uipushtool对象的属性(参见uipushtool):

类别:

外观|回调执行|创建/删除|显示|鼠标交互|对象识别|父/子|推送工具操作

外观

\_named\_icon\_\_: string, def. ""

要用作推送工具对象的图像的绑定图标文件的名称。

cdata: array, def. [](0x0)

用于表示pushtool对象的

separator: {"off"} | "on"

指示是否在当前冲压工具位置旁边绘制分隔线的状态。

回调执行

busyaction: "cancel" | {"queue"}

定义Octave在无法中断另一个对象的回调执行时如何处理该对象的回调属性的执行。只有当当前执行的回调对象的可中断属性设置为“off”时，这才有意义。中断回调对象的busyaction属性指示中断回调是排队(“queue”(默认))还是丢弃(“cancel”)。参见回调一节。

clickedcallback: string, def. [](0x0)

由有效的Octave表达式组成的字符串，该表达式将在单击此控件对象时执行。

interruptible: "off" | {"on"}

指定此对象的回调函数是否可能被其他回调中断。默认情况下，可中断状态为“on”，使用drawnow、figure、waitfor、getframe或pause函数的回调函数最终会被中断。参见回调一节。

创建/删除

beingdeleted: {"off"} | "on"

属性，指示函数已开始删除对象。Beingdeleted被设置为true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

uipushtool创建完成后立即执行的回调函数。函数通过使用根对象上的默认属性来设置，例如set (root， "defaultuipushtoolcreatefcn"， 'disp ("uipushtool created!")')。

有关如何编写图形监听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

删除upushtool之前立即执行的回调函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

显示

clipping: "off" | {"on"}

如果裁剪为“on”，则upushtool将被裁剪在其父轴限制内。

visible: "off" | {"on"}

如果“可见”为“关闭”，则upushtool不会在屏幕上呈现。

鼠标交互

buttondownfcn: string | function handle, def. [](0x0)

Buttondownfcn未使用。

contextmenu: graphics handle, def. [](0x0)

当前与uipushtool对象相关联的uicontextmenu对象的图形句柄。

hittest: "off" | {"on"}

指定uipushtool是处理鼠标事件还是将其传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定upushtool是否接受鼠标点击。默认情况下，pickableparts是“可见的”，只有uppushtool或其子工具的可见部分才会对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性指示是否选择此upushtool。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为“on”，则upushtool的选择状态将被可视化地高亮显示。

tooltipstring: string, def. ""

当鼠标指针悬停在推送工具对象上时出现在工具提示中的文本字符串。

目标识别

tag: string, def. ""

一个用户定义的字符串，用于标记图形对象。

type (read-only): string

图形对象的类名。类型总是“upushtool”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

父母/孩子

children (read-only): vector of graphics handles, def. [](0x1)

upushtool子进程的图形句柄。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility为"off"，则upushtool的句柄在其父的"children"属性中不可见。

parent: graphics handle

父图形对象的句柄。

Pushtool操作

enable: "off" | {"on"}

设置此推送工具对象是处于活动状态还是显示为灰色。

15.3.3.20 Uitoggletool属性

uitoggletool对象的属性(参见uitoggletool):

类别:

外观|回调执行|创建/删除|显示|鼠标交互|对象识别|父/子|切换操作

外观

\_\_named\_icon\_\_: string, def. ""

要用作toggletool对象的图像的绑定图标文件的名称。

cdata: array, def. [](0x0)

用于表示toggletool对象的图像数据，存储为M × N × 3 RGB数组。

separator: {"off"} | "on"

设置为在工具栏的左侧绘制一条垂直线。

回调执行

busyaction: "cancel" | {"queue"}

定义Octave在无法中断另一个对象的回调执行时如何处理该对象的回调属性的执行。只有当当前执行的回调对象的可中断属性设置为“off”时，这才有意义。中断回调对象的busyaction属性指示中断回调是排队(“queue”(默认))还是丢弃(“cancel”)。参见回调一节。

clickedcallback: string, def. [](0x0)

由有效的Octave表达式组成的字符串，该表达式将在单击此控件对象时执行。

interruptible: "off" | {"on"}

指定此对象的回调函数是否可能被其他回调中断。默认情况下，可中断状态为“on”，使用drawnow、figure、waitfor、getframe或pause函数的回调函数最终会被中断。参见回调一节。

offcallback: string, def. [](0x0)

由有效的Octave表达式组成的字符串，该表达式将在此控制对象被关闭时执行。

oncallback: string, def. [](0x0)

由有效Octave表达式组成的字符串，该表达式将在打开此控制对象时执行。

创建/删除

beingdeleted: {"off"} | "on"

属性，指示函数已开始删除对象。Beingdeleted被设置为true，直到对象不再存在。

createfcn: string | function handle, def. [](0x0)

创建uitoggletool后立即执行的回调函数。函数通过使用根对象上的默认属性来设置，例如set (root， "defaultuitoggletoolcreatefcn"， 'disp (" uitoggleool created!")')。

有关如何编写图形监听器函数的信息，请参阅回调部分。

deletefcn: string | function handle, def. [](0x0)

在删除uitoggletool之前立即执行的回调函数。

有关如何编写图形监听器函数的信息，请参阅回调部分。

显示

clipping: "off" | {"on"}

如果裁剪是“on”，则uitoggletool在其父轴限制内被裁剪。

visible: "off" | {"on"}

如果visible为“off”，则该工具不会在屏幕上呈现。

鼠标交互

buttondownfcn: string | function handle, def. [](0x0)

Buttondownfcn未使用。

contextmenu: graphics handle, def. [](0x0)

uicontextmenu对象的图形句柄，当前与此uitoggletool对象关联。

hittest: "off" | {"on"}

指定uitoggletool是处理鼠标事件还是将其传递给对象的祖先。当启用时，对象可以通过评估“buttondownfcn”来响应鼠标点击，显示uicontextmenu，并最终成为根“currentobject”。此属性仅在对象可以接受由“pickableparts”属性决定的鼠标点击时才相关。参见pickableparts属性。

pickableparts: "all" | "none" | {"visible"}

指定uitoggletool是否接受鼠标点击。默认情况下，pickableparts是“可见的”，并且只有uitoggletool或其子组件的可见部分才会对鼠标点击做出反应。当pickableparts为“all”时，可见部分和不可见部分(或子部分)都可能对鼠标点击做出反应。当pickableparts为“none”时，对象上的鼠标点击将被忽略，并传输到该对象下面的任何对象。当一个对象被配置为接受鼠标点击时，“hittest”属性将决定如何处理它们。参见hittest属性。

selected: {"off"} | "on"

属性指示是否选择此工具。

selectionhighlight: "off" | {"on"}

如果selectionhighlight为“on”，则uitoggletool的选择状态将被可视化地高亮显示。

tooltipstring: string, def. ""

当鼠标指针悬停在工具工具对象上时出现在工具提示中的文本字符串。

目标识别

tag: string, def. ""

一个用户定义的字符串，用于标记图形对象。

type (read-only): string

图形对象的类名。类型总是“uitoggletool”。

userdata: Any Octave data, def. [](0x0)

与图形对象关联的用户定义数据。

父母/孩子

children (read-only): vector of graphics handles, def. [](0x1)

uitoggletool子进程的图形句柄。

handlevisibility: "callback" | "off" | {"on"}

如果handlevisibility为"off"，则uitoggletool的句柄在其父的"children"属性中不可见。

parent: graphics handle

父图形对象的句柄。

切换操作

enable: "off" | {"on"}

设置此工具对象是活动的还是变灰的。

state: {"off"} | "on"

toggletool对象的当前“开”或“关”状态。

**15.3.4搜索属性**

: h = findobj ()

: h = findobj (prop\_name, prop\_value, …)

: h = findobj (prop\_name, prop\_value, "-logical\_op", prop\_name, prop\_value)

: h = findobj ("-property", prop\_name)

: h = findobj ("-regexp", prop\_name, pattern)

: h = findobj (hlist, …)

: h = findobj (hlist, "flat", …)

: h = findobj (hlist, "-depth", d, …)

查找具有指定属性的图形对象。

当不带参数调用时，返回从根对象(0)开始并包括其所有后代的所有图形对象。

缩小结果范围的最简单形式是

findobj (prop\_name, prop\_value)

它返回所有对象的句柄，这些对象的属性名为prop\_name，其值为prop\_value。如果指定了多个属性/值对，则只返回满足所有条件(相当于-and)的对象。

通过传递句柄或句柄集hlist作为第一个参数，可以将搜索限制在一组特定的对象及其后代。

可以使用"-depth"参数限制要搜索的对象层次结构的深度。一个只搜索三代子代的例子是:

findobj (hlist, "-depth", 3, prop\_name, prop\_value)

指定深度d为0将搜索限制在hlist中传递的对象集合中。深度为0也等同于“平坦”参数。默认的深度值是Inf，它包括所有的后代。

在prop\_name、prop\_value对之间可以使用指定的逻辑运算符。支持的逻辑运算符为:-and， -or， -xor， -not。定位所有图形和轴对象的示例代码是

findobj ("type", "figure", "-or", "type", "axes")

也可以通过将正则表达式与属性值进行比较来匹配对象，其中返回匹配regexp (prop\_value, pattern)的属性值。

最后，可以使用“-property”选项找到具有属性名称的对象。例如，定位带有“meshstyle”属性的对象的代码是

findobj ("-property", "meshstyle")

实现注意:搜索只包括具有可见句柄的对象(HandleVisibility = "on")。参见findall，搜索包括隐藏对象在内的所有对象。

参见:findall, allchild, get, set。

: h = findall ()

: h = findall (prop\_name, prop\_value, …)

: h = findall (prop\_name, prop\_value, "-logical\_op", prop\_name, prop\_value)

: h = findall ("-property", prop\_name)

: h = findall ("-regexp", prop\_name, pattern)

: h = findall (hlist, …)

: h = findall (hlist, "flat", …)

: h = findall (hlist, "-depth", d, …)

查找具有指定属性的图形对象，包括隐藏对象。

返回值h是找到的图形对象的句柄列表。

findall执行与findobj相同的搜索，但它包含隐藏对象(HandleVisibility = "off")。有关完整文档，请参阅findobj。

参见:findobj, allchild, get, set。

15.3.5管理默认属性

对象属性有两类默认值，即工厂默认值(初始值)和用户定义的默认值，后者可以覆盖工厂默认值。

虽然可以为任何对象设置默认值，但它们是在父对象中设置的，并应用于指定对象类型的子对象。例如，将线对象的默认颜色属性设置为“绿色”，对于根对象，将导致所有线对象继承颜色“绿色”作为默认值。

set (groot, "defaultlinecolor", "green");

为所有对象设置默认的线条颜色。构造属性名以设置默认值的规则是

default + object-type + property-name

该规则可能会导致一些奇怪的名称，例如defaultlinelinewidth"指定行对象的默认线宽属性。

上面的例子使用了根对象，所以默认属性值将应用于所有行对象。但是，默认值是分层的，因此在图形对象中设置的默认值将覆盖在根对象中设置的默认值。同样，在轴对象中设置的默认值覆盖在图或根对象中设置的默认值。例如,

subplot (2, 1, 1);

set (groot, "defaultlinecolor", "red");

set (1, "defaultlinecolor", "green");

set (gca (), "defaultlinecolor", "blue");

line (1:10, rand (1, 10));

subplot (2, 1, 2);

line (1:10, rand (1, 10));

figure (2)

line (1:10, rand (1, 10));

产生两个数字。第一个图的第一个子图窗口中的线是蓝色的，因为它继承了它的父轴对象的颜色。第一个图的第二个子图窗口中的线是绿色的，因为它继承了它的父图对象的颜色。第二个图形窗口中的行是红色的，因为它从全局根对象继承了它的颜色。

如果需要删除用户自定义的默认设置，可以将默认属性设置为“remove”。例如,

set (gca (), "defaultlinecolor", "remove");

从当前轴对象中移除用户定义的默认线条颜色设置。使用reset功能可以快速清除所有用户自定义的默认值。

默认情况下，高级绘图功能(如绘图重置和重新定义轴属性)独立于默认值。这种属性的一个例子是axesbox属性:它是由高级2d图形函数设置的，而不管属性“defaultaxesbox”。使用hold函数来防止这种行为:

set (groot, "defaultaxesbox", "off");

subplot (2, 1, 1);

plot (1:10)

title ("Box is on anyway")

subplot (2, 1, 2);

hold on

plot (1:10)

title ("Box is off")

: reset (h)

将图形对象h的属性重置为其默认值。

对于图形，属性“位置”、“单位”、“窗口样式”和“纸张单位”不受影响。对于轴，属性“位置”和“单位”不受影响。

输入h也可以是图形句柄的向量，在这种情况下，每个单独的对象将被重置。

参见:cla, clf, newplot。

获取对象的“default”属性将返回为该对象设置的用户定义默认值列表。例如,

get (gca (), "default");

返回当前坐标轴对象的用户定义默认值列表。

工厂默认值存储在根对象中。命令

get (groot, "factory");

返回出厂默认值的列表。

**15.4高级绘图**

**15.4.1颜色**

可以通过三种方式指定颜色:1)RGB三元组，2)通过名称，或3)通过HTML符号。

RGB值

RGB三元组是一个1x3向量，其中每个值都在0到1之间。第一个值表示红色的百分比，第二个值表示绿色的百分比，第三个值表示蓝色的百分比。例如，[1,0,1]表示完整的红色和蓝色通道，从而产生品红。

短名称或长名称

八种颜色可以通过名称直接指定，也可以通过单个字符短名称指定。

| **Name** | **Color** |
| --- | --- |
| ‘*k*’, *"black"* | blacK |
| ‘*r*’, *"red"* | Red |
| ‘*g*’, *"green"* | Green |
| ‘*b*’, *"blue"* | Blue |
| ‘*y*’, *"yellow"* | Yellow |
| ‘*m*’, *"magenta"* | Magenta |
| ‘*c*’, *"cyan"* | Cyan |
| ‘*w*’, *"white"* | White |

HTML符号

HTML表示法是一个字符串，以字符“#”开头，后跟3或6个十六进制数字。与RGB三元组一样，每个十六进制数表示以指定颜色呈现的红、绿和蓝通道的比例。例如，“#FF00FF”代表洋红色。

**15.4.2线条样式**

行样式由以下属性指定:

linestyle

可能是其中之一

“-”

实线。(默认)

“--”

虚线。

“:”

虚线。

“-.”

虚线。

“none”

没有线。点仍将使用当前的标记样式进行标记。

linewidth

指定行宽度的数字。默认为1。值2是默认值的两倍，等等。

**15.4.3标记样式**

标记样式由以下属性指定:

marker

表示在每个数据点上放置绘图标记的字符，或“无”，表示不显示任何标记。

Markeredgecolor

标记周围边缘的颜色，或“自动”，意思是边缘颜色与面颜色相同。看到颜色。

markerfacecolor

标记的颜色，或“none”表示不应该填充标记。看到颜色。

markersize

指定标记大小的数字。默认为1。值2是默认值的两倍，以此类推。

colstyle函数将解析绘图样式规范，并返回结果的颜色、线条和标记值。

: [style, color, marker, msg] = colstyle (style)

解析行规范样式并返回给定的行样式、颜色和标记。

如果出现错误，字符串msg将返回错误的文本。

**15.4.4回调**

回调函数可以与图形对象相关联，并在某些事件发生后触发。所有回调函数的基本结构是

function mycallback (hsrc, evt)

…

endfunction

HSRC是回调源的句柄，evt给出一些特定于事件的数据。

该函数可以作为普通Octave函数的函数句柄、匿名函数或表示Octave命令的字符串提供。不建议使用后一种语法，因为只有在求值字符串时才会出现语法错误。参见函数句柄部分。

然后可以在对象创建时将其与对象关联，或者稍后使用set函数将其与对象关联。例如,

plot (x, "DeleteFcn", @(h, e) disp ("Window Deleted"))

其中，在情节被删除的时刻，将显示“窗口已删除”的消息。

附加的用户参数可以传递给回调函数，并将在两个默认参数之后传递。例如:

plot (x, "DeleteFcn", {@mycallback, "1"})

…

function mycallback (h, evt, arg1)

fprintf ("Closing plot %d\n", arg1);

endfunction

注意:回调函数中的第二个参数evt仅在Qt图形工具包中部分实现:

* 鼠标单击事件:evt是一个类双值:1表示左键，2表示中键，3表示右键。
* 按键事件:evt是一个包含字段Key(字符串)、Character(字符串)和Modifier(字符串单元格数组)的结构。
* 其他事件:evt是一个类双空矩阵。

可用于所有图形对象的基本回调函数是

* CreateFcn:在对象创建时调用。如果以任何方式改变了对象，则不会调用它，因此只有在定义对象的函数调用中定义此回调才有意义。稍后通过set函数添加到CreateFcn的回调将永远不会被执行。
* DeleteFcn:在对象被删除时调用。
* ButtonDownFcn:当鼠标指针在此对象上时按下鼠标按钮时调用。注意，gnuplot接口没有实现这个回调。

默认情况下，回调函数是排队的(它们在事件队列中一个接一个地执行)，除非使用了drawnow、figure、waitfor、getframe或pause函数。如果正在执行的回调调用其中一个函数，则会导致Octave刷新事件队列，从而导致正在执行的回调被中断。

可以通过将对象的interruptible属性设置为“off”来指定对象的回调不应该被中断。在这种情况下，Octave根据中断回调对象的busyaction属性决定做什么:

queue (the default)

中断回调在执行回调返回后执行。

cancel

中断的回调被丢弃。

当中断回调函数是deletefcn、figure resizefcn或closerequestfcn时，interruptible属性不起作用。那些回调总是中断正在执行的回调。

持有正在执行的回调的对象的句柄可以通过gbo函数获得。该对象的祖先图的句柄可以使用gcbf函数获得。

: h = gcbo ()

: [h, fig] = gcbo ()

返回当前正在执行回调的对象的句柄。

如果没有回调正在执行，这个函数返回空矩阵。这个句柄是从根对象属性“CallbackObject”中获得的。

当使用第二个输出参数调用时，返回包含当前正在执行回调的对象的图形句柄。如果没有回调正在执行，第二个输出也被设置为空矩阵。

参见:gcbf, gco, gca, gcf, get, set。

: fig = gcbf ()

返回包含当前正在执行回调的对象的图形的句柄。

如果没有回调正在执行，这个函数返回空矩阵。该函数返回的句柄与gcbo的第二个输出参数相同。

参见:gcbo, gcf, gco, gca, get, set。

回调也可以使用下面描述的addlistener函数添加到属性中。

**15.4.5应用定义数据**

Octave提供了将应用程序定义的数据附加到图形句柄的功能。数据可以是任何对应用程序有意义的数据，并且会被Octave完全忽略。

: setappdata (h, name, value)

: setappdata (h, name1, value1, name2, value3, …)

: setappdata (h, {name1, name2, …}, {value1, value2, …})

将具有句柄h的图形对象的应用程序数据名称设置为值。

H也可以是图形句柄的向量。如果指定名称的应用程序数据不存在，则创建该数据。

可以指定多个名称/值对。或者，可以指定名称的单元格数组和相应的值的单元格数组。有关获取有效应用程序数据属性列表的详细信息，请参见getappdata。

参见:getappdata, isappdata, rmappdata, guidata, get, set, getpref, setpref。

: value = getappdata (h, name)

: appdata = getappdata (h)

返回句柄为h的图形对象的应用程序数据名称的值。

H也可以是图形句柄的向量。如果没有给出第二个参数名称，那么getappdata返回一个结构体appdata，其字段对应于appdata属性。

参见:setappdata, isappdata, rmappdata, guidata, get, set, getpref, setpref。

: rmappdata (h, name)

: rmappdata (h, name1, name2, …)

从句柄为h的图形对象中删除应用程序数据名称。

H也可以是图形句柄的向量。可以提供多个应用程序数据名来一次删除多个属性。

参见:setappdata, getappdata, isappdata。

: valid = isappdata (h, name)

如果为句柄为h的图形对象存在命名的应用程序数据name，则返回true。

H也可以是图形句柄的向量。

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

**15.4.6对象组**

许多Octave高级绘图函数返回其他图形对象的组，或者它们返回的图形对象的属性以这样一种方式链接，即对其中一个属性的更改会导致其他属性的更改。将其他对象分组的图形对象称为hggroup

: hggroup ()

: hggroup (hax)

: hggroup (…, property, value, …)

: h = hggroup (…)

创建具有轴父轴的手柄图形组对象。

如果没有指定父组，则在当前轴上创建组。

可以为hg组指定多个属性/值对，但它们必须成对出现。完整的属性列表记录在Axes properties。

可选的返回值h是创建的hggroup对象的图形句柄。

编程注意:hggroup是一种将基本图形对象(如直线对象或补丁对象)分组为一个可以适当反应的单元的方法。例如，等高线的各个线条被收集到单个hg\_group中，这样它们就可以通过一个命令set (hg\_handle， "visible"， "off")设置为可见/不可见。

参见:addproperty, addlistener。

例如，hggroup的一个简单用法可能是

x = 0:0.1:10;

hg = hggroup ();

plot (x, sin (x), "color", [1, 0, 0], "parent", hg);

hold on

plot (x, cos (x), "color", [0, 1, 0], "parent", hg);

set (hg, "visible", "off");

它将两个地块分组为一个对象，并直接控制它们的可见性。hggroup的默认属性与其他图形对象的通用属性集相同。可以使用addproperty函数添加其他属性。

: addproperty (name, h, type)

: addproperty (name, h, type, arg, …)

在图形对象h中创建一个名为name的新属性。

类型确定要创建的属性的类型。Args通常包含属性的默认值，但根据属性的类型可能会给出额外的参数。

支持的属性类型有:

string

字符串属性。Arg包含默认字符串值。

any

未类型化的属性。这种属性可以保存任何八度值。Args包含默认值。

radio

具有一组有限的可接受值的字符串属性。第一个参数必须是一个字符串，所有可接受的值由竖线(' | ')分隔。默认值可以通过用'{' '}'对括起来来标记。默认值也可以作为第二个可选字符串参数给出。

boolean

布尔属性。此属性类型相当于可接受值为“on / off”的单选属性。参数包含默认属性值。

double

一个标量双精度性质。Arg包含默认值。

handle

句柄属性。这种属性包含图形对象的句柄。参数包含默认句柄值。当没有给出默认值时，属性被初始化为空矩阵。

data

数据(矩阵)属性。参数包含默认数据值。当没有给出默认值时，数据被初始化为空矩阵。

color

颜色属性。Arg包含默认颜色值。当没有给出默认颜色时，该属性被设置为黑色。可以给出可选的第二个字符串参数来指定可接受的字符串值的附加集合(如无线电属性)。

Type也可以是核心对象类型和该对象类型的有效属性名的连接。然后创建的属性具有与引用属性相同的特征(类型、可能的值、隐藏状态……)。这允许将现有属性克隆到图形对象h中。

示例：

addproperty ("my\_property", gcf, "string", "a string value");

addproperty ("my\_radio", gcf, "radio", "val\_1|val\_2|{val\_3}");

addproperty ("my\_style", gcf, "linelinestyle", "--");

参见:addlistener, hgroup。

一旦属性被添加到hg组中，它就不会链接到该组的子属性或任何其他图形对象的任何其他属性。为了控制这个新添加的属性的使用方式，addlistener函数用于定义一个回调函数，该函数在属性被更改时执行。

: addlistener (h, prop, fcn)

将fcn注册为图形对象h的属性prop的监听器。

属性侦听器在设置属性时执行(按注册顺序执行)。在执行侦听器时，新值已经可用。

Prop必须是一个字符串，在h中指定一个有效的属性。

FCN可以是函数句柄、字符串或第一个元素是函数句柄的单元格数组。如果fcn是一个函数句柄，对应的函数应该接受至少2个参数，分别设置为对象句柄和空矩阵。如果fcn是一个字符串，它必须是任何有效的八度表达式。如果fcn是一个单元格数组，则第一个元素必须是一个函数句柄，其签名与上面描述的相同。单元格数组的下一个元素作为附加参数传递给函数。

例子:

function my\_listener (h, dummy, p1)

fprintf ("my\_listener called with p1=%s\n", p1);

endfunction

addlistener (gcf, "position", {@my\_listener, "my string"})

参见:dellistener, addproperty, hggroup。

: dellistener (h, prop, fcn)

删除fcn作为图形对象h的属性prop的侦听器的注册。

函数fcn必须是与传递给addlistener的原始调用相同的变量(不仅仅是相同的值)。

如果未定义fcn，则删除prop的所有侦听器函数。

例子:

function my\_listener (h, dummy, p1)

fprintf ("my\_listener called with p1=%s\n", p1);

endfunction

c = {@my\_listener, "my string"};

addlistener (gcf, "position", c);

dellistener (gcf, "position", c);

参见:addlistener。

使用这两个函数的示例如下

x = 0:0.1:10;

hg = hggroup ();

h = plot (x, sin (x), "color", [1, 0, 0], "parent", hg);

addproperty ("linestyle", hg, "linelinestyle", get (h, "linestyle"));

addlistener (hg, "linestyle", @update\_props);

hold on

plot (x, cos (x), "color", [0, 1, 0], "parent", hg);

function update\_props (h, d)

set (get (h, "children"), "linestyle", get (h, "linestyle"));

endfunction

这将向hg组添加一个linestyle属性，并将其值的任何更改传播给该组的子元素。linkprop函数可以用来将上面的简化为

x = 0:0.1:10;

hg = hggroup ();

h1 = plot (x, sin (x), "color", [1, 0, 0], "parent", hg);

addproperty ("linestyle", hg, "linelinestyle", get (h, "linestyle"));

hold on

h2 = plot (x, cos (x), "color", [0, 1, 0], "parent", hg);

hlink = linkprop ([hg, h1, h2], "color");

: hlink = linkprop (h, "prop")

: hlink = linkprop (h, {"prop1", "prop2", …})

链接图形对象属性，以便将其中一个对象的更改传播到其他对象。

输入h是要链接的图形句柄向量。

当链接单个属性时，Prop可以是字符串，或者链接多个属性时，Prop可以是字符串的单元格数组。在链接过程中，prop中的所有属性最初将被设置为列表h中第一个对象上存在的值。

函数返回hlink，它是一个描述链接的特殊对象。只要引用链接存在，图形对象之间的链接就会处于活动状态。这意味着hlink必须保存在工作空间变量、全局变量中，或者使用setappdata或guidata等函数进行存储。要取消链接属性，请执行clear hlink。

使用linkprop的一个例子是

x = 0:0.1:10;

subplot (1,2,1);

h1 = plot (x, sin (x));

subplot (1,2,2);

h2 = plot (x, cos (x));

hlink = linkprop ([h1, h2], {"color","linestyle"});

set (h1, "color", "green");

set (h2, "linestyle", "--");

参见:linkaxes, addlistener。

: linkaxes (hax)

: linkaxes (hax, optstr)

连接2-D图的轴极限，使其中一个的变化传播到其他的。

要链接的轴句柄作为第一个参数hax传递。

可选的第二个参数是一个字符串，它定义将链接哪些轴限制。optstr的可能值有:

“x”

链接x轴

“y”

连接轴

“xy”(默认)

连接两个轴

“off”

关闭链接

如果未指定，默认是链接X轴和Y轴。

链接时，hax中第一个轴的限制应用于列表中的其他轴。对其中任何一个轴的后续更改将传播到其他轴。

参见:linkprop, addproperty。

这些功能在许多基本图形对象中使用。由Octave函数创建的hggroup对象包含一个或多个图形对象，用于:

* 将多个图形对象组合在一起，
* 在不同图形对象之间创建链接属性，和
* 从对象的实际数据中隐藏名义用户数据。

例如，茎函数创建一个茎系列，其中茎系列的每个hgroup包含两个线对象，分别表示茎的主体和头部。干系列的hg组的ydata属性表示干的头部，而干的主体位于基线和该值之间。例如

h = stem (1:4)

get (h, "xdata")

⇒ [ 1 2 3 4]'

get (get (h, "children")(1), "xdata")

⇒ [ 1 1 NaN 2 2 NaN 3 3 NaN 4 4 NaN]'

显示干系列对象的hg组的xdata与底层行之间的差异。

此类组对象的基本属性是它们由一个或多个链接的hgroup组成，并且这些组的某些属性的更改会传播给组的其他成员。然而，组成员的某些属性仅适用于当前成员。

此外，组的成员也可以通过回调函数链接到其他图形对象。例如，条或杆函数的基线是一个线对象，其长度和位置根据相应的hg组元素的变化自动调整。

**15.4.6.1对象组中的数据源**

所有组对象都包含数据源参数。有一些字符串参数包含一个表达式，当调用refreshdata函数时，计算该表达式以更新组的相关数据属性。

: refreshdata ()

: refreshdata (h)

: refreshdata (h, workspace)

计算当前图形的任何“数据源”属性，如果相应的数据发生了变化，则更新绘图。

如果第一个参数h是一个图形句柄列表，则操作这些对象，而不是gcf返回的当前图形。

第二个可选参数workspace可以接受以下值:

"base"

评估基本工作区中的数据源属性。(默认)。

"caller"

在调用refreshdata的函数的工作空间中计算数据源属性。

使用refreshdata的一个例子是:

x = 0:0.1:10;

y = sin (x);

plot (x, y, "ydatasource", "y");

for i = 1 : 100

pause (0.1);

y = sin (x + 0.1\*i);

refreshdata ();

endfor

**15.4.6.2区域系列**

区域系列对象由Area函数创建。每个hg组元素都包含一个patch对象。面积级数的性质是

basevalue

绘制面积图底面的值。

linewidth

linestyle

构成区域的补丁对象边缘的线宽和样式。参见线条样式。

edgecolor

facecolor

构成区域的补丁对象的线条和填充颜色。看到颜色。

xdata

ydata

在area函数中使用的累积求和之前，传递给area的原始数据列的x和y坐标。

xdatasource

ydatasource

数据源变量。

**15.4.6.3 Bar系列**

Bar系列对象由Bar或barh函数创建。每个hggroup元素包含一个patch对象。柱级数的性质是

showbaseline

baseline

basevalue

属性showbaseline标记是否显示柱系列的基线(默认为“on”)。表示基线的图形对象的句柄由baseline属性给出，基线的y值由basevalue属性给出。

对这些属性的任何更改都会传播到柱形图系列的其他成员和基线本身。同样，基线本身属性的变化也会传播到相应柱序列的成员。

barwidth

barlayout

horizontal

属性barwidth是与传递给bar或barh的宽度变量相对应的条的宽度。条形条系列是“分组”还是“堆叠”由barlayout属性决定，条形条是水平的还是垂直的由水平属性决定。

对这些属性的任何更改都将传播到条形图系列的其他成员。

linewidth

linestyle

构成条的补丁对象边缘的线宽和样式。参见线条样式。

edgecolor

facecolor

构成条的补丁对象的线条和填充颜色。看到颜色。

xdata

标称的杆的x位置。这一性质的变化也影响到了柱形级数的其他成员。

ydata

纵轴的y值hggroup

xdatasource

ydatasource

数据源变量。

**15.4.6.4轮廓组**

轮廓组对象由Contour、contf和contour3函数创建。它们也是surfc和meshc函数返回的句柄之一。等高线群的性质是

contourmatrix

一个只读属性，包含用于创建绘图轮廓的contourc返回的数据。

fill

可以具有值“on”或“off”的单选属性，用于标记要绘制的轮廓是否要填充。

zlevelmode

zlevel

无线电属性zlevelmode的值可以是"none"， "auto"或"manual"。当它的值为“none”时，绘制的轮廓没有z分量。当其值为“auto”时，绘制轮廓的z值与轮廓本身的值相同。如果该值为“manual”，则绘制轮廓的z值由zlevel属性决定。

levellistmode

levellist

levelstepmode

levelstep

如果levellistmode是“manual”，那么绘制等高线的水平由levellist决定。如果levellistmode设置为“auto”，那么轮廓之间的距离由levelstep决定。如果levellistmode和levelstepmode都设置为“auto”，那么假定有10个等距轮廓。

textlistmode

textlist

textstepmode

textstep

如果textlistmode为“manual”，则标记的轮廓由textlist确定。如果textlistmode设置为“auto”，则标记轮廓之间的距离由textstep决定。如果textlistmode和textstepmode都设置为“auto”，那么假定有10个等距的标记轮廓。

showtext

标记轮廓标签是否显示。

labelspacing

以点为单位的单个轮廓上标签之间的距离。

linewidth

linestyle

linecolor

等高线的性质。属性linewidth和linestyle类似于线条的相应属性。属性linecolor是一个颜色属性(参见Colors)，它的值也可以是none或auto。如果linecolor为“none”，则不绘制等高线。如果linecolor为“auto”，则线的颜色由颜色映射决定。

xdata

ydata

zdata

等高线的原始x、y、z数据。

xdatasource

ydatasource

zdatasource

数据源变量。

15.4.6.5误差条系列

错误条系列是由errorbar函数创建的。每个hggroup元素包含两个行对象，分别表示数据和错误栏。误差条级数的性质为

color

错误条的线对象的RGB颜色或颜色名称。看到颜色。

linewidth

linestyle

误差条的线对象的线宽和样式。参见线条样式。

marker

markeredgecolor

markerfacecolor

markersize

误差条上标记的线条和填充颜色。看到颜色。

xdata

ydata

ldata

udata

xldata

xudata

原始x, y, l, u, xl, xu数据的误差条。

xdatasource

ydatasource

ldatasource

udatasource

xldatasource

xudatasource

数据源变量。

**15.4.6.6线路系列**

线系列对象由plot和plot3函数创建，类型为Line。具有添加数据源能力的行系列属性。

color

线对象的RGB颜色或颜色名称。看到颜色。

linewidth

linestyle

线对象的线宽和样式。参见线条样式。

marker

markeredgecolor

markerfacecolor

markersize

标记的线条和填充颜色。看到颜色。

xdata

ydata

zdata

原始的x y和z数据。

xdatasource

ydatasource

zdatasource

数据源变量。

**15.4.6.7 Quiver Group**

Quiver系列对象由Quiver或quiver3函数创建。该系列的每个hgroup元素包含三个线对象作为子对象，分别表示箭头的主体和头部，以及一个标记作为箭头的原点。颤振级数的性质是

autoscale

autoscalefactor

标记箭头的长度是缩放还是直接从u, v和w数据定义。如果将箭头长度标记为由autoscale属性缩放，则自动缩放箭头的长度由autoscalefactor控制。

maxheadsize

此属性控制箭筒系列中箭头的大小。默认值为0.2。

showarrowhead

标记箭头是否显示在箭袋图中。

color

箭筒的线条对象的RGB颜色或颜色名称。看到颜色。

linewidth

linestyle

线的宽度和样式的线的对象的颤抖。参见线条样式。

marker

markerfacecolor

markersize

标记对象的线条和填充颜色位于箭头的原始位置。看到颜色。

xdata

ydata

zdata

向量场值的原点。

udata

vdata

wdata

要绘制的向量场的值。

xdatasource

ydatasource

zdatasource

udatasource

vdatasource

wdatasource

数据源变量。

**15.4.6.8楼梯组**

楼梯系列对象由Stair函数创建。该系列的每个hg组元素都包含一个单线对象，作为表示楼梯的子对象。阶梯级数的性质是

color

楼梯的线条对象的RGB颜色或颜色名称。看到颜色。

linewidth

linestyle

楼梯线条对象的线条宽度和样式。参见线条样式。

marker

markeredgecolor

markerfacecolor

markersize

楼梯上标记的线条和填充颜色。看到颜色。

xdata

ydata

楼梯的原始x和y数据。

xdatasource

ydatasource

数据源变量。

**15.4.6.9阀杆系列**

Stem系列对象由Stem或stem3函数创建。每个hg - group元素都包含一个单行对象，作为表示茎的子对象。该级数的性质为

showbaseline

baseline

basevalue

属性showbaseline标记是否显示茎系列的基线(默认为“on”)。表示基线的图形对象的句柄由baseline属性给出，基线的y值(或stem3的z值)由basevalue属性给出。

对这些属性的任何更改都将传播到干系列的其他成员和基线本身。同样地，基线本身性质的变化也会传播到相应的干级数的成员上。

color

茎的行对象的RGB颜色或色名。看到颜色。

linewidth

linestyle

茎的线对象的线宽度和样式。参见线条样式。

marker

markeredgecolor

markerfacecolor

markersize

茎上标记的线条和填充颜色。看到颜色。

xdata

ydata

zdata

茎的原始x, y和z数据。

xdatasource

ydatasource

zdatasource

数据源变量。

**15.4.6.10表面组**

表面组对象是由surf或mesh函数创建的，但同样是由surf或mesh函数返回的句柄之一。表面群属于表面类型。

表面基团的性质为

edgecolor

facecolor

表面的边或面的RGB颜色或色名。看到颜色。

linewidth

linestyle

表面上线条的线宽和样式。参见线条样式。

marker

markeredgecolor

markerfacecolor

markersize

表面上标记的线条和填充颜色。看到颜色。

xdata

ydata

zdata

cdata

原始的x y z和c数据。

xdatasource

ydatasource

zdatasource

cdatasource

数据源变量。

**15.4.7变换组**

: h = hgtransform ()

: h = hgtransform (property, value, …)

: h = hgtransform (hax, …)

创建一个图形转换对象。

修复:需要写文档。修正:添加<makehgtform>来查看also列表，当它被实现。

参见:hggroup。

**15.4.8图形工具箱**

: tkit = graphics\_toolkit ()

: tkit = graphics\_toolkit (hlist)

: graphics\_toolkit (name)

: graphics\_toolkit (hlist, name)

查询或设置分配给新图形的默认图形工具包。

如果没有输入，则返回当前默认图形工具包。如果输入是图形句柄列表hlist，则返回用于每个图形的图形工具包的名称。

当使用单个输入名称调用时，将默认图形工具包设置为name。如果工具包尚未加载，则通过调用\_\_init\_name函数对其进行初始化。如果第一个输入是图形句柄列表hlist，那么图形工具包将被设置为仅为这些图形命名。

参见:available\_graphics\_toolkits。

: toolkits = available\_graphics\_toolkits ()

返回已注册图形工具包的单元数组。

参见:graphics\_toolkit, register\_graphics\_toolkit。

: toolkits = loaded\_graphics\_toolkits ()

返回当前加载的图形工具包的单元格数组。

参见:available\_graphics\_toolkits。

: register\_graphics\_toolkit ("toolkit")

将工具包列为可用的图形工具包。

编程注意:对输入字符串不进行输入验证;它只是添加到可能的图形工具包列表中。

参见:available\_graphics\_toolkits。

**15.4.8.1定制Toolkit行为**

可以使用以下实用程序函数修改后端工具包的特定行为。注意:并非所有函数都适用于每个图形工具包。

: [prog, args] = gnuplot\_binary ()

: [old\_prog, old\_args] = gnuplot\_binary (new\_prog)

: [old\_prog, old\_args] = gnuplot\_binary (new\_prog, arg1, …)

查询或设置图形工具箱设置为“gnuplot”时，plot命令调用的程序名称。

还可以给出传递给外部绘图程序的其他参数。默认值是“gnuplot”，不带其他参数。参见安装Octave。

参见:graphics\_toolkit。

此外，gnuplot程序通常提供许多不同的接口，称为终端。Octave通常会选择一个默认终端，但是您可以使用环境变量GNUTERM来覆盖它。这个变量可以在启动Octave之前在shell中设置，也可以在第一次绘图之前在Octave中设置。例如:

setenv ("GNUTERM", "wxt")

graphics\_toolkit ("gnuplot")

plot (1:10)

**15.4.8.2硬件与软件渲染**

当使用Octave for Windows安装程序时，用户可以选择“系统OpenGL”和“软件OpenGL”渲染器。此选项选择是否对OpenGL图形工具包(“qt”和“fltk”)使用软件渲染。软件渲染可以用来避免由于来自不同厂商的不同图形卡(特别是集成的英特尔图形卡)的不完善的OpenGL驱动实现而导致的渲染和打印问题。缺点是，软件渲染可能比硬件加速渲染慢得多(并且在32位平台或WoW64上可能无法正常工作)。要在显卡驱动的硬件加速渲染和软件渲染之间永久切换，请在Octave关闭时从开始菜单中使用“OpenGL切换器”应用程序。或者，在Octave关闭时重命名以下文件:

octave-home\bin\opengl32.dll

其中Octave -home是OCTAVE＿HOME返回的目录，即安装Octave的目录(默认为C: \Program Files \GNU Octave \Octave\Octave -version \mingw 64)。将文件扩展名更改为。bak(用于硬件呈现)或。dll(用于软件呈现)。

**15.4.8.3精度问题**

OpenGL图形工具包(“qt”和“fltk”)使用单精度进行渲染。这种限制特别适用于datenum、datestr、datestruct和datetick函数使用的时间序列与序列日期的对比图。

序列日期将时间戳编码为从零年开始经过的天数，以小时、分钟、秒为小数部分。在1999年12月31日，序列号日期表示为730485。带有整数部分的双精度变量允许其小数部分的分辨率为1.2e-10，表示大约5微秒。但在单精度情况下，分辨率降低到0.06左右，代表45分钟。任何用更细粒度绘制时间戳数据的尝试都会导致扭曲的图形。

作为一种变通方法，可以使用“gnuplot”图形工具包或减去2000年。， datenum(2000,0,0)或730485 -从时间值。由于日历结构每2000年重复一次，因此年、月、月中的一天和星期中的一天之间的关系将保持不变，datetick函数产生的刻度和ticklabel仍然是正确的。只有年份没有千禧年数字。因此，“2020”将被打印为“20”。例如:

# timestamps of 24 hours in one minute steps

t = datenum (2020, 1, 1):(1/1440):datenum (2020, 1, 2);

# some example time series data

x = -cos (2\*pi\*t) + rand (size (t)) / 10;

subplot (1, 2, 1);

plot (t, x);

datetick ("x");

xlabel ("serial date");

title ("problem");

subplot (1, 2, 2);

plot (t - 730485, x);

datetick ("x");

xlabel ("2000 years off");

title ("workaround");

其结果如图15.8所示。

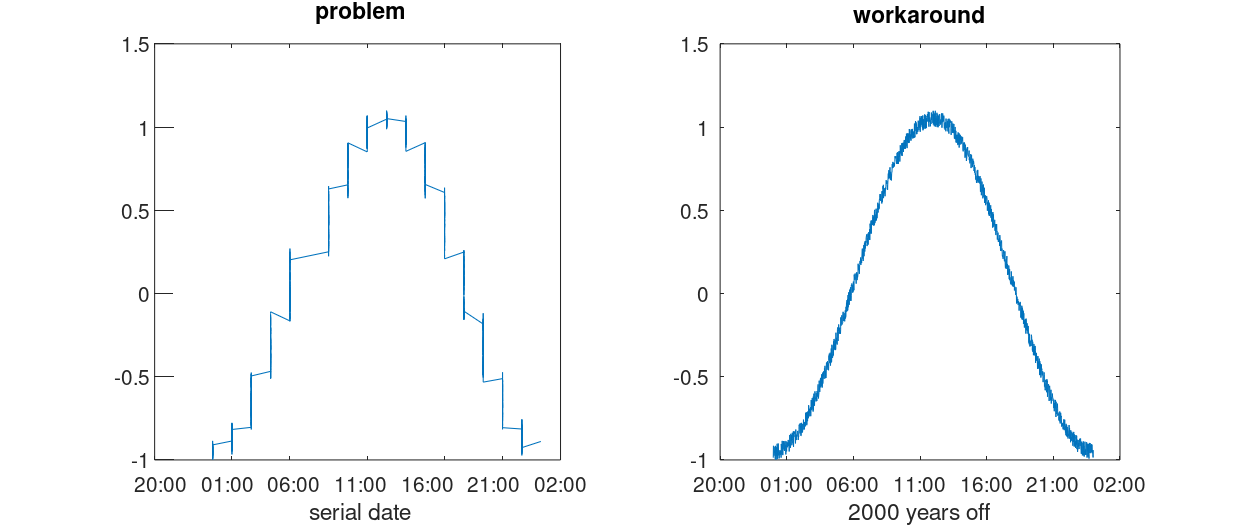


图15.8:OpenGL图形工具包的单精度问题

同样，可以翻译或重新缩放其他数据以解决此问题。