GNU Octave

A high-level interactive language for numerical computations
数値計算のための高水準対話的言語
Edition 3 for Octave version 2.1.x
February 1997

John W. Eaton

これは Octave ドキュメントの第 3 版であり，Octave のバージョン 2.1.x について一貫しています。

目次

まえがき

Octave は，もともと化学反応装置における学部レベルの教科書と共に使用するためのソフトウエアを意図して，Wisconsin-Madison 大学の James B. Rawlings と Texas 大学の G. Ekerdt によって書かれました。

明らかに，Octave は授業の枠を越え，現在では単なる低機能な「授業用」パッケージよりもはるかに多くの機能を持っています。我々の最初の目標がいくぶん漠然としたものであったにもかかわらず，我々は，学生たちに現実的な問題を解くことができ，化学反応装置問題以外の多くのことに対して使えるものを作りたいということは分かっていました。

工学向けコンピュータ言語であるという理由から，学生には Fortran を教えるべきだということを言う人もいるでしょう。しかし，我々がそれを試みたときにはいつでも，学生たちは，なぜ自作の Fortran コードがクラッシュするかを解明することにとても多くの時間を割いてしまい，化学工学について学ぶ時間がありませんでした。Octave では，大部分の学生たちはすぐに基礎を身につけ，ものの数時間でそれを自信たっぷりに使っています。

このソフトウエアは，本来は反応装置学を教えるために使用することを意図していましたが，Texas 大学の Chemical Engineering 学部では，学部および大学院の他の講義において使用されています。そして，Texas 大学の数学学部では，微分方程式および線形代数を教えるためによく使用しています。もし，このソフトウエアが役立つと思うならば，どうぞお知らせください。我々は，他の場所でどのように Octave が使用されているかを知ることに，いつも興味を持っているのです。

ほとんどの人は，Octave という名前は音楽に何か関係があると思うでしょう。しかし，実のところは，化学反応工学の著名なテキストを書き,「封筒の裏」を使って高速に計算を行う能力をもつことでよく知られた，私の前の教授の名前です。このソフトウエアが，多くの人々にとって，大がかりな計算をより簡単に実行できるようになることを望んでいます。

皆さんには，このマニュアルの先頭で述べたように，GNU General Public License（see 付記 F [Copying], 頁 329）の下で，このソフトウエアを他の人々と共有することをおすすめします。また，Octave をより有用なものにするために，新たな関数を書き下ろして提供したり，何らかの問題があったときにはそれを報告してみてください。

謝辞

すでに多くの人々が Octave の開発に貢献してくれています。John W. Eaton に加えて，以下の方々が Octave の一部を書くのを手伝ってくれたり，さまざまな点について援助してくださいました。

- Thomas Baier `baier@ci.tuwien.ac.at` は，`popen`，`pclose`，`execute`，`sync_system`，および `async_system`の最初のバージョンを書いてくれました。
- David Bateman `dbateman@free.fr` は，sort および min/max 関数を改良し，多くの関数を N 次元化し，いくつかの組み込み関数を Linpack の代わりに Lapack を使うように変換し，‘`load-save.cc`’の機能を `octave_value` に分割し，そして他の多くの点について貢献してくれました。
- Karl Berry `karl@cs.umb.edu` は，Octave が関数およびスクリプトファイルをディレクトリから再帰的に検索できるようにするための `kpathsea` ライブラリを書いてくれました。
- Georg Beyerle `gbeyerle@awi-potsdam.de` は，MATLAB の ‘`.mat`’形式のファイルに保存するためのコードを寄与してくれ，多くの有効なバグ報告と提案を行ってくれました。
- John Campbell `jcc@bevo.che.wisc.edu` は，大部分のファイルの入出力および C 言語形式の入出力関数を書いてくれました。

- Dirk Eddelbuettel edd@debian.org は，Octave を Debian GNU/Linux のユーザにとってインストールしやすくしてくれました。
- Brian Fox bfox@gnu.org は，コマンドライン編集のために使用されている readlineライブラリを書いてくれ，このマニュアルのそれに関する部分を書いてくれました。
- Klaus Gebhardt gebhardt@crunch.ikp.physik.th-darmstadt.de は，Octave を OS/2 に移植してくれました。
- Kai Habel kai.habel@gmx.de は，座標変換を実行するための関数を実装してくれました。
- A. Scottedward Hodel A.S.Hodel@eng.auburn.edu は，expm，qzval，qzhess，syl，lyapおよび balance を含む数多くの関数に貢献してくれました。
- Kurt Hornik Kurt.Hornik@ci.tuwien.ac.at は，corrcoef，cov，fftconv，fftfilt，gcd, lcd，kurtosis，null，orth，poly，polyfit，rootsおよび skewness関数を提供し，これら関数やその他の多くの関数に対してドキュメントを提供し，Octave コードを編集するための Emacs モードとそのドキュメントを書き直し，そしてものすごくテストを手伝ってくれました。彼は，Octave を改良するための新たなアイディアのもとを持っていてくれました。
- Cai Jianming caijianming@yahoo.co.uk initial cell array の実装に貢献してくれました。
- Phil Johnson johnsonp@nicco.sscnet.ucla.edu は，Linux 版が入手できるように手伝ってくれました。
- Steven G. Johnson stevenj@alum.mit.edu は，ATLAS，HDF5 形式のファイルへの保存をサポートし，Octave の configure スクリプトを Autoconf 2.50 に移植してくれました。
- Mumit Khan khan@nanotech.wisc.edu は，Octave を，GCC の他の ISO 標準に準拠した C++コンパイラでコンパイルできるようにすることを助けてくれました。
- Paul Kienzle pkienzle@users.sf.net は，Octave と MATLAB との互換性を向上させるための多くの機能を充実させてくれました。また，http://octave.sf.netにおいて，貢献されたコードのコレクションをメンテナンスしてくれています。
- Bill Lash lash@tellabs.com は，unwrap関数を提供してくれました。
- Dirk Laurie dlaurie@na-net.ornl.gov は，invhilbを，より速くより正確になるように書き直してくれました。
- Friedrich Leisch leisch@ci.tuwien.ac.at は，mahalanobis 関数を提供してくれました。
- Ken Neighbors wkn@leland.stanford.edu は，多くの有用なバグ報告と MATLAB との互換性についてのコメントを寄せてくれました。
- Rick Niles niles@axp745.gsfc.nasa.gov は，Octave のプロット関数に，線スタイルと一回の呼び出し当たりの無制限の線の数を特定する能力を追加するために書き換えてくれました。彼は，奇妙な非互換挙動およびバグを報告し続けてくれています。
- Mark Odegard meo@getech.com は，fread，fwrite，feofおよび ferror関数を最初に実装してくれました。
- Gabriele Pannocchia pannocchia@ing.unipi.it は，dkalman.m 関数を提供してくれ，dlqeと dlqrに singular system matrices へのサポートを追加してくれ，制御システム関数群に多くの改善を行ってくれました。
- Tony Richardson richardson@evansville.edu は，最初の多項式関数群の大多数ばかりでなく，Octave の画像処理関数を書いてくれました。
- Petter Risholm Petter.Risholm@idi.ntnu.no は，Octave の N 次元配列機能の多くを実装することを助けてくれました。

- Ben Sapp bsapp@lanl.gov は，デバッガ関数を実装してくれ，内部ドキュメント文字列にTexinfo マークアップコマンドを追加してくれました。
- R. Bruce Tenison btenison@rstc.cc.al.us は，hessと schur関数を書いてくれました。
- Teresa Twaroch twaroch@ci.tuwien.ac.at は，glsと ols 関数を提供してくれました。
- James R. Van Zandt jrv@vanzandt.mv.com は，MATLAB バージョン 5 データファイルの読み書きについてのサポートを追加してくれました。
- Andreas Weingessel Andreas.Weingessel@ci.tuwien.ac.at は，lin2mu，loadaudio，mu2lin，playaudio，record，saveaudioおよび setaudioなるオーディオ関数を書いてくれました。
- Fook Fah Yap ffy@eng.cam.ac.uk は，fftと ifft 関数を提供してくれ，初期バージョンにおいて価値あるバグ報告をしてくれました。

Octave の開発を援助してくださったことについて，以下の方々ならびに組織・団体に特にお礼申し上げます。

- The National Science Foundation, through grant numbers CTS-0105360, CTS-9708497, CTS-9311420, and CTS-8957123.
- The industrial members of the Texas-Wisconsin Modeling and Control Consortium (TWMCC (http://www.che.utexas.edu/twmcc)).
- The Paul A. Elfers Endowed Chair in Chemical Engineering at the University of Wisconsin-Madison.
- Digital Equipment Corporation, for an equipment grant as part of their External Research Program.
- Sun Microsystems, Inc., for an Academic Equipment grant.
- International Business Machines, Inc., for providing equipment as part of a grant to the University of Texas College of Engineering.
- Texaco Chemical Company, for providing funding to continue the development of this software.
- The University of Texas College of Engineering, for providing a Challenge for Excellence Research Supplement, and for providing an Academic Development Funds grant.
- The State of Texas, for providing funding through the Texas Advanced Technology Program under Grant No. 003658-078.
- Noel Bell, Senior Engineer, Texaco Chemical Company, Austin Texas.
- James B. Rawlings, Professor, University of Wisconsin-Madison, Department of Chemical Engineering.
- Richard Stallman, for writing GNU.

このプロジェクトは，Octave において使用されている，そして Octave を開発するために使用した GNU ソフトウエアなくしては実現しなかったでしょう。

Octave に寄与するには

Octave をよりよいシステムにするために，あなたが貢献できる多くの方法があります。おそらく，貢献するための最も重要な方法とは，新たな問題を解決するための高品質のコードを書き，それを他の人が自由に利用できるようにすることです。

Octave が役立つと思うならば，開発を継続するために追加の資金援助を考えてみてください。わずかな追加資金援助であっても，開発とサポートに割くことのできる時間が大幅に増えました。

資金援助あるいはコードを寄与することができなくても，Octave をより良く信頼性を高めるために，発見したバグを報告したり，Octave を進歩させるための方法について提案を行うことはできます。良いバグ報告の書き方については，付記 B [Trouble], 頁 307 を参照してください。

配布

Octave はフリーソフトウエア（*free* software）です。この意味とは，誰もがフリーに使用でき，ある条件下において再配布することもフリーということです。Octave はパブリック・ドメインではありません。著作権は明記されていますし，再配布にも制限があります。しかし，その制限とは，他の人も Octave の使用と再配布に対して，あなたと同じ自由を持つことができるようにすることです。詳細な条件については，Octave に付属している GNU General Public License をご覧ください。これは付記 F [Copying], 頁 329 にも掲載しています。

Octave は，他のさまざまなフリーソフトウエア集とともに，Free Software Foundation から CD-ROM として入手することができます。Free Software Foundation に Octave のコピーを注文することにより，多くのフリーソフトウエアの開発に資金援助することになります。もっと情報を得るには，以下の住所に連絡をとってください。

Free Software Foundation
51 Franklin Street, Fifth Floor
Boston, MA 02110-1301–1307
USA

Octaveはインターネットからも入手できます。URL は`ftp://ftp.che.wisc.edu/pub/octave`です。さらなる情報は，`http://www.che.wisc.edu/octave`から入手してください。

1 Octave の簡単な紹介

このマニュアルは，GNU Octave の実行，インストール，移植およびバグ報告の方法について記したものです。

GNU Octave は高水準言語であり，主に数値計算向けの言語です。このソフトウエアは，線形ならびに非線形問題を数値的に解き，他の数値実験を実行するための便利なコマンドラインインターフェースを備えています。バッチ指向の言語として使用することもできます。

GNU Octave は自由に再配布できるソフトウエアでもあります。あなたは，Free Software Foundation によって公開されている GNU General Public License の項の下で，このソフトウエア単体，あるいは改変したものを再配布してもかまいません。GPL は，このマニュアルの 付記 F [Copying], 頁 329 に掲載してあります。

このドキュメントは，Octave のバージョン 2.1.x 向けに書いてあります。

1.1 Octave の実行

多くのシステムにおいて，Octave を起動する方法は，シェルから ‘`octave`’ とコマンドを入力することです。Octave は起動時メッセージを表示し，入力の受け付け準備ができたことを示すプロンプトを表示します。その後，すぐに Octave コマンドの入力を開始することができます。

もしトラブルが発生したなら，通常はキーボードから *Control-C*（以降は短縮して *C-c* と表記します）と入力することにより，Octave を中断することができます。*C-c* とは，見ての通り，〈CTRL〉を押しながら 〈c〉 を押すことを表します。これを実行すると，Octave のプロンプトに問題なく戻るでしょう。

Octave を終了するには，Octave のプロンプトから *quit*あるいは *exit* と打ち込みます。

ジョブコントロールをサポートするシステムにおいては，通常は *C-z* と入力して SIGTSTPシグナルを送ることにより，Octave をサスペンドすることができます。

1.2 簡単な例

以降の章においては，Octave の全機能について詳細に記述してあります。しかし，その前に，その機能のいくつかについて例を示すことが役立つでしょう。

もし Octave に初めて触れるなら，Octave を学び始めるために，ここに示す例を試すことをお勧めします。‘`octave:13>`’ のように示した行は，実際に打ち込む行を表し，入力後に改行キーを押します。Octave は答えを返すか，グラフを表示します。

行列をつくる

新しい行列を作成して変数に格納しておけば，それを後から参照することができます。以下のコマンドを入力してください。

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

Octave は，きちんと列をそろえて，行列を表示してくれるでしょう。コマンドの終端にセミコロンをつけると，結果を画面に表示しない命令になります。たとえば，

```
octave:2> b = rand (3, 2);
```

このコマンドは，3 行 2 列の行列を作るものです。その各要素には，0 から 1 までの範囲のランダムな数値がセットされます。

任意の変数の値を表示するためには，単に変数名を入力するだけです。たとえば，行列 `b` に格納された行列を表示するためには，以下のように入力してください。

```
octave:3> b
```

行列

Octave には，行列演算を行うための便利な演算子表記ができます。たとえば，行列 aにスカラを乗じるには，以下のコマンドを入力してください。

```
octave:4> 2 * a
```

ふたつの行列 a と b を乗じるには，以下のコマンドを入力してください。

```
octave:5> a * b
```

積 $\mathtt{a}^T\mathtt{a}$, を計算するためには，以下のコマンドを入力してください。

```
octave:6> a' * a
```

一次方程式を解く

連立一次方程式 ax = b を解くためには，左除算演算子 '\' を使用します：

```
octave:7> a \ b
```

これは，概念的には $\mathtt{a}^{-1}\mathtt{b}$, とすることと等しいです。しかし，左除算演算子を使用することによって，逆行列を計算することを避けます。

もし係数行列が特異ならば，Octave は警告メッセージを表示し，最小ノルム解を表示します。

微分方程式の積分

Octave には，以下の形式の非線形微分方程式を解くための組み込み関数があります。

$$\frac{dx}{dt} = f(x,t), \qquad x(t=t_0) = x_0$$

です。Octave にて，この形式の方程式を積分するためには，最初に関数の定義 $f(x,t)$ を提供しなければなりません。これは簡単なことで，関数の本体をコマンドラインから入力することで遂行できます。たとえば，以下のコマンドは，非線形微分方程式の面白いペアについて，右辺の関数を定義しています。関数を入力している間は，入力が完成するまで待っていることを示す異なるプロンプトになることに注意してください。

```
octave:8> 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
```

初期条件を入力しておきます。

```
x0 = [1; 2];
```

そして，出力時間を列ベクトルとして入力します（最初の出力時間は，上で入力した初期条件に対応したものであることに注意してください）。

```
t = linspace (0, 50, 200)';
```

微分方程式の組を積分することは簡単です：

```
x = lsode ("f", x0, t);
```

この関数 `lsode` は通常微分方程式に対する Livermore Solver を使用しています。これは，以下の文献に記述されています: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.

グラフ出力の実行

前の例の解を，グラフで表示するためには，コマンド

```
plot (t, x)
```

を使用します。

もし X Window System を使用しているならば，Ocatve は，プロットを表示するために自動的に別ウインドウを開きます。もし他のグラフィック関係のコマンドをサポートする端末を使用しているならば，使用している端末を Octave に教えてあげる必要があります。サポートしている端末の種類のリストを見るには，以下のコマンドを入力してください。

```
gset term
```

Octave は，グラフを表示するために `gnuplot` を使用します。そして，`gnuplot` がサポートする任意の端末において，画像を出力することができます。

plot コマンドの出力を，直接端末に送らずにファイルに取り込むには，以下のような一連のコマンドを使用することができます。

```
gset term postscript
gset output "foo.ps"
replot
```

これは，他の出力デバイスについても同様に働くでしょう。Octave の `gset` コマンドは，現実には単に `gnuplot` サブプロセスにパイプしているだけです。その結果，いちど好みのプロットをスクリーンに表示すると，画像を印刷するのに適した出力ファイルを作成するため，これと同様の方法を実行することができるでしょう。

あるいは，以下のようなコマンドを使うことにより，中間ファイルを削除することができます。

```
gset term postscript
gset output "|lpr -Pname_of_your_graphics_printer"
replot
```

入力した内容の編集

Octave のプロンプトでは，Emacs あるいは vi スタイルの編集コマンドにより，以前に入力したコマンドの再呼び出し，編集，再実行ができます。標準状態のキー割り当てでは，Emacs スタイルのコマンドを使用します。たとえば，以前のコマンドを呼び出すには，*`Control-p`*（以降は，単に *`C-p`* と表記します）と打ち込みます。*`C-p`* とは，見ての通り，⟨CTRL⟩ を押しながら ⟨p⟩ を押すことを表します。これを行うことにより，以前の入力行を問題なく呼び戻すことになるでしょう。*`C-n`* は，次の入力行に進めることになります。*`C-b`* はカーソルを行末に移動させ，*`C-f`* はカーソルを行頭に向けて移動させます。このほかにも，いくつもキーが定義されています。

コマンドライン編集機能についての完全な記述は，このマニュアルの Section 2.4 [Command Line Editing], 頁 15 にあります。

ヘルプとドキュメント

Octave は，豊富なヘルプ機能をもっています。印刷物として入手できるドキュメントは，Octave のプロンプトからも読むことができます。これは，両方の文書形式とも同じ入力ファイルから作成されているからです。

よいヘルプを得るためには，まず，あなたが使いたいコマンドの名前を知る必要があります。関数の名前は，必ずしも明らかではないかもしれません。しかし，始めるための良い方法は，単に `help` と入力することです。このコマンドは，すべての演算子，予約語，関数，組み込み変数および関数ファイルを表示するものです。help コマンドの引数に名前を含めることによって，より多くのヘルプを得ることができます。たとえば，

```
help plot
```

これは，`plot` 関数に対するヘルプ文章を表示します。

Octave では，長すぎて一画面に収まらない出力は，`less` や `more` のようなページャに送ります。1 行進めるには ⟨RET⟩，1 ページ進めるには⟨SPC⟩，ページャを抜けるには ⓠ を押します。

Octave 内から印刷マニュアルの完全な文章を読めるような，Octave のヘルプ機能の一部は，Info と呼ばれる別のプログラムを使用します。Info を呼び出すとき，Octave のマニュアルを含むメニュー型プログラムに入ります。Info を使用するためのヘルプは，このマニュアルの Section 2.3 [Getting Help], 頁 14 に記述してあります。

1.3 表記法

この節では，本マニュアルで使用されている表記方法について説明します。この節は読み飛ばしてもかまいませんし，後で参照し直してもよいでしょう。

1.3.1 フォント

Octave コードの例は，`svd (a)` のようなフォントまたは形式で表します。引数や説明用の変数を表すときは，*first-number* のようなフォントまたは形式で表します。シェルのプロンプトに打ち込むコマンドは，‘`octave --no-init-file`’ のようなフォントまたは形式で表します。Octave のプロンプトに打ち込むコマンドは，*`foo --bar --baz`* のようなフォントまたは形式で表します。キーボード上の特定のキーを表すには，⟨ANY⟩ のような形式です。

1.3.2 命令結果の表記

このマニュアルの例において，あなたが命令した結果は，‘⇒’ で示します。たとえば，

```
sqrt (2)
     ⇒ 1.4142
```

となります。これは,「`sqrt (2)` は 1.4142 と計算された」と読みます。

場合によっては，条件式によって返される行列の値は，以下のように表示されます。

```
[1, 2; 3, 4] == [1, 3; 2, 4]
     ⇒ [ 1, 0; 0, 1 ]
```

その他の場合には，行列は以下のように表示されます。

```
eye (3)
     ⇒  1  0  0
        0  1  0
        0  0  1
```

これは，結果の構造をはっきりと示すためです。

ときどき，ある表現を記述する手助けのため，同じ結果を返す別の表記をすることがあります。全く等価な表現は，‘≡’ で示します。たとえば，以下のようになります。

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

1.3.3 出力の表記

このマニュアルに掲載した例の多くは，式が評価されたときに表示される文字列を表示しています。このマニュアルにある例は，出力された文字列を ‘⊣’ で示しています。式を評価するときに返される値（以下の例では 1）は，別の行に ‘⇒’ で表示しています。

```
printf ("foo %s\n", "bar")
     ⊣ foo bar
     ⇒ 1
```

1.3.4 エラーメッセージ

中にはエラーを返す例もあります。これは，通常はエラーメッセージを端末に表示します。エラーメッセージは，`error:` から始まる行で示します。

```
struct_elements ([1, 2; 3, 4])
error: struct_elements: wrong type argument ‘matrix’
```

1.3.5 解説のフォーマット

関数，コマンドおよび変数の解説は，このマニュアルでは同一の書式で記述しています。解説の最初の行は名前（関数名，コマンド名および変数名）に続いて，必要に応じて引数が含まれます。% The category—function, variable, or whatever—is printed next to the % right margin. カテゴリ—関数，変数などのようなもの—は，右端に表示します。次の行以降に解説が続き，ときどき例も続きます。

1.3.5.1 簡単な関数の解説

関数の解説において，関数名が最初に表れます。同じ行には，引数のリストが続いています。引数の名前は，解説中においても使用しています。

架空の関数 `foo` の解説例です：

`foo` (*x*, *y*, ...) [Function]

関数 `foo` は，*y* から *x* を減じ，その結果に残りの引数を加算します。もし *y* が与えられなければ，かわりに標準状態で 19 を使用します。

```
foo (1, [3, 5], 3, 9)
     ⇒ [ 14, 16 ]
foo (5)
     ⇒ 14
```

より一般には，以下のようになる。

```
foo (w, x, y, ...)
≡
x - w + y + ...
```

その名前に型の名称を含む引数（たとえば *integer*，*integer1* あるいは *matrix*）は，その型であることを期待しています。*object* と名付けた引数は，何らかの型となります。その他の種類の名前（たとえば *new_file*）は，関数の解説において具体的に論じています。ある節の中には，いくつかの関数に共通な引数の機能について，解説の最初に述べています。

Octave の関数は，いくつかの方式によって定義されています。関数についてのカテゴリ名は，関数が定義された方法を表す名称を表しています。これら付加的な情報は，以下のようなものです。

Built-in Function
: この表記のある関数は，C++，C あるいは Fortran 言語で書かれていて，コンパイル済みの Octave バイナリの一部です。

Loadable Function
: この表記のある関数は，C++，C あるいは Fortran 言語で書かれています。ユーザが供給した関数のダイナミックリンクをサポートするシステムでは，Octave を実行している間に，必要なときにだけ自動的にリンクされます。Section 13.8 [Dynamically Linked Functions], 頁 95 を参照してください。

Function File
: この表記のある関数は，Octave コマンドを使用して書かれていおり，テキストファイルに保存してあります。Section 13.6 [Function Files], 頁 92 を参照してください。

Mapping Function
: この表記のある関数は，引数として与えた行列およびベクトルの，各要素に対してそれぞれ処理を行います。

1.3.5.2 簡単なコマンドの解説

コマンドの解説は，関数の説明と似た形式です。ただ，‘関数’ が ‘コマンド’ に置きかわっれいるだけです。コマンドとは，引数をかっこでくくらないで呼び出す関数です。たとえば，Octave の `cd` コマンドについての解説を示します。

`cd` *dir* [Command]
`chdir` *dir* [Command]
: 現在の作業ディレクトリを *dir* に変更します。たとえば，`cd ~/octave` は，現在の作業ディレクトリを ‘`~/octave`’ に移動します。もし，そのディレクトリが存在しないならば，エラーメッセージを表示し，作業ディレクトリは変更されません。

1.3.5.3 簡単な変数の解説

変数とは，値を保持することのできる名前です。任意の変数は，ユーザによってセットすることができますが，ユーザが Octave の挙動を変更することができるようにするために，形式的に組み込み変数（*built-in variables*）が存在します（組み込み変数は，ときどき ユーザオプションとも呼ばれます）。通常の変数および組み込み変数は，引数が無いことを除いて，関数と同様の解説を行っています。

架空の組み込み変数 `do_what_i_mean_not_what_i_say`. についての解説例を示します。

`do_what_i_mean_not_what_i_say` [Built-in Variable]
: もしこの変数がゼロでないならば，たとえあなたが完全に間違っていて意味のないコマンドリストを入力したとしても，Octave はあなたが本当にやりたかったことを実行します。

他の変数の解説は，同様の書式をとっています。しかし，‘組み込み変数’ は通常の ‘変数’，あるいは変更されない ‘定数’ に置き換えられます。

2 始めましょう

この章では，Octave の基本機能，のいくつかについて説明しています。これには，Octave セッションを開始する方法，コマンドプロンプトでヘルプを得る方法，コマンドラインの編集のしかた，およびシェルからコマンドとして実行するための Octave プログラムを書く方法が含まれています。

2.1 Octave の起動

通常は，何も引数を与えずに ‘octave’ を実行することにより，Octave は対話的に使用することができます。一度開始してしまえば，終了の命令を与えるまで，Octave は端末からコマンドを読み込みます。

コマンドラインでファイル名を指定することもできます。そのとき，Octave はそのファイルを読み込んで実行し，末尾まで読み込んだ後に終了します。

次の節で解説してあるコマンドラインオプションを使用することにより，どのように Octave を起動するかを，さらにコントロールすることができます。

2.1.1 コマンドラインオプション

Octave が受け入れるすべてのコマンドラインオプションの完全な一覧を示します。

`--debug`
`-d`
: パーサデバッグモードに入ります。このオプションを使うと，Octave のパーサが，読み込んだコマンドについて多くの情報を表示するようになります。これは，おそらく実際にパーサのデバッグを試みるときにのみ有用でしょう。

`--echo-commands`
`-x`
: コマンドが実行されるときに，そのコマンドを表示します。

`--exec-path` *path*
: 実行すべきプログラムを検索するパスを指定します。コマンドラインで指定した *path* の値は，環境変数 `OCTAVE_EXEC_PATH` の値を上書きすることになります。しかし，組み込み変数 `EXEC_PATH` をセットしているシステムあるいはユーザのスタートアップファイル内のコマンドには影響しません。

`--help`
`-h`
`-?`
: 短いヘルプメッセージを表示して終了します。

`--info-file` *filename*
: 使用すべき info ファイルの名前を指定します。コマンドラインで指定した *filename* の値は，環境変数 `OCTAVE_INFO_FILE` の値を上書きすることになります。しかし，組み込み変数 `INFO_FILE` をセットしているシステムあるいはユーザのスタートアップファイル内のコマンドには影響しません。

`--info-program` *program*
: 使用すべき info プログラムの名前を指定します。コマンドラインで指定した *program* の値は，環境変数 `OCTAVE_INFO_PROGRAM` の値を上書きすることになります。しかし，組み込み変数 `INFO_PROGRAM` をセットしているシステムあるいはユーザのスタートアップファイル内のコマンドには影響しません。

`--interactive`
`-i` 強制的に対話的挙動をするようにします。これは，リモートシェルコマンドあるいはEmacsのシェルバッファを通してOctaveを実行する際に有用です。Emacs内でOctaveを実行するための別の方法は，付記 D [Emacs], 頁 319 を参照してください。

`--no-history`
`-H` コマンドライン履歴を使用できなくします。

`--no-init-file`
ファイル‘`~/.octaverc`’あるいは‘`.octaverc`’を読みません。

`--no-line-editing`
コマンドライン編集を不可にします。

`--no-site-file`
site-wide ‘`octaverc`’ ファイルを読みません。

`--norc`
`-f` 起動時に一切のシステムあるいはユーザ初期化ファイルを読みません。これは，`--no-init-file`と`--no-site-file`の両方のオプションを指定したものと等価です。

`--path` *path*
`-p` *path* 関数ファイルの検索パスを指定します。コマンドラインで指定した *path* の値は，環境変数 `OCTAVE_PATH` の値を上書きすることになります。しかし，組み込み変数 `LOADPATH` をセットしているシステムあるいはユーザのスタートアップファイル内のコマンドには影響しません。

`--silent`
`--quiet`
`-q` 起動時のメッセージとバージョン情報を表示しません。

`--traditional`
`--braindead`
MATLAB との互換性を保つため，ユーザ設定変数の初期値として，以下の値をセットします。

```
PS1                          = ">> "
PS2                          = ""
beep_on_error                = true
crash_dumps_octave_core      = false
default_save_format          = "mat-binary"
fixed_point_format           = true
page_screen_output           = false
print_empty_dimensions       = false
warn_function_name_clash     = false
```

`--verbose`
`-V` 冗長な出力に切り替えます。

`--version`
`-v` プログラムのバージョン番号を表示して終了します。

file *file* からコマンドを実行します。

Octave には，引数の数やオプションのすべてなど，コマンドラインについての情報を含む組み込み関数がいくつかあります。

`argv` [Built-in Variable]

Octave に渡されたコマンドライン引数は，この変数によって得ることができる。たとえば，以下ののコマンドにより Octave を起動したとすれば，

```
octave --no-line-editing --silent
```

`argv`は，`--no-line-editing`と`--silent`を含む文字列型配列となる。

もし実行可能な Octave スクリプトを入力したならば，`argv`はスクリプトに渡された引数リストを含む。実行可能な Octave スクリプトを作成する方法の一例は Section 2.6 [Executable Octave Programs], 頁 23 を参照せよ。

`program_invocation_name` [Built-in Variable]

`program_name` [Built-in Variable]

Octave が開始するとき，組み込み変数 `program_invocation_name`には，Octave を起動するためにシェルに入力した名前が自動的にセットされる。また，`program_name`の値には，`program_invocation_name`の最後の成分が自動的にセットされる。たとえば，Octave を起動するために‘`/usr/local/bin/octave`’と打ち込んだならば，`program_invocation_name`は`"/usr/local/bin/octave"` という値をとり，`program_name`は`"octave"`という値となる。

コマンドラインからスクリプトを実行する（たとえば，`octave foo.m` のように打ち込む）ならば，プログラム名は，スクリプトのファイル名にセットされる。実行可能な Octave スクリプトを作成する方法の一例は Section 2.6 [Executable Octave Programs], 頁 23 を参照せよ。

これらの変数を，Octave のコマンドラインを再生成するために使用した例を示します。

```
printf ("%s", program_name);
for i = 1:nargin
  printf (" %s", nth (argv, i));
endfor
printf ("\n");
```

Section 10.1 [Index Expressions], 頁 61 では，Octave において，どのように文字列や部分文字列の添え字を適切に扱うかを説明している。また，Section 13.1 [Defining Functions], 頁 87 には，変数 `nargin`についての情報がある。

2.1.2 スタートアップファイル

Octave を起動したとき，以下に示すファイルから，実行すべきコマンドを検索する。

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

octave-home は，Octave がインストールされているディレクトリ（標準では‘`/usr/local`’）である。このファイルが存在すると，インストールした Octave の全てのバージョンについて，全てのユーザに対して広域的に標準の Octave 環境を変更することとなります。このファイルを変更するときには，いくぶん注意が必要です。というのも，Octave を利用するすべてのユーザが影響を受けるからです。

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

octave-home は，Octave がインストールされているディレクトリ（標準では‘`/usr/local`’）である。このファイルが存在すると，インストールした Octave の特

定のバージョンについて，全てのユーザに対して広域的に標準の Octave 環境を変更することとなります。このファイルを変更するときには，いくぶん注意が必要です。というのも，Octave を利用するすべてのユーザが影響を受けるからです。

`~/.octaverc`
: このファイルは，通常は，標準の Octave 環境に個人的な変更を加えるために使用します。

`.octaverc`
: このファイルは，特定のプロジェクトに対して，標準の Octave 環境を変更するために使用することができます。Octave は，‘`~/.octaverc`’を読み込んだ後に，カレントディレクトリにおいてこのファイルを検索します。‘`~/.octaverc`’ファイルで `cd`コマンド利用すると，Octave が ‘`.octaverc`’を検索するディレクトリも影響を受けます。

  もし，ホームディレクトリで Octave を起動するならば，ファイル ‘`~/.octaverc`’ からのコマンドは一度だけ実行されるでしょう。

もし Octave を起動するときに`--silent`ではなく，`--verbose`オプションをつけるならば，各スタートアップファイルを実行するときには，メッセージを表示するようになります。

スタートアップファイルには，妥当な任意の Octave コマンドや関数の定義を含めることができます。

2.2 Octave の終了

`exit` (*status*) [Built-in Function]
`quit` (*status*) [Built-in Function]
現在の Octave セッションを終了する。オプション引数として整数 *status* を与えるならば，これを Octave の終了ステータスとしてオペレーティングシステムに渡す。

`atexit` (*fcn*) [Built-in Function]
Octave が終了するときに呼び出す関数を登録する。たとえば，

```
function print_fortune ()
  printf ("
%s
", system ("fortune"));
  fflush (stdout);
endfunction
atexit ("print_fortune");
```

この例は，Octave が終了するときにメッセージを表示することになる。

2.3 ヘルプを得るためのコマンド

このマニュアルの完全な文章は，Octave プロンプトから *help -i*というコマンドを入力することにより入手できます。さらに，個々のユーザ作成関数および変数についてのドキュメントも，*help*コマンドを通して得ることができます。この節ではマニュアル，およびユーザ提供関数や変数に対するドキュメントを読むために使用されるコマンドについて述べています。あなたの書いた関数に対して，どのように文書化すべきかについてのさらなる情報は，Section 13.6 [Function Files], 頁 92 を参照してください。

`help` [Command]

Octave の `help`コマンドは，GNU Info ブラウザを用いて，簡略な使用法を述べたメッセージを表示する，あるいは印刷マニュアルのオンライン版から直接情報を表示するために使用される。このコマンドを引数なしで呼び出すならば，全ての利用可能な演算子，関数および組み込み変数のリストを出力する。もし 1 番目の引数が`-i`であれば，与えられたトピックに対して，このマニュアルのオンライン版の索引を表示する。

たとえば，*`help help`*は，`help`コマンドを説明する短いメッセージを表示し，*`help -i help`* はマニュアルのオンライン版のこのノードにおいて，GNU Info ブラウザを起動する。

いちど GNU Info ブラウザを実行すると，キー入力 *`C-h`*を使用することにより，その使用法についてのヘルプを得ることができる。

`help`コマンドは，演算子についての情報を提供します。しかし，コマンドの区切りとして使用されるカンマやセミコロンについては，そうはいきません。それらに関するヘルプを得るには，*`help comma`*あるいは *`help semicolon`* と打ち込まなければなりません。

`INFO_FILE` [Built-in Variable]

変数 `INFO_FILE`は，Octave の info ファイルの位置を表す。初期状態では，"*`octave-home`*`/info/octave.info`"である。ここで *octave-home* は，Octave がインストールされたディレクトリである。

`INFO_PROGRAM` [Built-in Variable]

変数 `INFO_PROGRAM`は，実行すべき info プログラムを表す。初期状態では，"*`octave-home`*`/libexec/octave/`*`version`*`/exec/`*`arch`*`/info`" である。ここで *octave-home* は Octave がインストールされたディレクトリ，*version* は Octave のバージョン番号，および *arch* はシステムのタイプ（たとえば `i686-pc-linux-gnu`）である。初期値は，環境変数 `OCTAVE_INFO_PROGRAM`，あるいはコマンドライン引数`--info-program NAME`，さらにはスタートアップファイルにおいて `INFO_PROGRAM`変数に値をセットすることで上書きされる。

`MAKEINFO_PROGRAM` [Built-in Variable]

変数 `MAKEINFO_PROGRAM`は，Octave が Texinfo マークアップコマンドを含むヘルプ文章を書式化するために起動する makeinfo プログラムを表す。その初期値は`"makeinfo"`である。

`suppress_verbose_help_message` [Built-in Variable]

もし `suppress_verbose_help_message`の値がゼロでないならば，Octave は`help`コマンドからの出力あるいは組み込みコマンドの使用法メッセージの末端に，追加のヘルプ情報を追加することになる。

2.4 コマンドラインの編集

Octave は，広範囲なコマンドライン編集とヒストリ機能を提供するために，GNU readline ライブラリを使用しています。このマニュアルでは，最もよく使用される機能のみを解説しています。さらなる情報は，GNU Readline ライブラリのマニュアルを参照してください。

印刷可能文字（アルファベット，数字，機能など）を入力するには，単にその文字を打ち込みます。Octave は，その文字を挿入し，カーソルを次に進めます。

コマンドライン編集機能の多くは，制御文字を使用して操作します。たとえば，*`Control-a`*という文字は，カーソルを行の先頭に移動します。*`C-a`*を入力するには，〈CTRL〉キーを押したまま ⓐ キーを押します。次の節では，*`Control-a`*のような制御文字を *`C-a`*と表記しています。

コマンドライン編集機能の別のセットは，メタ文字を使用します。ある端末では，〈META〉キーを押しながらⓤを押すことにより，*M-u*を打ち込みます。もし使用している端末に〈META〉キーが無いならば，*ESC*キーで始まる 2 文字を入力することにより，メタ文字を入力できます。したがって，*M-u* を入力するには，〈ESC〉ⓤと打ち込めばよいのです。*ESC*入力法は，メタキーを持つ端末においても使用できます。以降の節において，*Meta-u* という ¥ メタ文字は，*M-u*と表記しています。

2.4.1 カーソルの移動

以下のコマンドは，カーソルを移動するものです。

C-b 1 文字戻します。

C-f 1 文字進めます。

〈DEL〉 カーソルの左側にある文字を削除します。

C-d カーソルが重なっている文字を削除します。

M-f 後の単語に進めます。

M-b 前の単語に戻ります。

C-a 行の先頭に移動します。

C-e 行の終端に移動します。

C-l スクリーンをクリアし，現在の行を最上部に表示します。

C-_
C-/ 最後に実行したことを取り消して戻します（アンドゥ）。すべての処理をアンドゥし，何も入力していない状態まで戻すことができます。

M-r この行について行った変更をすべてアンドゥします。これは，最初まで戻すために何回も ‘undo’ コマンドを入力するのと同じ動作です。

上の表は，入力行の編集を行うために必要な，最も基本的なキーストロークについて解説しています。大部分の端末においては，カーソルを進めたり戻したりするには，*C-f*や *C-b*の代わりに矢印キーを使用できます。

*C-f*が 1 文字進めるのに対して，*M-f*は 1 単語進めることに注意してください。おおざっぱに言えば，コントロールキーの組み合わせは文字を操作するものであって，メタキーの組み合わせは単語を操作するのです。

Octave プログラム内でスクリーンをクリアする関数もあります。

`clc ()` [Built-in Function]
`home ()` [Built-in Function]
端末のスクリーンをクリアし，カーソルを左上隅に移動する。

2.4.2 切り取りと貼り付け

テキストを切り取る（killing）というのは，その行からテキストを削除し，後で使用するためにその内容を保存しておくことです。通常は，貼り付け（yanking）することにより，それを行に戻します。もしコマンドについての解説で，テキストを「切り取る」という表現があるならば，後でテキストを違う位置（または同じ位置）に戻すことになると思ってください。

テキストを切り取るためのコマンド一覧を示します。

C-k 現在のカーソル位置から行末までのテキストを切り取ります。

M-d カーソル位置から現在の単語末までを切り取ります。もし，単語間にカーソルがあれば，次の単語末までになります。

M-⟨DEL⟩ カーソル位置から前の単語の先頭までを切り取ります。もし，単語間にカーソルがあれば，前の単語の先頭までになります。

C-w カーソル位置から前の空白までを切り取ります。これは *M-*⟨DEL⟩とは異なる動作です。なぜならば，単語区切りが異なるからです。

また，テキストを行に貼り付けて戻す方法について示します。貼り付けとは，切り取りバッファから，最も最近切り取ったテキストをコピーすることを意味します。

C-y 最も最近切り取ったテキストを，カーソル位置に貼り付けます。

M-y 切り取りリングを回し，新しい内容を貼り付けます。このコマンドは，事前に *C-y*または *M-y*コマンドを実行している場合にのみ実行できます。

切り取りコマンドを使用するとき，そのテキストは切り取りリング（kill-ring）に保存されます。任意回数の連続した切り取りは，切り取ったテキストとともに保存されます。その結果，それを貼り付けして戻すとき，それを一気に得ることがことができます。切り取りリングは，特定の行についてのものではありません。つまり，以前に入力した行において切り取ったテキストは，別の行を入力しているときに，後に貼り付けのために利用することができます。

2.4.3 テキストを変更するためのコマンド

以降のコマンドは，特別な意味を持つ文字（たとえば *TAB*や *C-q*など），を入力する，あるいはタイプミスを素早く修正するために使用できます。

C-q
C-v これを入力した次の文字を，そのまま行に追加します。これは，たとえば，*C-q*のような文字を挿入するためな方法です。

M-⟨TAB⟩ タブ文字を挿入します。

C-t カーソルの後ろにある文字をカーソル位置の文字の前に移動し，カーソルを進めます。もしカーソルが行の終端にあるならば，カーソル前の 2 つの文字を入れ替えます。

M-t カーソルの後ろにある単語をカーソルの前にある単語の位置に移動します。

M-u カーソル位置から現在の（あるいは次の）単語の末尾までの文字を大文字に変換し，カーソルをその単語末まで移動します。

M-l カーソル位置から現在の（あるいは次の）単語の末尾までの文字を小文字に変換し，カーソルをその単語末まで移動します。

M-c カーソルに続く文字（もしカーソルが単語間にあるならば，次の単語の始まり）を大文字に変換し，カーソルをその単語末に移動します。

2.4.4 readline 入力

以下のコマンドは，Octave でコマンドや変数名を補完できるようにするものです。

⟨TAB⟩ カーソル前の位置にテキストの補完を試みます。Octave は，コマンド名および変数名を補完できます。

M-? カーソル前の位置に補完することのできるテキストの一覧を表示します。

`completion_append_char` [Built-in Variable]

`completion_append_char`の値は，コマンドライン補完の試みがうまくいったときに付加する文字として使用します。初期状態の値は，`" "`（単一のスペース）です。

`completion_matches (`*`hint`*`)` [Built-in Function]

hint により与えられる可能な補完を生成します。

この関数は，Emacs のような Octave を操作したりユーザの入力を管理できるようなプログラムの利便のために提供されています。この関数が呼ばれるとき，現在のコマンド番号はインクリメントされません。これは仕様であり，バグではありません。

2.4.5 履歴編集のためのコマンド

Octave は，通常，あなたがタイプしたコマンドの足跡を保持しています。これにより，編集や再実行するために以前のコマンドを呼び出すことができます。Octave を終了するとき，最も最近入力したコマンドはファイルに保存されます。保存数の上限は，`history_size`なる変数で指定されます。Octave を起動するとき，変数`history_file`で指定したファイル名から，コマンドの初期リストを読み込みます。

履歴リストを単純に閲覧および検索するためのコマンドを示します。

⟨LFD⟩
⟨RET⟩ カーソル位置にかかわらず，その行を受け入れます。この行が空でないならば，それを履歴リストに追加します。もしこの行が履歴行であったならば，その履歴行をもとの位置に復元します。

C-p 履歴リストを「上に」（古い方に）移動します。

C-n 履歴リストを「下に」（新しい方に）移動します。

M-< 履歴における最初の行に移動します。

M-> 入力された履歴の末端に移動します。つまり，いま入力している行ですね！

C-r 現在の行から開始し，必要に応じて履歴を「上に」さかのぼって検索します。これはインクリメンタルサーチです。

C-s 現在の行から開始し，必要に応じて履歴を「下に」新しい方に向けて検索します。

大部分の端末では，履歴リストをたぐるために，*C-p*および *C-n*の代わりに矢印キーも使うことができます。

履歴リストを移動するためのキーボードコマンドに加えて，Octave は，履歴リストからのコマンド群を閲覧，編集，再実行するための関数を提供しています。

`history` *options* [Command]

もし引数なし実行するならば，`history`は，あなたが実行したコマンドのリストを表示する。以下のオプションが使用できる。

`-w` *file* 現在の履歴をファイル *file* に書き込む。もしそのファイル名を省略するならば，標準のファイル名（通常は‘`~/.octave_hist`’）を使用する。

`-r` *file* ファイル *file* を読み込み，現在の履歴をその内容で置き換える。もしファイル名を省略するならば，標準のファイル名（通常は‘`~/.octave_hist`’）を使用する。

n 履歴の最近 *n* 行のみを表示する。

`-q` 履歴リストを表示するときに，番号をつけない。これは，X Window System を使っているときに，コマンドの切り取りと貼り付けをするのに便利である。

たとえば，最近入力した 5 つのコマンドを，行番号なしで表示するには，*`history -q 5`* というコマンドを使用する。

`edit_history` *options* [Command]

もし引数なしで実行するならば，`edit_history`は，`EDITOR`変数で指定したエディタを使用して履歴リストを編集できるようにする。編集されることになるコマンド群は，最初にテンポラリファイルへとコピーされる。エディタを終了するとき，Octave は，そのファイルに残ったコマンドを実行する。関数を定義するために`edit_history`を使う方が，コマンドラインに直接入力しようとするよりも，より便利である。標準設定により，一連のコマンドは，エディタを終了するとすぐに実行される。コマンドの実行を避けるためには，エディタを終了する前に，バッファから単に全ての行を削除せよ。

`edit_history`コマンドは，編集したい最初と最後のコマンドの履歴番号を指定するための 2 つのオプション引数をとる。たとえば，以下のコマンド

```
edit_history 13
```

は，履歴リストの 13 番目から最後までの全てのコマンドを取り出す。以下のコマンド

```
edit_history 13 169
```

は，13 番目から 169 番目までのコマンドのみを展開する。最初の番号よりも 2 番目の番号に大きい値を指定するならば，編集するためのバッファにおく前に，コマンドのリストを逆順にする。もし両方の引数を省略するならば，履歴リストに存在する以前のコマンドが使用される。

`run_history` [*first*] [*last*] [Command]

`edit_history`と同様であるが，エディタを実行せず，履歴リストに存在するコマンドを単に実行する。

`EDITOR` [Built-in Variable]

A string naming the editor to use with the `edit_history` command. If the environment variable `EDITOR` is set when Octave starts, its value is used as the default. Otherwise, `EDITOR` is set to `"emacs"`.

`history_file` [Built-in Variable]

この変数は，コマンド履歴を保存するために使用するファイル名を指定する。標準の値は`"~/.octave_hist"`であるが，環境変数 `OCTAVE_HISTFILE`によって上書きすることができる。

`history_size` [Built-in Variable]

この変数は，どのくらいの項目を履歴ファイルに保存するかを指定する。標準の値は `1024`であるが，環境変数 `OCTAVE_HISTSIZE`によって上書きすることができる。

`saving_history` [Built-in Variable]

もし変数 `saving_history`がゼロでないならば，コマンドラインで入力した行が，変数`history_file`によって指定したファイルに保存される。

2.4.6 readlineのカスタマイズ

`read_readline_init_file` (*file*) [Built-in Function]
readline 初期化ファイル *file* を読み込む。もし *file* を省略するならば，標準のファイル（通常は‘`~/.inputrc`’）を読み込む。

2.4.7 プロンプトのカスタマイズ

以下の変数は，コマンドラインプロンプトの見た目をカスタマイズするために利用できます。Octave では，バックスラッシュでエスケープされる数々の特殊文字を挿入することにより，プロンプトをカスタマイズできるようになっています。それらの特殊文字は，以下のように解釈されます。

‘`\t`’ 時間です。

‘`\d`’ 日付です。

‘`\n`’ ラインフィードに続くキャリッジリターンを表示することにより，新しい行を開始します。

‘`\s`’ プログラム名（通常は‘`octave`’）です。

‘`\w`’ 現在の作業ディレクトリです。

‘`\W`’ 現在の作業ディレクトリのベース名です。

‘`\u`’ 現在のユーザ名です。

‘`\h`’ 最初の ‘.’ までのホスト名です。

‘`\H`’ ホスト名です。

‘`\#`’ 現在のコマンドの，Octave wを起動したときからカウントしたコマンド番号です。

‘`\!`’ このコマンドの履歴番号です。これは，Octave を起動したときの履歴リストのコマンド番号という点で，‘`\#`’とは異なっています。

‘`\$`’ もし有効な UID が 0 ならば‘`#`’，そうでなければ‘`$`’です。

‘`\nnn`’ 8 進数の文字コードで表記した文字が *nnn* です。

‘`\\`’ バックスラッシュです。

`PS1` [Built-in Variable]
プライマリプロンプトの文字列である。対話的に実行するとき，Octave がコマンドを読み込む準備ができたときに，プライマリプロンプトを表示する。

`PS1`の初期値は`"\s:\#> "`である。これを変更するには，

```
octave:13> PS1 = "\\u@\\H> "
```

のようなコマンドを入力する。これは，ホスト‘`kremvax.kgb.su`’にログインしたユーザ‘`boris`’について，‘`boris@kremvax> `’なるプロンプトになるだろう。2 つのバックスラッシュは，文字列に 1 個のバックスラッシュを入力するために必要であることに留意してほしい。Chapter 5 [Strings], 頁 35 を参照のせよ。

`PS2` [Built-in Variable]
セカンダリプロンプトの文字列である。これは，Octave が，コマンド入力を完了するために追加入力を期待するときに表示するものである。たとえば，複数行にわたる関数を定義するとき，Octave は，2 行目以降の各行の先頭に`PS1`の値を表示することになる。`PS2`の初期値は`"> "`である。

PS4 [Built-in Variable]
もし Octave が--echo-commandsオプションをつけて起動されるならば，各入力行をエコーする前に PS4の値を表示する。PS4の初期値は\n"+ "である。--echo-commandsの解説は，Section 2.1 [Invoking Octave], 頁 11 を参照せよ。

2.4.8 日記とエコーコマンド

Octave の日記機能は，打ち込んだ入力および Octave が表示した出力を記録することにより，対話セッションの全ての実行結果（ログ）を，別のファイルに保存できるようにします。

diary *options* [Command]
全てのコマンドおよびそれらが生み出した出力結果のリストを生成する。これは，端末で見えているように，入力と表示を一緒に混ぜる。

on 現在の作業ディレクトリの ‘diary’なる名前のファイルに，実行結果の記録を開始する。

off 日記ファイルへの記録を中止する。

file *file* という名前のファイルに実行結果を記録する。

引数を何もつけないときは，現在の設定のオン・オフを切り替える。

ときどき，関数あるいはスクリプトが評価されるときに，そのコマンドを見ることが有用なこともあります。これは，ある種の問題のデバッグに特に役立ちます。

echo *options* [Command]
コマンドを実行したときに，それを表示（エコー）するかどうかどうかをコントロールする。以下のオプションが利用できる。

on スクリプトファイルで実行されるコマンドを，画面に表示するようにする。

off スクリプトファイルで実行されるコマンドを，画面に表示しないようにする。

on all スクリプトファイルと関数で実行されるコマンドを，画面に表示するようにする。

off all スクリプトファイルと関数で実行されるコマンドを，画面に表示しないようにする。

もし引数をつけずに実行するならば，現在の設定のオン・オフを切り替える。

echo_executing_commands [Built-in Variable]
この変数は，エコー設定をコントロールするために使用される。これは，以下の値の合計となる。

1 スクリプトファイルから読み込んだコマンドをエコーする。

2 関数から読み込んだコマンドをエコーする。

4 コマンドラインから読み込んだコマンドをエコーする。

1 つ以上の設定を同時に行うことができる。たとえば，3 という値は，*echo on all*コマンドと等価である。

echo_executing_commandsの値は，*echo*コマンドおよびコマンドラインオプション--echo-inputによって設定される。

2.5 Octave はどのようにエラーを報告するのか

Octave は，妥当ではないプログラムについて，2 種類のエラーを報告します。

もし Octave が打ち込まれた内容を理解できないならば，*parse error* が発生します。たとえば，キーワードのスペルを間違ったとします。

```
octave:13> functon y = f (x) y = x^2; endfunction
```

Octave は，以下のようなメッセージをただちに表示するでしょう。

```
parse error:

  functon y = f (x) y = x^2; endfunction
          ^
```

大部分のパースエラーについて，Octave では，あなたの入力で意味をなすことのできない行の位置をマークするために，キャレット記号（‘^’）を使用します。この例の場合，`function`というキーワードのスペルが誤っていたため，Octave はエラーを生成しました。‘`function f`’と解釈するのではなく，Octave は 2 つの連続する変数名として解釈しました。これは，この文脈では妥当ではありません。`y`においてエラーをマークしたのは，最初の名前（functon）がそれ自身，妥当な入力として受け入れられたからです。

エラーメッセージの別の種類は，評価時に発生します。これらのエラーは実行時（ランタイム）エラー，あるいは時々評価エラーとも呼ばれます。これは，それらのエラーは，プログラムが実行（*run*），あるいは評価（*evaluated*）されたときに発生することによるものです。たとえば，以前の関数定義におけるミスを修正した後に，以下のように打ち込むとします。

```
octave:13> f ()
```

Octave は以下のように応答するでしょう。

```
error: ‘x’ undefined near line 1 column 24
error: evaluating expression near line 1, column 24
error: evaluating assignment expression near line 1, column 22
error: called from ‘f’
```

このエラーメッセージは複数の部分に分かれており，エラーの原因を突き止める手助けとなる一片の情報を与えています。これらメッセージは，最も奥にあるエラー点から生成されており，式および関数呼び出しを囲むトレースバックを提供します。

上の例において，最初の行は，‘`x`’なる名前の変数が，何かの関数あるいは式の 1 行 24 桁目付近で定義されていないことが判明したことを示しています。関数内で発生したエラーについて，その行は，関数の定義を含むファイルの始まりからカウントされます。トップレベルで発生したエラーについて，その行番号は，入力行の番号を示します。これは，通常はプロンプト文字列に示されています。

この例の 2 行目および 3 行目は，代入式でエラーが発生したことを示しており，エラーメッセージの最後の行は，エラーが関数 `f`の中で発生したことを示しています。もし仮に，関数 `f`が別の関数（たとえば `g`）から呼ばれていたならば，エラーのリストはもう 1 行多くなったことでしょう。

```
error: called from ‘g’
```

これらの関数呼び出しリストは，ふつうは，エラーが起こる前にプログラムを実行したパスをかなり容易にたどれるようになっており，再度実行する前にエラーを修正しやすくなっています。

2.6 Octaveで書かれたプログラムの実行

いちど Octave を習得してしまえば，‘#!’スクリプト機構を使用して，自己完結型の Octave スクリプトを書きたいと思うかもしれません。これは，GNU システムおよび多くの UNIX システム[1] で実行できます。

たとえば，以下のような行を含む‘hello’という名前のテキストファイルを作成してみます（ここで *octave-interpreter-name* は，お使いの Octave バイナリの完全なファイル名で置き換えるべきです）。

```
#! octave-interpreter-name -qf
# a sample Octave program
printf ("Hello, world!\n");
```

このファイルを（chmodを用いて）実行可能にした後，シェルから単に以下のように入力できます。

```
hello
```

システムは，以下のように入力したかのように Octave を実行する手はずを整えるでしょう。

```
octave hello
```

‘#!’で始まる行は，実行すべきインタプリタのフルパス名と，インタプリタに渡すコマンドライン引数を並べて項目リストとします。オペレーティングシステムは，与えた引数および実行したプログラムの引数リストをつけてインタプリタをを実行します。このリストの最初の項目は，Octave プログラムの完全なファイル名です。項目リストの残りは，Octave のオプション，あるいはデータファイル，またはその両方をとることができます。‘-qf’オプションは，通常の起動時メッセージの表示を抑制し，特定ユーザの‘~/.octaverc’ファイルの内容に依存して異なる挙動をしないようにするため，ふつうは，単独で実行する Octave プログラムにおいて指定されます。これについて，Section 2.1 [Invoking Octave], 頁 11 を参照してください。オペレーティングシステムの中には，‘#!’の後に認識される文字数に制限があるものがあるかもしれません。

自己完結型の Octave スクリプトは，プログラムを Octave 言語で記述する知識のないユーザが実行できるプログラムを書きたいときに役立ちます。

実行可能な Octave スクリプトを，コマンドライン引数をつけて実行するならば，その引数は組み込み変数 argvとして利用することができます。これについて，Section 2.1.1 [Command Line Options], 頁 11 を参照してください。たとえば以下のプログラムは，それを実行するために使用したコマンドラインを再生産するでしょう。

```
#! /bin/octave -qf
printf ("%s", program_name);
for i = 1:nargin
  printf (" %s", argv{i});
endfor
printf ("\n");
```

2.7 Octaveプログラム内のコメント

コメントは，人間が読むためにプログラムに含められ，ふつうはプログラムの一部ではないテキストのことです。コメントは，そのプログラムが行うことと，それがどのような動作をするか説明することができます。ほぼ全てのプログラミング言語は，コメント機能を提供しています。なぜならば，プログラムはコメントなしで理解することは概して困難であるためです。

---

[1] ‘#!’機構は Berkeley Unix， System V Release 4，およびいくつかの System V Release 3 systems から派生した UNIX システムで動作します。

Octave 言語において，コメントはシャープ記号 '#'，あるいはパーセント記号 '%'のいずれかで始まり，行末まで続きます。Octave インタプリタは，シャープまたはパーセント記号以降の行の内容を無視します。たとえば，関数 f に以下のようなコメントをおくことができます。

```
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
```

help コマンド（see Section 2.3 [Getting Help], 頁 14）は，関数のコメントの最初のブロック（コマンドラインから直接入力された関数についても）を見つけることができます。これは，Octave のユーザが，組み込み関数と自分で定義した関数のヘルプを得るためには，同じコマンドを使用できることを意味します。たとえば，上のように関数 f を定義した後，*help f* というコマンドを実行すると，以下の出力が得られます。

```
usage: f (x, t)

This function defines the right hand
side functions for a set of nonlinear
differential equations.
```

キーボードから入力した一時的な Octave プログラムにコメント行を含めることは可能ですが，ふつうはあまり役に立ちません。なぜならば，コメントの目的は，後から，そのプログラムを入力者または別の人に理解してもらう手助けをすることだからです。

3 データ型

Octave のすべてのバージョンには，数々の組み込みデータ型があります。その中には，実数および複素数のスカラと行列，文字列，あるいはデータ構造体型が含まれます。

少しの C++コードを書くことにより，新しい特殊なデータ型を定義することが可能です。システムの中には，Octave を実行している間に，新たなデータ型を動的にロードするものもあります。その場合は，新たなデータ型を追加するために，Octave の全てを再コンパイルする必要はありません。Octave の動的リンク機能に関する情報は，Section 13.8 [Dynamically Linked Functions], 頁 95 を参照してください。Section 3.2 [User-defined Data Types], 頁 26 には，Octave で新たなデータ型を定義するためにしなければならないことについて解説しています。

`typeinfo (`*expr*`)` [Built-in Function]
式 *expr* の型を，文字列として返す。もし *expr* を省略するならば，現在のところ準備されている全てのデータ型を含む文字列の配列を返す。

3.1 組み込みデータ型

標準の組み込みデータ型は，実数および複素数型のスカラと行列，範囲，文字列，およびデータ構造体です。さらなる組み込みデータ型が，将来のバージョンにおいて追加されるかもしれません。もし，現在のところ組み込みデータ型として提供されていない特殊な型が必要であれば，あなた独自のユーザ定義型を書くことを決心し，それを Octave の将来のリリース配布に向けて寄贈してください。

3.1.1 数値オブジェクト

Octave の組み込み数値オブジェクトには，実数および複素数のスカラと行列があります。すべての数値データは，現在，倍精度数値として保持されます。IEEE 浮動小数点書式を利用するシステムにおいては，とりうる数値は，およそ 2.2251×10^{-308} から 1.7977×10^{308} までの範囲で保持することになり，相対精度は，およそ 2.2204×10^{-16} となります。正確な値は，それぞれ `realmin`，`realmax`，および `eps` という変数によって与えられます。

行列オブジェクトは任意のサイズをとることができ，動的に変形したりサイズ変更することができます。さまざまな添え字機能を利用することにより，個々の行や列，部分行列を抜き出すことが容易です。Section 10.1 [Index Expressions], 頁 61 を参照してください。

さらなる情報を得るには，Chapter 4 [Numeric Data Types], 頁 29 を参照してください。

3.1.2 欠損データ

`NA` [Built-in Variable]
欠損値を表す。

`isna (`*x*`)` [Mapping Function]
x の要素が NA （欠損値）であれば 1，それ以外は 0 を返す。以下に例を示す。

```
is_NA ([13, Inf, NA, NaN])
     ⇒ [ 0, 0, 1, 0 ]
```

`is_nan_or_na (`*x*`)` [Mapping Function]
x の要素が NaN あるいは NA （欠損値）であれば 1，それ以外は 0 を返す。以下に例を示す。

```
is_NAN_or_NA ([13, Inf, NA, NaN])
     ⇒ [ 0, 0, 1, 1 ]
```

3.1.3 文字列オブジェクト

Octave における文字列型は，ダブルクオーテーションもしくはシングルクオーテーション記号でくくられた文字の並びから構成されています。内部的には，Octave は現在，文字列を文字の配列として保持しています。行列オブジェクトで動作する添え字の操作は，文字列に対しても動作します。

さらなる情報を得るには，Chapter 5 [Strings], 頁 35 を参照してください。

3.1.4 データ構造体オブジェクト

Octave のデータ構造体型は，さまざまな型の関連するオブジェクトを結びつける助けとなります。現在の実装では，結びつけた配列の添え字には文字列しか使えませんが，文法は C 言語の構造体に近いものになっています。

さらなる情報を得るには，Chapter 6 [Data Structures], 頁 43 を参照してください。

3.2 ユーザ定義データ型

いつの日か，ユーザ定義データ型を管理することについて，Octave のメカニズムの完全な記述を含めるために，これを展開したいと思っています。この機能がここで述べられるまで，‘`ov.h`’，‘`ops.h`’，および Octave の‘`src`’ディレクトリにある関連するファイルを読むことによって，これを実践しなければならないのです。

3.3 オブジェクトサイズ

以下に示す関数は，変数あるいは式のサイズを決定できるようにするものです。これらの関数は，全てのオブジェクトに対して使用することができます。意味のない命令を与えたときには，これら関数は -1 を返します。たとえば，Octave のデータ構造体型には行も列もありません。したがって，`rows` と `columns`関数は，引数に構造体を与えると -1 を返します。

`columns (a)` [Function File]
: a の列数を返す。

`rows (a)` [Function File]
: a の行数を返す。

`length (a)` [Built-in Function]
: a の「長さ」を返す。行列に対しては，行数と列数のどちらか大きい方の値を長さとします（この奇妙な定義は，Matlab との互換性を保つためです）。

`size (a, n)` [Built-in Function]
: a の行数と列数を返す。

  引数を 1 つだけ入力し，1 つの出力結果を受け取るとき，その結果は行ベクトルとして返される。複数の出力結果を受け取るならば，最初に行数，2 番目に列数のように値が割り当てられる。たとえば，以下のようになる。

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

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

2 つめの引数を与えるならば，sizeは，指定した次元のサイズを返す。たとえば，

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

この例では，与えた行列の列数を返す。

`isempty (a)` [Built-in Function]

a が空行列（行数と列数の片方，もしくは両方がゼロの行列）であれば 1 を返す。そうでなければ 0 を返す。

4 数値データ型

数値定数にはスカラ，ベクトル，あるいは行列，さらには複素数値があります。

数値定数の最も単純な型は，スカラです。これは単一の値であって，整数，小数，科学（指数）表記の数値，あるいは複素数をとることができます。すべての数値定数は，Octave の内部では倍精度浮動小数点形式で表されています（複素数型は，一対の倍精度浮動小数点数値として保持しています）。実数の数値の例を以下に示します。これらはすべて同じ値です。

```
105
1.05e+2
1050e-1
```

複素数の定数を指定するには，以下の形式の式を書くことになります。

```
3 + 4i
3.0 + 4.0i
0.3e1 + 40e-1i
```

これらはすべて等価です。上の例にある文字 'i'は，虚数単位を表します。虚数単位は，$\sqrt{-1}$ と定義されます。

Octave では，複素数の虚部を値として認識するため，数値と 'i'の間にスペースを入れてはいけません。もしスペースを含めたならば，Octave は以下のようなエラーメッセージを表示することでしょう：

```
octave:13> 3 + 4 i

parse error:

  3 + 4 i
        ^
```

上で用いた 'i'の代わりに，'j'，'I'あるいは 'J'を使うことができます。これら 4 つの記号は，すべて等価です。

4.1 行列

Octave では，行列を定義することは容易です。行列のサイズは，自動的に決定されます。ですから，行列の次数を明示的に指定する必要はありません。以下の式

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

は，以下の行列になります。

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

行列の要素には，任意の式をおくことができます。これは，各部分を組み合わせたときに，次数が全て意味をなすようにします。たとえば，上に示した行列が与えられているとき，以下の式

```
[ a, a ]
```

は，次の行列を生み出します。

```
ans =

  1  2  1  2
  3  4  3  4
```

しかし，以下の式

```
[ a, 1 ]
```

は，エラーになります。

```
error: number of rows must match near line 13, column 6
```

（もちろん，このエラーメッセージは，13 行目に入力したことを仮定しています）

行列を区切る角カッコの中身について，Octave は，スペースおよび改行が，要素および行の区切りに変換できるかどうかを決めます。そうでなければ，何もしません。その結果，以下の式

```
a = [ 1 2
      3 4 ]
```

は，正しく動きます。しかし，まだ混乱の原因も残っています。たとえば，以下の式において，

```
[ 1 - 1 ]
```

‘-’は二項演算子として扱われ，結果はスカラ 0 になります。しかし，以下の式においては，

```
[ 1 -1 ]
```

‘-’は単項演算子とみなされ，結果は，`[ 1, -1 ]`というベクトルになります。同様に，

```
[ sin (pi) ]
```

という式は，以下のように解釈されます。

```
[ sin, (pi) ]
```

これはエラーになるでしょう。なぜならば，`sin`関数を引数なしで呼び出そうとしているからです。これを修正するには，`sin`と開きカッコの間にスペースを入れないようにするか，以下のように，式全体をかっこでくくらねばなりません。

```
[ (sin (pi)) ]
```

シングルクオート文字（‘’’；これは転置演算子または文字列の区切りに使用されます）の周囲にある空白文字も，混乱のもとになることがあります。

```
[ 1 a' ]
```

この式は，シングルクオート文字が転置演算子として扱われ，その結果は `[ 1, 1 ]`なるベクトルとなる。しかし，以下の式

```
[ 1 a ' ]
```

は，次のようなエラーメッセージを出す。

```
error: unterminated string constant
```

妥当な式を解釈するときには，問題が起きないでしょう。

```
[ a 'foo' ]
```

明快にするには，行列の要素や行を区切るためには，いつもカンマやセミコロンを使うことが，おそらく最もよいことであろう。

`warn_separator_insert` [Built-in Variable]
もしカンマあるいはセミコロンがリテラル行列に自動的に挿入されるならば，警告を表示する。

行列や行列を表す変数名を打ち込むときには，Octave は行と列をきちんとそろえて表示します。もし行列の行が長すぎて画面に収まらないならば，Octave は行列を分割し，各部分の前に表示される列数に関する情報を表示します。

`output_max_field_width` [Built-in Variable]
この変数は，数値の出力フィールドの最大幅を指定する。初期状態では 10 である。

output_precision [Built-in Variable]
この変数は，数値を出力する際に表示する小数点以下の桁数を指定する。初期状態では 5 である。

output_precisionと output_max_field_widthの 2 つの変数を使用することにより，幅広い出力スタイルを達成することが可能です。合理的な組み合わせは，format関数を使用してセットすることになるでしょう。詳しくは，Section 16.1 [Basic Input and Output], 頁 106 を参照してください。

split_long_rows [Built-in Variable]
大きな行列について，Octave は与えられた行の全ての列を画面に表示することはできないであろう。これにより，端末が長い行を折り返したり切りつめるかどうかによっては，情報を失うか，ほとんど判読できない出力になってしまう。

split_long_rowsの値がゼロでないならば，Octave は行列を細かく分割して表示します。その各々は，端末の幅の限界に収まるようにします。行の各集合にはラベルが付けられており，現在表示している列数を容易に知ることができる。

```
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
```

split_long_rowsの標準状態は，ゼロではない。

値が非常に大きいまたは小さいとき，Octave は自動的に科学表記に切り替えます。これは，行列のどの値についても，有効桁を表示することを保証します。もし，すべての行列の値を固定小数点形式で表示したいならば，組み込み変数 fixed_point_formatにゼロでない値をセットすればよいです。しかし，そうすることは推奨しません。なぜならば，誤解しやすい出力になるかもしれないからです。

fixed_point_format [Built-in Variable]
この変数にゼロでない値をセットするならば，Octave は行列内の値をすべてスケール化する。その結果，最大の数値は，one leading digit で表記される。スケール因子は，出力の最初の行に表示される。たとえば，以下のようになる。

```
octave:1> logspace (1, 7, 5)'
ans =

  1.0e+07  *

  0.00000
  0.00003
  0.00100
  0.03162
  1.00000
```

一番目の値が実際には 1 であっても，ゼロであるように見えることに注意せよ。この理由により，`fixed_point_format`をゼロでない値にセットするときには，慎重になるべきである。

`fixed_point_format`は，標準状態では 0 である。

4.1.1 空行列

行と列の片方または両方の次数がゼロである空行列について，空行列に対する演算は以下の文献に記述されているように実行します：Carl de Boor, *An Empty Exercise*, SIGNUM, Volume 25, pages 2–6, 1990；C. N. Nett and W. M. Haddad, in *A System-Theoretic Appropriate Realization of the Empty Matrix Concept*, IEEE Transactions on Automatic Control, Volume 38, Number 5, May 1993. 簡潔に言えば，スカラ s，$m \times n$ の行列 $M_{m\times n}$，および $m \times n$ の空行列 $[]_{m\times n}$（片方または両方の次数がゼロに等しい行列）が与えられるとき，以下の関係が成立する。

$$
\begin{aligned}
s \cdot []_{m\times n} = []_{m\times n} \cdot s &= []_{m\times n} \\
[]_{m\times n} + []_{m\times n} &= []_{m\times n} \\
[]_{0\times m} \cdot M_{m\times n} &= []_{0\times n} \\
M_{m\times n} \cdot []_{n\times 0} &= []_{m\times 0} \\
[]_{m\times 0} \cdot []_{0\times n} &= 0_{m\times n}
\end{aligned}
$$

標準状態では，空行列の次元は空行列記号 '`[]`' とともに表示する。組み込み変数 `print_empty_dimensions`は，この挙動をコントロールする。

`print_empty_dimensions` [Built-in Variable]

`print_empty_dimensions`の値がゼロでない値をとるならば，空行列の次元は空行列記号 '`[]`' とともに表示される。たとえば，以下の式

```
zeros (3, 0)
```

は，次のような表示になる。

```
ans = [](3x0)
```

行列の行または列を削除する便利な方法として，代入文で空行列を使用することもできます。Section 10.6 [Assignment Expressions], 頁 68 を参照してください。

`warn_empty_list_elements` [Built-in Variable]

もし `warn_empty_list_elements`の値がゼロでないならば，行列リストに空行列が存在するとき，警告を表示する。たとえば，以下のような状況である。

```
a = [1, [], 3, [], 5]
```

標準状態は 0 である。

Octave が行列を含む式を解釈するとき，そのリストの要素がすべて定数かどうか決定するために試験をします。もしそうならば，単一の行列定数を含むリストに置き換えます。

4.2 範囲

範囲とは，等間隔に区切られた要素の行ベクトルを表記するために，便利な方法です。範囲式は，範囲の最初の要素，要素の増分（省略可），およびその範囲を超えない要素の最大値により定義されます。開始値，増分，および終端値はコロン（':'記号）で区切られます。これには，任意の数式や関数呼び出しを含むこともできます。もし増分を省略するならば，これは 1 であると仮定する。たとえば，以下の範囲

```
1 : 5
```

は，値の集合 ‘[1, 2, 3, 4, 5]’ を定義します。また，以下の範囲

```
1 : 3 : 5
```

は，‘[1, 4]’なる集合を定義します。

範囲定数は行ベクトルを指定しますが，Octave は，必要がないときには，範囲定数をベクトルに変換しません。これにより，‘1 : 10000’のような定数を，典型的な 32 ビットワークステーションでは，80,000 バイトの記憶領域を使用せずに書くことができます。

範囲の上限（増分が負値であれば下限）が，常に集合に含まれるとは限らないということに注意してください。また，浮動小数点を使用して定義された範囲は，驚くべき結果を生み出すことがあることにも留意してください。これは，Octave が範囲の値を計算するために，浮動小数点演算を使用しているためです。もし要素の数が決められていて，範囲に終点を含めることが重要ならば，かわりに `linspace`関数を使用するべきです（see Section 18.3 [Special Utility Matrices], 頁 148 を参照してください）。

Octave が範囲式を解釈するとき，式の要素がすべて定数かどうかを試験します。もしそうならば，範囲式を単一の範囲定数で置き換えます。

4.3 論理値

`true` [Built-in Variable]

理論真値を表す。

`false` [Built-in Variable]

理論偽値を表す。

4.4 数値オブジェクトの判定

`isnumeric (x)` [Built-in Function]

もし x が数値オブジェクトならば，ゼロ以外を返す。

`isreal (x)` [Built-in Function]

もし x が実数値オブジェクトならば，真値を返す。

`iscomplex (x)` [Built-in Function]

もし x が複素数の数値オブジェクトならば，真値を返す。

`ismatrix (a)` [Built-in Function]

もし x が行列ならば 1 を返し，それ以外は 0 を返す。

`isvector (a)` [Function File]

もし a がベクトルならば 1 を返し，それ以外は 0 を返す。

`isscalar (a)` [Function File]

もし x がスカラならば 1，それ以外は 0 を返す。

`issquare (x)` [Function File]
もし x が正方行列ならば，x の次元を返し，それ以外は 0 を返す。

`issymmetric (x, tol)` [Function File]
もし x が，*tol* によって指定される基準内にある対称行列ならば，x の次元を返す。そうでなければ，0 を返す。もし *tol* を省略するならば，計算機の精度に等しい基準を使用する。

`isbool (x)` [Built-in Functio]
もし x がブールオブジェクトならば，真値を返す。

5 文字列

文字列定数とは，ダブルクオートまたはシングルクオート文字のいずれかでくくられた文字の並びから構成されます。たとえば，以下の式は，

```
"parrot"
'parrot'
```

‘parrot’を含む文字列です。Octave における文字列は，任意の長さをとることができます。

シングルクオート文字は，行列の転置演算子にも使用されます（see Section 10.3 [Arithmetic Ops], 頁 64 を参照）。しかし，ダブルクオート記号は，Octave では他の用途に使用されていません。ですから，文字列を表すには，ダブルクオートを使用することが最善です。

いくつかの文字は，文字列定数にはそのまま含めることはできません。これらの文字は，エスケープシーケンスで表してください。エスケープシーケンスは，バックスラッシュ（‘\’；日本語環境では円記号）で始まる文字です。

エスケープシーケンスのひとつの利用法は，文字列にダブルクオート（あるいはシングルクオート）文字そのものを含めることです。素のダブルクオート記号は文字列の終端となるので，単一のダブルクオート文字を表すには，文字列の一部として ‘\"’ を使わなければなりません。バックスラッシュ（日本語では円記号）そのものも，ふつうでは含めることができない文字です。ですから，2 つの連続する文字列 ‘"\’は，"\"\\"あるいは’"\\’と表記することになります。同様に，2 つの文字 ‘’\’ を含む文字列は，’\’\\’ あるいは"’\\"と表記することになるでしょう。

バックスラッシュのもう一つの利用法は，改行などの非印刷記号を表すことです。一方で，これらの記号の多くを文字列定数に直接書き込むと，見た目が汚くなります。それを避けることはできません。

Octave で使用されるすべてのエスケープシーケンスを，以下の表に示します。これらは，C 言語で使用されているものと同じものです。

`\\` バックスラッシュ文字 ‘\’ そのものを表します。

`\"` ダブルクオート文字 ‘"’ そのものを表します。

`\’` シングルクオート文字 ‘’’ そのものを表します。

`\0` ASCII コード 0 の “null”（control-@）を表します。

`\a` ASCII コード 7 の “警告音”（control-g）を表します。

`\b` ASCII コード 8 の後退（control-h）を表します。

`\f` ASCII コード 12 の復帰（control-l）を表します。

`\n` ASCII コード 10 の改行（control-j）を表します。

`\r` ASCII コード 13 のキャリッジリターン（control-m）を表します。

`\t` ASCII コード 9 の水平タブ（control-i）を表します。

`\v` ASCII コード 11 の垂直タブ（control-k）を表します。

複数の文字列は，行列を定義する表記を用いて，連結することができます。たとえば，以下の式は，

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

‘foobarbaz’を含む文字列を生成します。行列を作ることについて，さらなる情報は See Chapter 4 [Numeric Data Types], 頁 29. を参照してください。

5.1 文字列の作成

`blanks (n)` [Function File]
n 個の空白からなる文字列を返す。

`char (x)` [Built-in Function]
`char (cell_array)` [Built-in Function]
`char (s1, s2, ...)` [Built-in Function]
数値行列，セル配列あるいは数値リストから文字列配列を作る。

もし引数が数値行列であれば，その行列の各要素を対応する ASCII 文字に変換する。例を以下に挙げる。

```
char ([97, 98, 99])
     ⇒ "abc"
```

もし引数が文字列のセル配列であれば，その結果は各要素がセル配列のひとつに対応するような文字配列となる。

複数の文字列を引数とした場合は，その結果は，引数に対応する各要素をもつ文字列配列となる。

返される値は，文字列配列の各行が同じ長さとなるように，空白がつめられている。

`int2str (n)` [Function File]
`num2str (x, precision)` [Function File]
`num2str (x, format)` [Function File]
数値を文字列に変換する。これらの関数は，それほど柔軟性はないが，MATLAB の互換性のために用意してある。結果をもっとよく操作するには，`sprintf`（see Section 16.2.4 [Formatted Output], 頁 115）を使用せよ。

`com2str (zz, flg)` [Function File]
この関数は，使用しないほうがよい。かわりに num2str 関数を使用せよ。

複素数を文字列に変換する。**Inputs**

zz 複素数値

flg 書式フラグ 0 （標準状態）: -1, 0, 1, 1i, 1 + 0.5i 1 （zpout を使用する）: -1, 0, + 1, + 1i, + 1 + 0.5i

`strcat (s1, s2, ...)` [Function File]
すべての引数を連結した文字列を返す。以下に例を示す。

```
s = [ "ab"; "cde" ];
strcat (s, s, s)
⇒ "ab ab ab "
        "cdecdecde"
```

`string_fill_char` [Built-in Variable]
この変数の値は，文字列行列の全ての文字列を同じ長さにするために，詰め物として使用される。これは，単一の文字にすべきである。標準状態の値は，" "（単一のスペース文字）である。以下の例を参照せよ。

```
string_fill_char = "X";
[ "these"; "are"; "strings" ]
     ⇒ "theseXX"
        "areXXXX"
        "strings"
```

str2mat (*s_1*, . . . , *s_n*) [Function File]

文字列 *s_1*, . . . , *s_n* を行に含む行列を返す。各々の文字列は，妥当な行列を作るため，空白文字で同じ長さにそろえられる。

この関数は，MATLAB を参考にして作成された。Octave では，全ての文字列が同じ長さでなくとも，[*s_1*; ...; *s_n*]とすることにより，文字列の行列を作ることができる。

ischar (a) [Built-in Function]

a が文字列ならば 1，そうでなければ 0 を返す。

isstr (a) [Function File]

この関数は，使用しない方がよい。かわりに ischar を使用せよ。

5.2 検索と置換

deblank (*s*) [Function File]

文字列 *s* から，連続する空白や null を取り除く。もし *s* が行列ならば，*deblank* は各々の行を，最長の文字列の長さに調整されます。

findstr (*s*, *t*, *overlap*) [Function File]

ふたつの文字列 *s* と *t* の短い方が，長い方に現れる位置を含むベクトルを返す。もしオプション引数 *overlap* がゼロでなければ，返されるベクトルには，重なりあう位置を含む（標準状態では，この動作をする）。以下の例を参照せよ。

```
findstr ("abababab", "a")
⇒ [ 1, 3, 5 ]
findstr ("abababa", "aba", 0)
⇒ [ 1, 5 ]
```

この実装は，2 番目の引数が最初の引数よりも長いならば，文字列を入れ替えるようになっている。したがって，長い方の文字列を最初に与えるようにしたほうがよい。

index (*s*, *t*) [Function File]

文字列 *s* の中に，文字列 *t* が最初に現れる位置を返す。もし見つからなければ，0 を返す。以下に例を示す。

```
index ("Teststring", "t")
⇒ 4
```

注意：この関数は，文字列配列には動作しない。

rindex (*s*, *t*) [Function File]

文字列 *s* の中に，文字列 *t* が最後に現れる位置を返す。もし見つからなければ，0 を返す。以下に例を示す。

```
rindex ("Teststring", "t")
⇒ 6
```

注意：この関数は，文字列配列には動作しない。

`split (`*s*`, `*t*`)` [Function File]

文字列 *s* を，*t* で区切って小片に分割し，文字列配列（妥当な行列になるように空白文字で詰め物をする）として返す。以下に例を示す。

```
split ("Test string", "t")
⇒ "Tes "
        " s  "
        "ring"
```

`strcmp (`*s1*`, `*s2*`)` [Function File]

ふたつの文字列を比較し，両者が同じならば真値を，そうでなければ偽値を返す。

注意：MATLAB との互換性を保つため，Octave の strcmp 関数は，文字列が等しいならば真値を，そうでなければ偽値を返す。これは，C 言語の同名のライブラリ関数とは逆の動作である。

`strrep (`*s*`, `*x*`, `*y*`)` [Function File]

文字列 *s* 内に現れる部分文字列 *x* のすべてを *y* で置き換える。以下に例を示す。

```
strrep ("This is a test string", "is", "&%$")
⇒ "Th&%$ &%$ a test string"
```

`substr (`*s*`, `*beg*`, `*len*`)` [Function File]

文字列 *s* の，*beg* 文字目から *len* 文字ぶんの長さを抜き出した部分文字列を返す。

もし *beg* が負値ならば，文字列の末端から開始する。もし *len* を省略するならば，*s* の末端までの部分文字列を返す。

以下に例を示す。

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

この関数は，AWK を模倣したものである。式 `s (beg : (beg + len - 1))` と同じ結果が得られる。

5.3 文字列の変換

`hex2dec (`*s*`)` [Function File]

文字列 *s* に格納された 2 進数に対応する，10 進の数値を返す。以下に例を示す。

```
hex2dec ("1110")
⇒ 14
```

もし *s* が文字列行列であれば，*s* の各行ごとに，変換した数値の列ベクトルを返す。変換できない行は，NaN とする。

`dec2bin (`*n*`, `*len*`)` [Function File]

非負の 10 進数値 *n* に対応する 2 進数を，1 と 0 の文字列として返す。以下に例を示す。

```
dec2bin (14)
⇒ "1110"
```

もし n がベクトルならば，値あたり 1 行として，最も長い値の幅にあわせてゼロを詰めた文字列行列を返す。

2 番目の引数 *len* はオプションであり，結果の最小桁数を指定する。

dec2hex (*n*, *len*) [Function File]

非負の整数 n に対応する 16 進文字列を返す。以下に例を示す。

```
dec2hex (2748)
⇒ "ABC"
```

もし n がベクトルならば，値あたり 1 行として，最も長い値の幅にあわせてゼロを詰めた文字列行列を返す。

2 番目の引数 *len* はオプションであり，結果の最小桁数を指定する。

hex2dec (*s*) [Function File]

16 進文字列 s に対応する非負の整数 n を返す。以下に例を示す。

```
hex2dec ("12B")
⇒ 299
hex2dec ("12b")
⇒ 299
```

もし s が文字列行列ならば，s の行ごとに，変換された数値の列ベクトルを返す。変換できない行は，NaN とする。

dec2base (*n*, *b*, *len*) [Function File]

非負の整数 n に対応する，b 進数の文字列を返す。以下の例は，3 進数である。

```
dec2base (123, 3)
⇒ "11120"
```

もし n がベクトルならば，値あたり 1 行として，最も長い値の幅にあわせてゼロを詰めた文字列行列を返す。

b が文字列ならば，b の文字が n の桁に対する記号として使用される。空白（' '）は，記号としては使用できない。以下の例は，上の例を文字列で置き換えたものである。

```
dec2base (123, "aei")
⇒ "eeeia"
```

3 番目の引数 *len* はオプションであり，結果の最小桁数を指定する。

base2dec (*s*, *b*) [Function File]

b 進数で表記された文字列 s を，整数へと変換する。以下の例は，3 進数で表記された文字列 "11120"を整数に変換している。

```
base2dec ("11120", 3)
⇒ 123
```

もし s が文字列行列であれば，s の各行ごとに，変換した数値の列ベクトルを返す。変換できない行は，NaN とする。変換前に，行は右寄せされるので，末尾の空白は無視される。

b が文字列であれば，b の文字列が，s の桁を表すために使用される。空白文字（' '）は，その記号としては使用されない。

```
base2dec ("yyyzx", "xyz")
⇒ 123
```

strjust (*s*, ["*left*"|"*right*"|"*center*"]) [Function File]
空白を含まない文字列 *s* を，文字列の左，右あるいは中央に寄せる。もし *s* が文字列配列ならば，配列の各文字列を寄せる。null 文字は空白で置き換える。もし寄せる方向が指定されないならば，すべての行は右寄せになる。

str2num (*s*) [Function File]
文字列 *s* を数値に変換する。

toascii (*s*) [Mapping Function]
文字列 *s* の，ASCII コードによる表現を行列として返す。以下に例を示す。

```
toascii ("ASCII")
     ⇒ [ 65, 83, 67, 73, 73 ]
```

tolower (*s*) [Mapping Function]
文字列 *s* について，大文字を小文字に置き換えた文字列を返す。アルファベットではない文字は，変更しないでそのままにする。以下に例を示す。

```
tolower ("MiXeD cAsE 123")
     ⇒ "mixed case 123"
```

toupper (*s*) [Built-in Function]
文字列 *s* について，小文字を大文字に置き換えた文字列を返す。アルファベットではない文字は，変更しないでそのままにする。以下に例を示す。

```
toupper ("MiXeD cAsE 123")
     ⇒ "MIXED CASE 123"
```

do_string_escapes (*string*) [Built-in Function]
文字列 *string* 内の特殊文字をエスケープして返す。

undo_string_escapes (*s*) [Built-in Function]
文字列内の特殊文字をエスケープ文字に戻す。たとえば，以下の式は，文字列変数 bellにアラート文字（control-g, ASCII コード 7）を代入している。

```
bell = "\a";
```

この文字列を表示すると，システムにより警告音が鳴るであろう（そのような設定にしてあれば)。これは，通常は望むべき結果である。しかし，時には，文字列のもともとの表現，すなわち特殊文字をエスケープシーケンスで置き換えた表現を表示できることが有効なことがある。以下の例，

```
octave:13> undo_string_escapes (bell)
ans = \a
```

は，表示できないアラート特殊文字を，表示可能な形式で置き換えている。

`warn_num_to_str` [Built-in Variable]
もし `warn_num_to_str`の値がゼロでないならば，行列表記において，文字列が文字と数値が混在した状態になっているときには，数値を ASCII コードに暗黙的に変換するときに，警告を表示する。以下の例

```
[ "f", 111, 111 ]
     ⇒ "foo"
```

は，`warn_num_to_str`がゼロでないときには，警告を出す。標準状態では 1 である。

`warn_str_to_num` [Built-in Variable]
もし `warn_str_to_num`の値がゼロでないならば，文字列をその対応する ASCII コードの数値へと暗黙的に変換するときに，警告を表示する。以下の例

```
"abc" + 0
     ⇒ 97 98 99
```

は，`warn_str_to_num`がゼロでないときには，警告を出す。標準状態では 0 である。

`warn_single_quote_string` [Built-in Variable]
文字列定数を導入するために，シングルクオート文字を使うならば，警告を表示する。

5.4 キャラクタクラス関数

Octave では，標準 C 言語ライブラリにある関数と同様に，以下のような文字種類をテストする関数も提供しています。これらは，すべて文字列配列について処理を行い，0 と 1 からなる行列を返します。0 でない要素は，もとの文字列配列における対応する文字の状態が真であることを示しています。以下に例を示します。

```
isalpha ("!Q@WERT^Y&")
     ⇒ [ 0, 1, 0, 1, 1, 1, 1, 0, 1, 0 ]
```

`isalnum` (*s*) [Mapping Function]
アルファベットあるいは数値である文字について，1 を返す（`isalpha (`*s*`)`あるいは `isdigit (`*s*`)`のいずれかが真である）。

`isalpha` (*s*) [Mapping Function]
`isletter` (*s*) [Mapping Function]
アルファベットである文字について，1 を返す（`isupper(`*s*`)` あるいは `islower(`*s*`)`のいずれかが真である）。

`isascii` (*s*) [Mapping Function]
ASCII コード（10 進数で 0 から 127 の範囲にある）文字について，1 を返す。

`iscntrl` (*s*) [Mapping Function]
制御文字に対して 1 を返す。

`isdigit` (*s*) [Mapping Function]
10 進数値を表す文字に対して 1 を返す。

`isgraph (s)` [Mapping Function]
印刷可能文字に対して 1 を返す（空白文字は含まない）。

`islower (s)` [Mapping Function]
小文字アルファベットに対して 1 を返す。

`isprint (s)` [Mapping Function]
印刷可能文字に対して 1 を返す（空白文字を含む）。

`ispunct (s)` [Mapping Function]
句読点に対して 1 を返す。

`isspace (s)` [Mapping Function]
ホワイトスペース文字（スペース，改行，復帰，キャリッジリターン，タブおよび垂直タブ）に対して 1 を返す。

`isupper (s)` [Mapping Function]
大文字アルファベットに対して 1 を返す。

`isxdigit (s)` [Mapping Function]
16 進数値文字に対して 1 を返す。

6 データ構造体

Octave は，データを構造体にまとめる機能をもっています。現在の実装では，連想配列の添え字には文字列しか使用できませんが，文法は C 言語の構造体に似ています。この章では，Octave でデータ構造体を使用する例を紹介します。

構造体の要素は，どのような変数型でも含めることができます。たとえば，以下の 3 つの式を入力すると，3 つの要素を持つ構造体ができあがります。

```
x.a = 1
x.b = [1, 2; 3, 4]
x.c = "string"
```

構造体の値を表示するには，通常の変数と同じように，単に変数名を入力するだけです。

```
octave:2> x
x =
{
  a = 1
  b =

    1  2
    3  4

  c = string
}
```

Octave では，要素が任意の順序で表示されることに注意してください。

構造体はコピーすることができます。

```
octave:1> y = x
y =
{
  a = 1
  b =

    1  2
    3  4

  c = string
}
```

構造体はそれ自体が値なので，構造体の要素として他の構造体を参照することもできます。以下の例では，構造体 `x` の要素 `b` の値を，構造体に変更しています。

```
octave:1> x.b.d = 3
x.b.d = 3
octave:2> x.b
ans =
{
  d = 3
}
octave:3> x
x =
{
  a = 1
  b =
  {
    d = 3
  }

  c = string
}
```

Octave が，他の構造体を含む構造体の値を表示するときには，ある深さまでしか表示されないことに注意してください。以下に例を示します。

```
octave:1> a.b.c.d.e = 1;
octave:2> a
a =
{
  b =
  {
    c =
    {
      d: 1x1 struct
    }
  }
}
```

これは，深く入れ子になった構造体を出力するに当たり，長くなったり混乱したりしないようにするためのものです。

`struct_levels_to_print` [Built-in Variable]

`struct_levels_to_print`に値を設定することにより，Octave が，どの深さまで構造体を表示するかを指定できる。標準状態は 2 である。

関数は，構造体を返すことができます。たとえば，以下の関数は，行列の実部と虚部を分離して，同じ構造体変数の 2 つの部分に格納します。

```
octave:1> function y = f (x)
> y.re = real (x);
> y.im = imag (x);
> endfunction
```

この関数に複素数を渡して呼び出すと，`f`はもとの引数の実部と虚部を含むデータ構造体を返します。

```
octave:2> f (rand (2) + rand (2) * I);
ans =
{
  im =

    0.26475  0.14828
    0.18436  0.83669

  re =

    0.040239  0.242160
    0.238081  0.402523
}
```

関数が返した値のリストは，構造体の要素を含むことになります。また，それらは任意の他の変数のように添え字をつけることもできます。以下の例を参照してください。

```
octave:1> [ x.u, x.s(2:3,2:3), x.v ] = svd ([1, 2; 3, 4])
x.u =

  -0.40455  -0.91451
  -0.91451   0.40455

x.s =

  0.00000  0.00000  0.00000
  0.00000  5.46499  0.00000
  0.00000  0.00000  0.36597

x.v =

  -0.57605   0.81742
  -0.81742  -0.57605
```

forステートメントの特別な形式をとることにより，ループの中で，構造体の全ての要素を通して処理することも可能です（see Section 12.5 [The for Statement], 頁 80 を参照してください）。

以下の関数を使用することにより，構造体についての情報を得ることができます。

`isstruct (`*expr*`)` [Built-in Function]

式 *expr* が構造体であれば 1 を返す。

`fieldnames (`*struct*`)` [Built-in Function]

構造体 *struct* の要素に名付けられた文字列の，セル配列を返す。引数に構造体を渡さずに`fieldnames`を呼び出したときは，エラーとなる。

`isfield (`*expr*`, `*name*`)` [Built-in Function]

式 *expr* が構造体であり，*name* という名前の要素を含むときに 1 を返す。最初の引数は構造体，2 番目の引数は文字列でなければならない。

7 コンテナ

7.1 リスト

`list` (*a1*, *a2*, ...) [Built-in Function]
引数 *a1*，*a2*，...によって与えられる要素を持つ，新たなリストを生成する。

`nth` (*list*, *n*) [Built-in Function]
リスト *list* の *n* 番目の要素を返す。

`append` (*list*, *a1*, *a2*, ...) [Built-in Function]
リスト *list* に *a1*，*a2*，...を追加することにより生成される新たなリストを返す。もし，追加すべき引数の中にリストが含まれたならば，その要素は個々に追加される。たとえば，

```
x = list (1, 2);
y = list (3, 4);
append (x, y);
```

この式は，4 つの要素を含むリスト '`(1 2 3 4)`'となるが，3 つの要素 '`(1 2 (3 4))`'を含むリストにはならない。

`reverse` (*list*) [Built-in Function]
リスト *list* の要素を逆順にして作られる新たなリストを返す。

`splice` (*list_1*, *offset*, *length*, *list_2*) [Built-in Function]
リスト *list_1* について，*offset* 番目から *length* 個の要素を，*list_2* の成分で置き換える。もし *length* を省略するならば，*offset* から *list_1* の終端までの全要素が置き換えられる。特別な場合として，*offset* が *list_1* の長さ+1 よりも大きく，*length* が 0 ならば，splice 関数は `append (list_1, list_2)`に等価である。

`islist` (*x*) [Built-in Function]
もし *x* がリストならば，ゼロ以外を返す。

7.2 セル配列

`cell` (*x*) [Built-in Function]
`cell` (*n*, *m*) [Built-in Function]
新たなセル配列オブジェクトを生成する。もし単一のスカラを引数にして呼び出すならば，指定した次元の正方セル配列を返す。もし，引数にふたつのスカラを与えるならば，それらを行数および列数と見なす。もし，ふたつの要素をもつベクトルを与えるならば，その要素をそれぞれ行数および列数として使用する。

`cellstr` (*string*) [Built-in Function]
文字列配列 *string* の要素から，新たなセル配列オブジェクトを生成する。

`iscell` (*x*) [Built-in Function]
もし *x* がセル配列オブジェクトならば真値を返し，そうでなければ偽値を返す。

8 I/Oストリーム

`isstream` (*x*) [Built-in Function]

もし *x* がストリームオブジェクトならば真値を，そうでなければ偽値を返す。

9 変数

変数とは，値に名前を割り当て，それを後に参照できるようにするものです。多くの例において，すでに変数を見ていることでしょう。変数の名前は，文字と数値，あるいはアンダースコアから構成されていなければなければなりません。しかし，数字で始まってはいけません。Octave は，変数名の長さに制限を設けないようにしています。しかし，約 30 文字よりも長い変数名は，まれには役に立つかもしれません。以下に示す変数名は，いずれも有効です。

```
x
x15
__foo_bar_baz__
fucnrdthsucngtagdjb
```

しかし，`__foo_bar_baz__`のように，先頭と末尾に 2 つのアンダースコアをつけた変数名は，Octave が内部的に使用するために予約されていると思ってください。あなたが書こうとするコードには，このような変数名を使うべきではありません。ただし，Octave の文書化された内部変数および組み込みシンボリック定数にアクセスする場合を除きます。

変数名の大文字と小文字は区別されます。`a`と `A`は，異なる変数です。

変数名は，それ自体が妥当な式です。それは，変数の現在値を表します。変数には，代入演算子あるいはインクリメント演算子によって，新しい値を割り当てます。Section 10.6 [Assignment Expressions], 頁 68 を参照してください。

いくつかの変数には，特別な意味が込められているものがあります。たとえば，`ans`は現在の作業ディレクトリ名を保持しており，`pi`は円周率が割り当てられています。事前に定義されている全ての変数名のリストは，Section 9.4 [Summary of Built-in Variables], 頁 55 を参照してください。これらの組み込み変数の中には，定数であって，値が変更できないものがあります。そのほかの組み込み変数は，ふつうの変数のように値を代入できますが，その値は Octave によって使われたり，自動的に変更されたりします。

Octave における変数には，特定の型はありません。したがって，同じプログラムにおいて，ある変数にまず数値を格納しておき，後になって同じ変数に文字列を格納することが可能です。変数は，値を代入する前に使用することはできません。これを実行しようとすると，エラーになります。

9.1 グローバル変数

グローバル宣言された変数は，きちんと引数として渡すことなく，関数の本体内からアクセスすることができるようになります。

任意の変数は，`global`宣言ステートメントを使用して，グローバル宣言することができます。以下の文は，すべてグローバル宣言です。

```
global a
global a b
global c = 2
global d = 3 e f = 5
```

グローバル変数は，`global`ステートメントにおいて，一度しか初期化されません。たとえば，以下のようなコードを実行します。

```
global gvar = 1
global gvar = 2
```

このとき，グローバル変数 `gvar`は 1 であり，2 ではありません。

グローバル変数にアクセスするためには，関数の本体内で変数をグローバルとして宣言しておく必要があります。たとえば，以下の式について，

```
global x
function f ()
  x = 1;
endfunction
f ()
```

グローバル変数 xの値は，1 にセットされません。グローバル変数 xの値を変更するためには，以下のように，関数の本体内でグローバル宣言を行わなければなりません。

```
function f ()
  global x;
  x = 1;
endfunction
```

関数の引数リストにグローバル変数を渡すとき，ローカルコピーが生成され，グローバル値は修正されない。たとえば，ある関数

```
function f (x)
  x = 0
endfunction
```

が存在し，上の段階でグローバル変数として xを

```
global x = 13
```

として定義してあるとすれば，

```
f (x)
```

この関数は，関数内部にある xの値 0 を表示するでしょう。しかし，上の段階にある xの値は変更されない。なぜならば，この関数は，その引数のコピーについて動作するからである。

`isglobal (`*name*`)` [Built-in Function]

もし *name* がグローバルにアクセスできるならば 1 を，そうでなければ 0 を返す。以下に例を示す。

```
global x
isglobal ("x")
     ⇒ 1
```

9.2 持続型変数

関数内で持続宣言された変数は，次に同じ関数を呼び出すまで，その内容をメモリ内に保持したままになります。持続変数とグローバル変数の違いは，持続変数は特定の関数内でしか利用できないローカルなものであるのに対し，後者はどこからでも見ることができることです。

任意の変数は，persistent宣言ステートメントを使用して，持続宣言することができます。以下の文は，すべて持続宣言です。

```
persistent a
persistent a b
persistent c = 2
persistent d = 3 e f = 5
```

持続変数の挙動は，C 言語の静的変数と等価です。Octave では staticコマンドも使用でき，persistentと同じ意味です。グローバル変数とは異なり，初期化ステートメントごとに変数を最初期化します。たとえば，以下のコード

```
persistent pvar = 1
persistent pvar = 2
```

を実行した後では，持続変数 pvarの値は 2 になります。

9.3 変数の状態

clear [*-x*] *pattern* . . . [Command]

与えられたパターンにマッチする名前を，シンボルテーブルから削除する。パターンには，以下の特殊文字を含めることができる。

`?` 任意の 1 文字にマッチする。

`*` ゼロ文字以上の文字列にマッチする。

`[` *list* `]` *list* によって指定する文字リストのいずれかにマッチする。もし，最初の文字が`!`または`^`であれば，*list* によって指定された文字を除くすべての文字にマッチする。たとえば，‘`[a-zA-Z]`’というパターンは，アルファベットの全ての大文字と小文字にマッチするだろう。

たとえば，以下のコマンド

```
clear foo b*r
```

は，fooという名前と，bという文字で始まり rで終わるすべての名前をクリアする。

もし何も引数を与えずに clearを呼び出すと，すべてのユーザ定義変数（ローカルとグローバルの両方）を，シンボルテーブルから削除します。もし clearを少なくともひとつ引数を与えて呼び出すならば，引数にマッチする可視的な名前のみをクリアする。たとえば，fooなる関数を定義してあり，foo = 2を実行して隠してあったと仮定する。コマンド *clear foo*を一度実行すると，変数の定義をクリアし，関数としての fooの定義を元に戻す。2 回目に *clear foo* を実行すると，関数定義を削除することになる。

-xをつけて実行すると，パターンにマッチしない名前をクリアする。

このコマンドは，関数内で使用することはできない。

who *options pattern* . . . [Command]
whos *options pattern* . . . [Command]

与えられたパターンにマッチする，現在定義されているシンボルを一覧表示する。利用可能なオプションを以下に示す。これらは，先頭の 1 文字に省略することはできるが，組み合わせて使用することはできない。

`-all` 現在で意義されているシンボルを，全て表示する。

`-builtins`
組み込み変数および関数を一覧表示する。これには，現在のところコンパイルされている関数ファイルを含む。しかし，LOADPATHに存在する関数ファイルは含まれない。

`-functions`
ユーザ定義関数を一覧表示する。

`-long` あるシンボルの型と次元を含めた，詳細な一覧を表示をする。出力結果の 1 列目におけるシンボルは，シンボルとして再定義が可能かどうか，およびクリアできるかどうかを示している。

`-variables`
ユーザ定義変数を一覧表示する。

妥当なパターンは，上で示した `clear`コマンドの内容と同じである。パターンが与えられなかったならば，与えられたカテゴリからの全てのシンボルを表示する。標準状態では，ローカルで見ることのできるユーザ定義変数および関数のみを表示する。

*`whos`*コマンドは，*`who -long`*と等価である。

`whos` *options pattern ...* [Command]
`who`を参照せよ。

`whos_line_format` [Built-in Variable]
この文字列は，表示される変数の属性についての順序を決定する。以下のコマンドを使用する。

`%b` 変数によって占められるバイト数を表示する。

`%e` 変数によって保持される要素を表示する。

`%n` 変数名を表示する。

`%p` 変数のプロテクト属性を表示する。

`%s` 変数の次元を表示する。

`%t` 変数の型名を表示する。

どのコマンドも，修飾子をとることもできる。

`l` 左寄せにする。

`r` 右寄せにする（標準設定）。

`c` 中央寄せにする（コマンド%s にのみ有効）。

コマンドは，以下のように構成される。%[modifier]<command>[:size_of_parameter[:center-specific[:print_dims[:balance]]]];

コマンドおよび修飾子は，すでに説明したとおりである。size_of_parameter は，表示に何列必要なのかを指定するものである。print_dims は，表示に何次元必要かを指定する。もし次数が print_dims を越えるならば，次数は x-D のように表示される。center-specific および print_dims は，コマンド%s にのみ応用することができる。print_dims に負の値を指定すると，どんな場合でも全ての次元を表示するようになる。balance は，次元文字列を表示するオフセットを指定する。

標準状態のフォーマットは，" %p:4; %ln:6; %cs:16:6:8:1; %rb:12; %lt:-1; " である。

`exist` (*`name`*, *`type`*) [Built-in Function]
name が変数として存在すれば 1，Octave の `LOADPATH`にある関数ファイル（*name* に‘`.m`’を付加したもの）であれば 2，Octave の `LOADPATH`にある‘`.oct`’ファイルがあれば 3，組み込み関数であれば 5，ディレクトリ名であれば 7，組み込み変数であれば 101，組み込み定数であれば 102,（コマンドラインから入力された）ファイルに関連しない関数であれば 103 を返す。

それ以外は 0 を返す。

この関数は，*name* で呼び出される通常のファイルが Octave の `LOADPATH` に存在するならば 2 を返す。もしファイルの他の種類について情報が欲しいならば，かわりに `file_in_path`および `stat`関数を適当に組み合わせることになる。

オプション引数として *type* を与えるならば，特定の型に該当する名前についてのみチェックする。*type* には，以下の値を指定できる。

‘`"var"`’ 変数についてのみチェックする。

‘`"builtin"`’
組み込み関数についてのみチェックする。

‘`"file"`’ ファイルについてのみチェックする。

‘`"dir"`’ ディレクトリについてのみチェックする。

`document` (*symbol*, *text*) [Built-in Function]
symbol に解説文字列 *text* をセットする。

`type` *options name* . . . [Command]
関数を参照する各々の *name* の定義を表示する。

通常は，*name* がユーザ定義あるいは組み込み関数かどうかも表示する。この挙動を抑制するには`-q`オプションを使用する。

現在，Octave は明らかにコンパイルされた関数のみを表示する。なぜならば，プログラム文を再生成するための関数の内部表現を使用するためである。

コメントは表示されない。これは，現在，Octave のパーサが関数ファイルのテキストを内部表現へと変換するとき，コメントを捨てるからである。この問題は，将来のリリースで修正する予定である。

`which` *name* . . . [Command]
各々の *name* の型を表示する。もし *name* が関数ファイルにより定義されているならば，ファイルの完全な名前も表示する。

9.4 組み込み変数の要約

Octave の全ての組み込み関数を，その初期状態および追加情報のための相互参照とともに以下にまとめます。以下の表において，*octave-home* は Octave がインストールされているルートディレクトリ（標準では‘`/usr/local`’）を表し，*version* は Octave のバージョン番号（たとえば 2.1.x）を表し，*arch* は Octave がコンパイルされたシステムのタイプ（たとえば `i586-pc-linux-gnu`）を表します。

`DEFAULT_LOADPATH`
Section 13.6 [Function Files], 頁 92 を参照してください。
初期値: "`.:`*octave-home*`/lib/`*version*"

`EDITOR` Section 2.4.5 [Commands For History], 頁 18 を参照してください。
初期値: `"emacs"`

`EXEC_PATH`
: Section 34.3 [Controlling Subprocesses], 頁 293 を参照してください。
初期値: `":$PATH"`

`INFO_FILE`
: Section 2.3 [Getting Help], 頁 14 を参照してください。
初期値: "*octave-home*`/info/octave.info`"

`INFO_PROGRAM`
: Section 2.3 [Getting Help], 頁 14 を参照してください。
初期値: "*octave-home*`/libexec/octave/`*version*`/exec/`*arch*`/info`"

`LOADPATH` Section 13.6 [Function Files], 頁 92 を参照してください。
初期値: `":"`, これは，組み込み変数 `DEFAULT_LOADPATH`によって指定されるディレクトリを使用するように Octave に伝えます。

`OCTAVE_HOME`
: 初期値: `"/usr/local"`

`PAGER` Chapter 16 [Input and Output], 頁 105 を参照してください。
初期値: `"less"`, or `"more"`

`PS1` Section 2.4.7 [Customizing the Prompt], 頁 20 を参照してください。
初期値: `"\s:\#> "`

`PS2` Section 2.4.7 [Customizing the Prompt], 頁 20 を参照してください。
初期値: `"> "`

`PS4` Section 2.4.7 [Customizing the Prompt], 頁 20 を参照してください。
初期値: `"+ "`

`automatic_replot`
: Section 17.1 [Two-Dimensional Plotting], 頁 129 を参照してください。
初期値: 0

`beep_on_error`
: Chapter 14 [Error Handling], 頁 101 を参照してください。
初期値: 0

`completion_append_char`
: Section 2.4.4 [Commands For Completion], 頁 17 を参照してください。
初期値: `" "`

`default_save_format`
: Section 16.1.3 [Simple File I/O], 頁 110 を参照してください。
初期値: `"ascii"`

`crash_dumps_octave_core`
: Section 16.1.3 [Simple File I/O], 頁 110 を参照してください。
初期値: 1

`fixed_point_format`
: Section 4.1 [Matrices], 頁 29 を参照してください。
初期値: 0

`gnuplot_binary`
: Section 17.3 [Three-Dimensional Plotting], 頁 135 を参照してください。
初期値: `"gnuplot"`

`history_file`
: Section 2.4.5 [Commands For History], 頁 18 を参照してください。
初期値: `"~/.octave_hist"`

`history_size`
: Section 2.4.5 [Commands For History], 頁 18 を参照してください。
初期値: 1024

`ignore_function_time_stamp`
: Section 13.6 [Function Files], 頁 92 を参照してください。
初期値: `"system"`

`max_recursion_depth`
: Section 10.2.2 [Recursion], 頁 64 を参照してください。
初期値: 256

`output_max_field_width`
: Section 4.1 [Matrices], 頁 29 を参照してください。
初期値: 10

`output_precision`
: Section 4.1 [Matrices], 頁 29 を参照してください。
初期値: 5

`page_screen_output`
: Chapter 16 [Input and Output], 頁 105 を参照してください。
初期値: 1

`print_answer_id_name`
: Section 16.1.1 [Terminal Output], 頁 106 を参照してください。
初期値: 1

`print_empty_dimensions`
: Section 4.1.1 [Empty Matrices], 頁 32 を参照してください。
初期値: 1

`return_last_computed_value`
: Section 13.5 [Returning From a Function], 頁 91 を参照してください。
初期値: 0

`save_precision`
: Section 16.1.3 [Simple File I/O], 頁 110 を参照してください。
初期値: 17

`saving_history`
: Section 2.4.5 [Commands For History], 頁 18 を参照してください。
初期値: 1

`sighup_dumps_octave_core`
: Section 16.1.3 [Simple File I/O], 頁 110 を参照してください。
初期値: 1

`sigterm_dumps_octave_core`
: Section 16.1.3 [Simple File I/O], 頁 110 を参照してください。
初期値: 1

`silent_functions`
: Section 13.1 [Defining Functions], 頁 87 を参照してください。
初期値: 0

`split_long_rows`
: Section 4.1 [Matrices], 頁 29 を参照してください。
初期値: 1

`struct_levels_to_print`
: Chapter 6 [Data Structures], 頁 43 を参照してください。
初期値: 2

`suppress_verbose_help_message`
: Section 2.3 [Getting Help], 頁 14 を参照してください。
初期値: 1

`warn_assign_as_truth_value`
: Section 12.1 [The if Statement], 頁 75 を参照してください。
初期値: 1

`warn_comma_in_global_decl`
: Section 9.1 [Global Variables], 頁 51 を参照してください。
初期値: 1

`warn_divide_by_zero`
: Section 10.3 [Arithmetic Ops], 頁 64 を参照してください。
初期値: 1

`warn_empty_list_elements`
: Section 4.1.1 [Empty Matrices], 頁 32 を参照してください。
初期値: 0

`warn_fortran_indexing`
: Section 10.1 [Index Expressions], 頁 61 を参照してください。
初期値: 0

`warn_function_name_clash`
: Section 13.6 [Function Files], 頁 92 を参照してください。
初期値: 1

`warn_imag_to_real`
: Section 18.3 [Special Utility Matrices], 頁 148 を参照してください。
初期値: 0

`warn_missing_semicolon`
: Section 13.1 [Defining Functions], 頁 87 を参照してください。
初期値: 0

`warn_neg_dim_as_zero`
: Section 18.3 [Special Utility Matrices], 頁 148 を参照してください。
初期値: 0

`warn_num_to_str`
: Section 5.3 [String Conversions], 頁 38 を参照してください。
初期値: 1

`warn_reload_forces_clear`
: Section 13.8 [Dynamically Linked Functions], 頁 95 を参照してください。
初期値: 1

`warn_resize_on_range_error`
: Section 10.1 [Index Expressions], 頁 61 を参照してください。
初期値: 0

`warn_separator_insert`
: Section 4.1 [Matrices], 頁 29 を参照してください。
初期値: 0

`warn_single_quote_string`
: Section 5.3 [String Conversions], 頁 38 を参照してください。
初期値: 0

`warn_str_to_num`
: Section 5.3 [String Conversions], 頁 38 を参照してください。
初期値: 0

`warn_undefined_return_values`
: Section 13.2 [Multiple Return Values], 頁 89 を参照してください。
初期値: 0

`warn_variable_switch_label`
: Section 12.2 [The switch Statement], 頁 77 を参照してください。
初期値: 0

9.5 環境変数からの初期値

Octave は，以下の環境変数を，対応する組み込み変数に対する初期値にセットします。さらに，環境変数からの値は，コマンドライン引数によって上書きされることになります。See Section 2.1.1 [Command Line Options], 頁 11. を参照してください。

`EDITOR`
: Section 2.4.5 [Commands For History], 頁 18 を参照してください。
組み込み変数: `EDITOR`

`OCTAVE_EXEC_PATH`
: Section 34.3 [Controlling Subprocesses], 頁 293 を参照してください。
組み込み変数: `EXEC_PATH` コマンドライン引数: `--exec-path`

`OCTAVE_PATH`
: Section 13.6 [Function Files], 頁 92 を参照してください。
組み込み変数: `LOADPATH` コマンドライン引数: `--path`

`OCTAVE_INFO_FILE`
: Section 2.3 [Getting Help], 頁 14 を参照してください。
組み込み変数: `INFO_FILE` コマンドライン引数: `--info-file`

`OCTAVE_INFO_PROGRAM`
: Section 2.3 [Getting Help], 頁 14 を参照してください。
組み込み変数: `INFO_PROGRAM` コマンドライン引数: `--info-program`

`OCTAVE_HISTSIZE`
: Section 2.4.5 [Commands For History], 頁 18 を参照してください。
組み込み変数: `history_size`

`OCTAVE_HISTFILE`
: Section 2.4.5 [Commands For History], 頁 18 を参照してください。
組み込み変数: `history_file`

10 式

Octave において，式は，ステートメント（文）の基本的な構成要素です。ある式は値へと評価され，表示，テスト，変数への格納，関数への受け渡し，あるいは代入演算子を用いて変数へ新しい値を割り当てたりすることができます。

ある式は，それ自身をステートメントとして提供することができます。大部分のその他のステートメントは，操作しようとするデータを指定する 1 つ以上の式を含んでいます。他の言語と同様に，Octave における式は，変数，配列，定数および関数呼び出し，さらにはそれらを様々な演算子で結びつけたものを含みます。

10.1 インデックス式

インデックス式を用いると，行列やベクトルの選択した要素を参照したり，抜き出したりできるようになります。

インデックスは，スカラ，ベクトル，範囲あるいは特殊な演算子 ':' をとることができます。この演算子は，全体の行あるいは列を選択するために使うことになるでしょう。

ベクトルは，単一の式を使用することによりインデックス化できます。行列は，1 つまたは 2 つのインデックスを使用することによりインデックス化できます（もし組み込み変数 `warn_fortran_indexing`の値がゼロでないならば，1 つのインデックスを指定したときに警告が発生します）。

`warn_fortran_indexing` [Built-in Variable]
: もし `warn_fortran_indexing`の値がゼロでなければ，単一のインデックスを用いて 2 次元行列の要素を選択する式に対して，警告を表示する。標準状態では 0 である

以下の行列について，

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

以下の式は，どれも同じ意味である。

```
a (1, [1, 2])
a (1, 1:2)
a (1, :)
```

これらは，行列の最初の行を選択している。

1 を含むベクトルでスカラをインデックス化すると，各要素がもとのスカラの値に等しく，インデックスベクトルと同じサイズのベクトルを生成することができます。たとえば，以下の式は，

```
a = 13;
a ([1, 1, 1, 1])
```

4 つの要素がすべて 13 であるベクトルを生成します。

同様に，1 を含む 2 つのベクトルでスカラをインデックス化すると，行列を生成することができます。たとえば，以下の式は，

```
a = 13;
a ([1, 1], [1, 1, 1])
```

すべての要素が 13 であって，2 行 3 列の行列を生成します。

これは分かりにくい表記であって，避けるべきです。すべての要素が 1 である適切な大きさの行列を生成するために `ones`関数を使用し，望む結果を生み出すためにそれをスケール化する方がよいのです。Section 18.3 [Special Utility Matrices], 頁 148 を参照してください。

`warn_resize_on_range_error` [Built-in Variable]
もし `warn_resize_on_range_error`がゼロでないならば，行列の添え字の範囲を超えたインデックスを割り当ててサイズ変更するときに警告を表示する。標準状態では 0 である。

上の例で示した式のように，ループを使用してベクトルを生成することは，完全に非効率です。このような場面では，以下のような式を使用することが，ずっと効率的です。

```
a = sqrt (1:10);
```

この式は，完全にループを避けています。それでもループが必要となる場面，あるいは，値の数が大きな行列を形成するために結びつけられなければならない場面においては，最初に行列のサイズをセットし，その後にインデックスコマンドを用いて要素を挿入する方がずっと高速です。たとえば，行列 `a`が与えられているとして，

```
[nr, nc] = size (a);
x = zeros (nr, n * nc);
for i = 1:n
  x(:,(i-1)*nc+1:i*nc) = a;
endfor
```

この式は，特に大きな行列において以下の式よりもかなり高速です。

```
x = a;
for i = 1:n-1
  x = [x, a];
endfor
```

なぜならば，Octave は，結果を繰り返しサイズ変更する必要がないためです。

10.2 関数の呼び出し

関数は，特定の計算手順に対する名称です。関数には名前がついているので，プログラムの任意の場所でそれを呼び出すことができます。たとえば，`sqrt`関数は，ある数の平方根を計算します。

あらかじめ用意されている関数群は，組み込み関数です。これは，どの Octave プログラムにおいても利用可能であることを意味します。`sqrt` 関数は，組み込み関数のひとつです。さらに，あなた独自の関数を定義することもできます。これを実行するためのさらなる情報は，Chapter 13 [Functions and Scripts], 頁 87 を参照してください。

関数を使用するための方法は，関数呼び出し式を使うことです。この式は，関数名と，それに続くかっこでくくられた引数のリストから構成されます。その引数は，関数が実行する計算に用いる生の対象物を与える式です。1 つ以上の引数があるとき，それらはカンマで区切ります。もし引数がなければ，カッコを省略することができます。しかし，関数の呼び出しを行ったということを明確にするため，いつでもカッコを使うことがよい考えです。いくつかの例を挙げます。

```
sqrt (x^2 + y^2)      # 1 個の引数
ones (n, m)           # 2 個の引数
rand ()               # 引数なし
```

各々の関数ごとに，とるべき引数の数が決まっています。たとえば `sqrt` 関数は，1 個の引数（平方根をとるべき数）をつけて呼び出さなければなりません。

```
sqrt (argument)
```

組み込み関数の中には，特定の利用法に応じて引数の数を変えるものがあります。また，その関数の挙動は，与えた引数の数によって異なります。

他の式と同じように，関数の呼び出しは値です。これは与えた引数に基づいて，その関数により計算されたものです。この例において，`sqrt (`*`argument`*`)`の値は，引数の平方根です。関数は，ある変数に値を代入したり，入出力を実行したりするような使い方もできます。

大部分の言語とは異なり，Octave における関数は，複数の値を返すことがあります。たとえば，

```
[u, s, v] = svd (a)
```

この式は，行列 aの特異値分解を計算し，3 つの計算結果の行列を u，sおよび vに代入します。

複数の代入式の左辺は，それ自体が式になっており，変数名やインデックス式のリストをとることができます。Section 10.1 [Index Expressions], 頁 61 および Section 10.6 [Assignment Ops], 頁 68 も参照してください。

10.2.1 値による呼び出し

Octave では，Fortran とは異なり，関数の引数は値として渡されます。これは，関数呼び出しにおける各々の引数は，関数に渡される前に評価され，メモリの一時領域に割り当てられます。現在のところ，引数を値ではなく参照渡しとするように指定する方法はありません。これは，関数呼び出しにおいて，引数の値を直接変更することは不可能だということです。関数内では，一時的にコピーされた値のみを変更することはできます。たとえば，以下の関数があります。

```
function f (x, n)
  while (n-- > 0)
    disp (x);
  endwhile
endfunction
```

この関数は，1 番めの引数の値を n 回繰り返して表示するものです。この関数において，変数 n は，その値が関数の呼び出し元で変更されることを心配する必要なく，一時変数として使用されています。その関数が引数を修正しようとしないことを最初に決定する必要なく，どの引数に対しても定数として渡すことが常に可能となるため，値渡しは役に立ちます。

呼び出す側は，引数に対する式として変数を使用するでしょうが，呼び出される関数はこれを知りません。ただ引数のもつ値だけを知るのです。たとえば，以下のように呼び出される関数があるとします。

```
foo = "bar";
fcn (foo)
```

あなたは，引数が「fooという変数」であると考えるべきではありません。かわりに，引数が"bar"という文字列値であると考えるべきです。

Octave は，関数の引数に値渡しを使用していますが，値は必要のない時にはコピーされません。たとえば，

```
x = rand (1000);
f (x);
```

この例では，もし関数 fがその引数の値を変更しないのであれば，2 つの 1000 行 1000 列の行列を，実際には存在しないようにします。ですから，Octave は関数 fの範囲外で値を変更することを避けるためにコピーを作らなければなりません。そうでないとすれば，定数または一時的な結果の値を修正しようとします（そして失敗します）。

10.2.2 再帰

いくつかの制限[1] はありますが，関数の再帰呼び出しが許されています。再帰関数は，直接・間接にそれ自身を呼び出す関数のことです。たとえば，与えられた整数の階乗を計算する非効率な[2] 例を示します。

```
function retval = fact (n)
  if (n > 0)
    retval = n * fact (n-1);
  else
    retval = 1;
  endif
endfunction
```

この関数は，それ自身を直接呼び出す再帰的関数です。この関数は，自分自身を呼び出すたびに，以前の呼び出しで用いた値よりも 1 だけ小さい引数を使用しているので，最終的には終了します。いちど引数がゼロ以下になれば，この関数は自分自身を呼び出さずに，再帰は終了します。

組み込み変数 `max_recursion_depth`は，再帰の深さの限界を指定し，Octave が無限回の再帰にはまることを回避します。

`max_recursion_depth` [Built-in Variable]
関数が再帰的に呼び出される回数の限界を決定する。もし限界を超えるならば，エラーメッセージを表示し，トップレベルに制御を戻す。

初期値では 256 である。

10.3 算術演算子

以下の算術演算子が利用でき，スカラと行列に対して動作する。

`x + y` 加算します。もし両方のオペランドが行列ならば，行と列の数の両方とも一致していなければなりません。もし片方のオペランドがスカラならば，その値を他方のオペランドの全要素に加算します。

`x .+ y` 要素どうしの加算です。この演算子は`+`と等価です。

`x - y` 減算します。もし両方のオペランドが行列ならば，行数と列数の両方とも一致していなければなりません。

`x .- y` 要素どうしの減算です。この演算子は`-`と等価です。

`x * y` 行列の乗算です。x の列数は，y の行数と一致していなければなりません。

`x .* y` 要素どうしの乗算です。もし両方のオペランドとも行列ならば，行数と列数の両方とも一致していなければなりません。,

`x / y` 右除算です。これは，概念的には，以下の式と等価です。

---

[1] Octave の関数の中には，再帰的に呼び出すことのできない関数として実装されているものがあります。たとえば，ODE ソルバ `lsode` は，結局のところ，再帰呼び出しのできない Fortran サブルーチンで実装されています。したがって，`lsode`を必要とするユーザ提供関数を，その内部から直接・間接を問わず呼び出すべきではありません。そのようなことを行うと，予期しない挙動を起こします。

[2] 値 `n`が実際に正の値をとることをチェックした後に，`prod (1:n)`または `gamma (n+1)`を使用することがよいでしょう。

```
(inverse (y') * x')'
```

しかし，この演算子は *y'* の逆数（逆行列）を形成することなく計算します。

もし方程式が正方でない，あるいは係数行列が特異ならば，最小ノルム解を計算します。

`x ./ y` 要素ごとの除算です。

`x \ y` 左除算です。これは，概念的には，以下の式と等価です。

```
inverse (x) * y
```

しかし，この演算子は *x'* の逆数（逆行列）を形成することなく計算します。

もし方程式が正方でない，あるいは係数行列が特異ならば，最小ノルム解を計算します。

`x .\ y` 要素ごとの左除算です。*y* の各々の要素を，*x* の各々の対応する要素で割ります。

`x ^ y`
`x ** y` ベキ乗の演算子です。もし *x* と *y* の両方ともスカラならば，この演算子は *x* の *y* 乗を返します。もし *x* がスカラであり，*y* が正方行列ならば，その結果は固有値展開を用いて計算されます。もし *x* が正方行列ならば，*y* が整数のときには repeated multiplication によって，*y* が整数でないときには固有値展開によって計算します。*x* と *y* の両方とも行列ならば，エラーとなります。

この演算子の実装は，改善される必要があります。

`x .^ y`
`x .** y` 要素ごとのベキ乗演算子です。もし両方のオペランドとも行列ならば，その行数と列数が一致していなければなりません。

`-x` 負値です。

`+x` 正値です。この演算子は，オペランドに何もしません。

`x'` 複素共役転置です。実数の引数に対しては，この演算子は転置演算子と同じです。複素数の引数に対しては，この演算子は式

```
conj (x.')
```

と等価です。

`x.'` 転置です。

Octave の要素ごとの演算子は，‘.’から始まるので，以下のような式はあいまいになる可能性があることに注意してください。

```
1./m
```

なぜならば，ピリオドは，定数の一部あるいは演算子の一部のどちらにも解釈できるためです。この衝突を避けるために，Octave は，型を持っているかのように式を扱います。つまり，

```
(1) ./ m
```

であって，

```
(1.) / m
```

ではありません。これは Octave の文法解釈の普通の挙動，すなわち，通常はある与えられた点において最も長くマッチするように入力をトークンに分割すること，とは相容れないのですが，この場合には上記の解釈の方が役に立ちます。

`warn_divide_by_zero` [Built-in Variable]

もし `warn_divide_by_zero`が 0 でないならば，Octave がゼロで除算したときに警告が発生する。もしこの値が 0 ならば，警告を省略する。標準状態は 1 である。

10.4 比較演算子

比較演算子は,（たとえば等しいというような）数値の関係を比較します。これらは関係演算子を用いて書かれています。

Octave のすべての比較演算子は，比較が真であれば 1 を，偽であれば 0 を返します。行列の値について，これらは要素ごとに基づいて動作します。たとえば，以下のようになります。

```
[1, 2; 3, 4] == [1, 3; 2, 4]
     ⇒  1  0
        0  1
```

もし片方のオペランドがスカラであり，他方が行列ならば，このスカラを行列の各々の要素と順に比較して，結果はその行列と同じサイズになります。

`x < y` x が y よりも小さいならば真です。

`x <= y` x が y と同じか小さい（以下）ならば真です。

`x == y` x が y と等しいならば真です。

`x >= y` x が y と同じか大きい（以上）ならば真です。

`x > y` x が y よりも大きいならば真です。

`x != y`
`x ~= y`
`x <> y` x が y と等しくないならば真です。

文字列の比較は，上で示した比較演算子ではなく，`strcmp`関数により実行することができます。Chapter 5 [Strings], 頁 35 を参照してください。

10.5 ブール演算子

10.5.1 要素ごとのブール演算子

要素ごとの論理（ブール）式は，ブール演算子 “or”（‘|’），“and”（‘&’）および “not”（‘!’）と，入れ子をコントロールするための括弧を用いた比較式の組み合わせです。ブール式の真理値は，成分式の対応する要素の真理値の組み合わせによって計算されます。ある値がゼロならば偽であり，それ以外は真であると見なします。

ようそごとのブール式は，使用できる比較式であれば，何でも使えます。これらは，`if`および `while` ステートメントにおいても利用できます。しかし，もし `if`または `while`ステートメントにおいて条件として行列を使用するならば，その要素のすべてがゼロでないときに限り真になります。

比較演算子のように，要素どうしのブール式の各々の要素は数値を持ちます（真ならば 1，偽ならば 0)。これは，ブール式の結果が変数に格納されたときには，数値になり，ふつうの値として使用されます。

要素どうしのブール演算子についての解説です。

boolean1 `&` *boolean2*
: もし *boolean1* と *boolean2* の対応する要素の両方が真ならば，結果の要素は真となります。

boolean1 `|` *boolean2*
: もし *boolean1* と *boolean2* の対応する要素のどちらかが真ならば，結果の要素は真となります。

`!` *boolean*
`~` *boolean*
もし *boolean* の要素が偽ならば，結果の要素は真となります。

オペランドに行列を指定したとき，これらの演算子は要素ごとに働きます。たとえば，

```
[1, 0; 0, 1] & [1, 0; 2, 3]
```

この式は，2 行 2 列の単位行列を返します。

二項演算子に対して，両方のオペランドが行列ならば，オペランドの次元はそろっていなければなりません。もしオペランドの片方がスカラであり，もう片方が行列ならば，この演算子は行列の各要素にスカラを適用します。

要素ごとの二項ブール演算子について，式 *boolean1* と *boolean2* の両方とも，結果を計算する前に評価されます。このことは，その式が別の効果をもつときに異なる結果を生むことになります。たとえば，

```
a & b++
```

この式は，たとえ変数 *a* がゼロであっても，*b* はインクリメントされます。

この挙動は，ブール演算子が，行列のオペランドで説明したように動作するために必要なことです。

10.5.2 短絡回路ブール演算子

`if`および `while`の条件において，スカラへの暗黙的な変換と結合して，Octave の要素ごとのブール演算子は，しばしば大部分の論理演算子を実行するには十分です。しかし，全体の真理値を決定することができた時点で，ただちにブール式の評価をやめることが望ましいことがあります。Octave の短絡回路ブール演算子は，そのように動作します。

boolean1 `&&` *boolean2*
`all(all (`*boolean1*`))`と等価な操作を利用して，式 *boolean1* は評価され，スカラに変換されます。もしそれが偽ならば，式全体は 0 という結果になります。もしそれが真ならば，`all (all (`*boolean2*`))`と等価な操作を利用して，式 *boolean2* は評価され，スカラに変換されます。もしこれが真ならば，式全体の結果は 1 であり，そうでなければ全体の式の結果は 0 になります。

boolean1 `||` *boolean2*
`all (all (`*boolean1*`))`と等価な操作を利用して，式 *boolean1* は評価され，スカラに変換されます。もしそれが真ならば，式全体は 1 という結果になります。もしそれが偽ならば，`all (all (`*boolean2*`))`と等価な操作を利用して，式 *boolean2* は評価され，スカラに変換されます。もしこれが真ならば，全体の式の結果は 1 であり，そうでなければ式全体の結果は 0 になります。

両方のオペランドが，式の全体の真理値を決定する前に評価されないことがあるという事実は，重要に成り得ます。たとえば，

```
a && b++
```

この式は，変数 *b* の値は，変数 *a* がゼロでないときにだけインクリメントされます。

これは，いくぶん簡潔なコードを書くのに使えるでしょう。たとえば，

```
function f (a, b, c)
  if (nargin > 2 && isstr (c))
    ...
```

このコードは，存在しない引数を評価しようとするのを避けるために，2 つの ifステートメントを使わねばならないところに使うことができます。たとえば，短絡回路機能を使わないとすれば，以下のように書く必要があります。

```
function f (a, b, c)
  if (nargin > 2)
    if (isstr (c))
      ...
```

以下のコード

```
function f (a, b, c)
  if (nargin > 2 & isstr (c))
    ...
```

は，fを 1 個または 2 個の引数をつけて呼び出すと，エラーになります。なぜならば，Octave は，演算子 '&'に関する 2 つのオペランドを両方とも評価しようとするからです。

10.6 代入式

代入は，ある新たな値をある変数に格納する式です。たとえば，以下の式は，1 という値を変数 zに代入します。

```
z = 1
```

この式を実行した後，変数 zは値 1 を持ちます。zが代入の前に持っていた古い値は，どんなものも忘れてしまいます。イコール '='記号は，代入演算子と呼ばれています。

文字列値も代入することができます。たとえば，以下の式は，変数 message に"this food is good"という値を格納することになります。

```
thing = "food"
predicate = "good"
message = [ "this " , thing , " is " , predicate ]
```

（これは，文字列の連結を示してもいます）

大部分の演算子（加算や連結など）は，値を計算すること以外の作用はありません。もしその計算結果を無視するなら，演算子を使うことはないでしょう。でも，代入演算子は違います。これは値を生み出しますが，たとえその値を無視したとしても，代入は，値の改変を通して関知されています。これを副作用と読んでいます。

代入の左辺オペランドは，変数（see Chapter 9 [Variables], 頁 51）である必要はありません。これには，行列の要素（see Section 10.1 [Index Expressions], 頁 61）あるいは返り値のリスト（see Section 10.2 [Calling Functions], 頁 62）をとることができます。これらはすべて *lvalues* と呼ばれており，代入演算子の左辺に現れることを意味しています。右辺のオペランドは，どのような式にもなります。これは特定の変数や行列，戻り値のリストに格納される新しい値を生み出します。

変数は，もともとの型を持っていないことに注意することは重要です。変数の型は，単にその時点に保持することになったどんな値の型をも取ります。以下のプログラムの一部分において，変数 fooは，最初は数値をもち，その後は文字列値をもちます。

```
octave:13> foo = 1
foo = 1
octave:13> foo = "bar"
foo = bar
```

2 番めの代入によって fooに文字列を与えるとき，以前に数値を持っていたということを忘れてしまいます。

インデックス付き行列にスカラを代入すると，インデックスで参照した要素のすべてにスカラがセットされます。たとえば，もし aが少なくとも 2 つの列をもつ行列であるならば，

```
a(:, 2) = 5
```

この式は，aの 2 列めのすべての要素に 5 をセットします。

空行列 '[]' を代入することは，大部分の場面において，行列またはベクトルの行あるいは列を削除できるように動作します。Section 4.1.1 [Empty Matrices], 頁 32 を参照してください。たとえば，4 行 5 列の行列 A が与えられるとき，

```
A (3, :) = []
```

この代入式は，A の 3 行めを削除します。また，

```
A (:, 1:2:5) = []
```

この代入文は，1 列め，2 列めおよび 5 列めを削除します。

代入は式ですから，値をもちます。したがって，式としての z = 1は，値 1 をもちます。この結果にひとつとして，複数の代入式を一度に書くことができます。

```
x = y = z = 0
```

この式は，すべての 3 つの変数に値 0 を格納します。この動作になるのは，z = 0の値は 0 であり，これを yに代入し，さらに y = z = 0 の値 0 が xに代入されるためです。

このことは，値のリストを代入するときにも当てはまります。でうから，以下の式は妥当です。

```
[a, b, c] = [u, s, v] = svd (a)
```

これは，以下の式と完全に等価です。

```
[u, s, v] = svd (a)
a = u
b = s
c = v
```

このような表記において，式の各部分の値の数はそろえる必要がありません。たとえば，

```
[a, b, c, d] = [u, s, v] = svd (a)
```

この式は，変数 'd'の値は変化しないままであることを除いて，上の式と等価です。また，

```
[a, b] = [u, s, v] = svd (a)
```

この式は，以下の式と等価です。

```
[u, s, v] = svd (a)
a = u
b = s
```

式を呼び出したどこの場所でも，代入文を使用することができます。たとえば，yに 1 をセットした後で xが 1 に等しいかどうかをテストするために，x != (y = 1)と書くことは妥当です。しかし，この書き方は，プログラムを読みづらくします。使い捨てのプログラムを除き，このような代入の入れ子を排除するように書き換えるべきです。これは決して難しいことではありません。

print_rhs_assign_val [Built-in Variable]

もしこの変数の値がゼロでないならば，Octave は，代入後に左辺の値を表示する代わりに，代入式の右辺の値を表示するようになります。

10.7 インクリメント演算子

インクリメント演算子は，ある変数の値を 1 だけ増やしたり減らしたりします。変数をインクリメントする（1 だけ増やす）ための演算子は ‘++’と書きます。この演算子は，その値を参照する前または後に変数をインクリメントするために使用できます。

たとえば，変数 x を参照する前にインクリメントするには，`++x` と書きます。これは，x に 1 を加え，その後で式の結果として x の新しい値を返します。これは，式 `x = x + 1`と全く同じです。

変数 x を参照後にインクリメントするためには，`x++`と書くことになります。これは変数 x に 1 を加えますが，x をインクリメントする前の値を返します。たとえば，もし x が 2 に等しいならば，式 `x++`の結果は 2 であり，x の新しい値は 3 です。

行列またはベクトルを引数にすると，インクリメントおよびデクリメント（1 だけ減らす）は，オペランドの各々の要素に対して作用します。

すべてのインクリメントおよびデクリメント式のリストを示します。

`++x`　この式は，変数 x をインクリメントします。この式の値は，x の新しい値です。これは，`x = x + 1`と等価です。

`--x`　この式は，変数 x をデクリメントします。この式の値は，x の新しい値です。これは，`x = x - 1`と等価です。

`x++`　この式は，変数 x をインクリメントします。この式の値は，x の元の値です。

`x--`　この式は，変数 x をデクリメントします。この式の値は，x の元の値です。

インデックス式をインクリメントすることは，現在のところ可能ではありません。たとえば，式 `v(4)++`は，ベクトル v の 4 番めの要素をインクリメントすることを期待するでしょう。しかし，かわりにパースエラーを起こします。この問題は，Octave の将来のリリースにおいて修正されるでしょう。

10.8 演算子の優先順位

演算子の優先順位は，1 つの式の中にさまざまな演算子が近い位置に現れたとき，どのように演算子をグループにするかを決定します。たとえば，‘*’ は ‘+’よりも高い順位です。ですから，式 `a + b * c`は，`b`と `c`を掛けて，その後で `a`を加えることを意味します（つまり，`a + (b * c)`）。

かっこを使用することによって，演算子の順序を覆すことができます。優先順位とは，かっこ自身を書かないときに，かっこを仮定する位置であると言えると思ってください。事実，あまり使わない演算子の組み合わせをとるところでは，いつでもかっこを使うことが賢明です。なぜならば，そのプログラムを読む人々は，その場合の計算順位を覚えていないかもしれないからです。完全に忘れてしまえば，間違いを起こすことになります。かっこを明記することは，このようなミスを抑える助けになるでしょう。

同じ優先順位の演算子をともに使用するとき，最も左の演算子は，代入とベキ乗演算子（これらは逆向きにグループ化します）を除いて，最初にグループを作ります。ですから，式 `a - b + c`は，`(a - b) + c`のようにグループ化しますが，式 `a = b = c`は，`a = (b = c)`のようにグループ化します。

前置する単項演算子の順位は，別の演算子がオペランドの後ろに来るときに重要です。たとえば，`-x^2`は`-(x^2)`という意味になります。なぜならば，‘-’は ‘^’よりも順位が低いためです。

Octave における演算子の表を示します。下に行くほど順位が高くなるようになっています。

ステートメント演算子
　‘;’, ‘,’.

代入　‘=’. （この演算子は右から左へとグループ化します）

論理 ‘‘or’’ および ‘‘and’’
‘||’, ‘&&’.

要素ごとの ‘‘or’’ および ‘‘and’’
‘|’, ‘&’.

関係 ‘<’, ‘<=’, ‘==’, ‘>=’, ‘>’, ‘!=’, ‘~=’, ‘<>’.

コロン ‘:’.

加算と減算 ‘+’, ‘-’.

乗算と除算 ‘*’, ‘/’, ‘\’, ‘.\’, ‘.*’, ‘./’.

転置 ‘’’, ‘.’’

単項のプラス，マイナス，インクリメント，デクリメントおよび ‘‘not’’
‘+’, ‘-’, ‘++’, ‘--’, ‘!’, ‘~’.

ベキ乗 ‘^’, ‘**’, ‘.^’, ‘.**’.

11 評価

通常，Octaveのプロンプトにおいて単に式を入力することにより，それを評価します。あるいは，ファイルに保存しておいたコマンドが解釈できるかをOctaveに問い合わせることによって評価することもできます。

ときどき，計算されて文字列に格納された文字列を評価すること，あるいは呼び出すべき関数名を文字列として使用する必要があると思うかもしれません。`eval`および`feval`関数は，それを行えるようにします。または，実行時には知られていないコマンドを評価するため，あるいはユーザ提供関数を呼び出す必要のある関数を書くために必要です。

`eval` (*try*, *catch*) [Built-in Function]

文字列 *try* を解釈し，それがOctaveのコマンドであるかのように評価する。もしそれが失敗するならば，文字列 *catch* を評価する。文字列 *try* は現在の状態で評価され，結果は`eval`が戻った後に入手できるようになったままである。

`feval` (*name*, ...) [Built-in Function]

name という名前をもつ関数を評価します。1番め以降の引数は，いずれもその名前の関数に渡されます。たとえば，

```
feval ("acos", -1)
     ⇒ 3.1416
```

この式は，引数‘`-1`’をつけて関数`acos`を呼び出す。

関数`feval`は，ユーザ提供関数を呼び出すような関数を書くことができるようにするために必要である。なぜならば，Octaveは（C言語のような）関数へのポインタを宣言したり，(Fortranの`EXTERNAL`のような）関数名を保持するために使用される特殊な変数を宣言するための方法をもたないからである。かわりに，名前によって関数を参照し，それを呼び出すために`feval`を使用しなければならない。

`feval`を使用し，ニュートン法を用いてある変数のユーザ提供関数の根を見いだす単純な関数を示します。

```
function result = newtroot (fname, x)

# 使用法: newtroot (fname, x)
#
#   fname : 関数 f(x) の名前を含む文字列
#   x     : 初期値

  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
```

この例は，単にユーザ提供関数を呼び出す一例という意味でしかなく，ニュートン法を真剣に理解しろということではないことに気を付けてください。本格的なコードを書くならば，より頑健なアルゴリズムを使用することに加えて，数値とすべての引数の型，および，与えられる関数が本当に関数なのかなどをチェックすることになります。たとえば，数値オブジェクトの判定関数のリストは Section 4.4 [Predicates for Numeric Objects], 頁 33 を，`exist`関数の説明については，Section 9.3 [Status of Variables], 頁 53 を参照してください。

12 ステートメント

ステートメントは，単純な定数式，あるいは入れ子になったループや条件式の複雑なリストになるでしょう。

`if`や`while`などのような制御ステートメントは，Octave プログラムにおける実行の流れを制御します。すべての制御ステートメントは，単純な式と区別するために，`if`や`while`のような特別なキーワードで開始します。多くの制御ステートメントは，他のステートメントを含んでいます。たとえば，`if`ステートメントは，実行されたりされなかったりする他のステートメントを含みます。

各々の制御ステートメントには，その制御ステートメントの終了を明記するような，対応する終了ステートメントがあります。たとえば，`endif` というキーワードは，`if`ステートメントの終端をマークし，`endwhile`は `while`ステートメントの終端をマークします。キーワード `end`は，より具体的な終了キーワードが期待できるようなところではどこでも使用することができます。しかし，より具体的な終了キーワードを使用するほうが好ましいのです。なぜならば，それを使用すると，Octave は，ミスマッチや end 句の付け忘れを診断しやすくなるからです。

`if`や`while`のようなキーワードと，それに対応する終端ステートメントの間に含まれる一連のステートメントは，本文（body）と呼ばれます。

12.1 `if`ステートメント

`if`ステートメントは，Octave の意志決定を行うステートメントです。`if`ステートメントには，3 種類の基本的な使い方があります。その最も単純な使い方は，以下のようなものです。

```
if (condition)
  then-body
endif
```

condition は，ステートメントの残り部分が実行するであろうことを制御する式です。*then-body* は，*condition* が真であるときのみ実行されます。

`if`ステートメントにおける条件は，その値がゼロでないならば真，その値がゼロならば偽であると見なされます。もし `if`ステートメントの条件式の値がベクトルまたは行列ならば，その要素のすべてがゼロでない場合のみ真であると見なされます。

`if`ステートメントの 2 番めの使い方は，次のようなものです。

```
if (condition)
  then-body
else
  else-body
endif
```

もし *condition* が真ならば，*then-body* を実行します。そうでないならば，*else-body* を実行します。

例を示します。

```
if (rem (x, 2) == 0)
  printf ("x is even\n");
else
  printf ("x is odd\n");
endif
```

この例において，もし式 `rem (x, 2) == 0`が真ならば（すなわち，`x` が 2 で割り切れるならば），最初の `printf`ステートメントが評価され，そうでないならば 2 番めの `printf`ステートメントが評価されます。

ifステートメントの第 3 の，そして最後の使い方は，複数の決定事項が，ひとつのステートメントに結びつけられているようにします。それは，以下のようなものです。

```
if (condition)
  then-body
elseif (condition)
  elseif-body
else
  else-body
endif
```

任意の数の elseif節を入れてもかまいません。各々の条件は，順にテストされ，もしある条件が真であるならば，それに対応する *body* を実行します。もしどの条件も真でなく，else節が存在するならば，その本文を実行します。else句は 1 つだけ入れることができ，それはステートメントの最後の部分に置かなければなりません。

以下のサンプルにおいて，もし最初の条件が真であれば（すなわち xの値が 2 で割り切れるならば），最初の printfステートメントが実行されます。もしそれが偽ならば 2 番めの条件が検定され，それが真ならば（すなわち x の値が 3 で割り切れるならば），2 番めの printfステートメントが実行されます。どれでもないならば，3 番めの printfステートメントを実行します。

```
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
```

elseifというキーワードは，Fortran では許容されているような else ifと綴ってはいけません。もしそうするならば，elseと ifの間のスペースは，Octave は，別の if内に新しい ifがあるように扱うように命令します。たとえば，もし次のように書くならば，

```
if (c1)
  body-1
else if (c2)
  body-2
endif
```

Octave は，最初の ifステートメントを完了するために，追加入力を期待するでしょう。もし Octave を対話的に使用しているならば，追加的な入力のためにプロンプトを出し続けるでしょう。もし Octave がこの入力をファイルから読み込んでいるならば，endステートメントが欠損しているかミスマッチである旨のエラーを吐くでしょう。あるいは，より具体的な endステートメント（endif，endforなど）を使っていないとすれば，何の警告メッセージを表示することなく，単に正しくない結果を生み出すでしょう。

上のステートメントを以下のように書き直すと，エラーを見つけるのがずっと容易になります。

```
if (c1)
  body-1
else
  if (c2)
    body-2
  endif
```

Octaveがステートメントをどのようにグループ化するのかを示すための字下げをしてあります。Chapter 13 [Functions and Scripts], 頁 87 を参照してください。

warn_assign_as_truth_value [Built-in Variable]

もし warn_assign_as_truth_valueの値がゼロでないならば，以下のようなステートメントについて警告を発する。

```
if (s = t)
  ...
```

なぜならば，このようなステートメントは普通には起こりえず，かわりに以下のように書きたかったように思われるからである。

```
if (s == t)
  ...
```

whileあるいは ifステートメントの条件内に代入演算子を含むコードを書く場合に有効である。たとえば，以下のようなコードは，C 言語ではごくふつうに使われる。

```
while (c = getc())
  ...
```

warn_assign_as_truth_valueに 0 をセットすることにより，このようなステートメントについての全ての警告を回避することが可能である。しかし，そのようなステートメントは，以下のような本当のエラーを見逃すことにもなる。

```
if (x = 1)  # intended to test (x == 1)!
  ...
```

このような場面において，さらにもうひと組のかっこを付けて書くことにより特定のステートメントに対してエラーを抑えることが可能である。たとえば，以前のサンプルを，

```
while ((c = getc()))
  ...
```

と書くことにより，このステートメントに対する警告を表示しないようにするだろう。一方で，条件文において使用される他の代入について，警告を発するようになる。

warn_assign_as_truth_valueの初期値は 1 である。

12.2 switchステートメント

switchステートメントは，Octave のバージョン 2.0.5 で導入されました。これは実験的なものであると考えるべきですし，実装の詳細は Octave の将来のバージョンでわずかに変更される可能性があります。もし実際のプログラムにおいて，この新しいコマンドを試そうとしたときの経験を共有したいと思ったり，コメントがあるならば，maintainers@octave.orgに寄せてくだされば幸いです（でも，もしバグを発見したと思ったならば，bug@octave.org に報告してください）。

switchステートメントの一般的な形式は，以下のようになります。

```
switch expression
  case label
    command_list
  case label
    command_list
  ...

  otherwise
    command_list
endswitch
```

- 識別子 switch，case，otherwiseおよび endswitch はキーワードになっています。
- *label*（ラベル）には任意の式を置くことができます。
- 重複する *label* は認められません。最初にマッチした *label* に対応する *command_list* が実行されます。
- 少なくとも 1 つの case *label command_list* 句がなければなりません。
- otherwise *command_list* 句は，あってもなくてもかまいません。
- 全ての他の endキーワードと同じように，endswitchも endに置き換えることができます。しかし，endswitchを使用する構文を使うと，より解釈されやすくなります。
- 場合分けは排他的なので，C 言語の switch ステートメントの場合のような「下に素通りする」ようなことはありません。
- *command_list* 要素はオプションではありません。リストをオプショナルにすることは，ラベルとコマンドリストの間に区切りが必要であるという意味になります。一方，以下のような書き方について，

  ```
  switch (foo)
    case (1) -2
    ...
  ```

  これは，特に C 言語のプログラマにとって，驚くべき結果になります。以下のコードと同じ挙動をします。

  ```
  switch (foo)
    case (1)
    case (2)
      doit ();
    ...
  ```

- この実装は単純指向であり，現在のところ等価な ifブロックを上回る実性能を提供するわけではありません。たとえ，全てのラベルが整数の定数であったとしても，です。おそらく，これにおける将来のバリエーションは，全ての定数整数ラベルを検出し，ジャンプ表による性能の向上ができるようになるでしょう。

warn_variable_switch_label [Built-in Variable]

If the value of this variable is nonzero, Octave will print a warning if a switch label is not a constant or constant expression

12.3 whileステートメント

プログラミングにおいて，ループは，連続して 2 回以上実行される（あるいは，少なくとも実行される可能性がある）プログラムの一部分を意味します。

whileステートメントは，Octave における最も単純なループ実行ステートメントです。これは，条件が真である限り，そのステートメントを繰り返し実行します。ifステートメントにおける条件と同様に，whileにおける条件は，その値がゼロでないときに真，ゼロのときに偽であると見なされます。もし whileステートメントにおける条件式の値がベクトルまたは行列であるならば，要素のすべてがゼロでないときのみ，真であると見なされます。

Octave の whileステートメントは，以下のように見えます。

```
while (condition)
  body
endwhile
```

ここで，*body* はループの本体と呼ばれるステートメント（またはそのリスト）であり，*condition* は，そのループがどのくらいのあいだ実行を維持するかを制御する式です。

whileステートメントが行う最初のことは，*condition* をテストすることです。もし *condition* が真ならば，*body* 部分を実行します。*body* がを実行した後，*condition* が再びテストされ，これがまだ真であれば *body* を再度実行します。この過程を，*condition* が真でなくなるまで繰り返します。もし *condition* が最初に偽であれば，ループの本体を一度も実行しません。

この例では，フィボナッチ数列の 10 番めまでの要素を含む変数 fibを生成します。

```
fib = ones (1, 10);
i = 3;
while (i <= 10)
  fib (i) = fib (i-1) + fib (i-2);
  i++;
endwhile
```

ここでループの本体には，2 つのステートメントを含みます。

このループは以下のように動作します：まず，iの値を 3 にセットします。その後，whileは，iが 10 以下であるかどうかをテストします。この場合は iが 3 に等しいので，fibの i番めの要素の値には，直前の連続する 2 値の和をセットします。その後，i++が iの値をインクリメントし，ループを繰り返します。このループは，iが 11 に達するときに終了します。

条件と本体の間に，改行は必要ありません。しかし，本体が非常に単純でない場合は，改行を入れるとプログラムが読みやすくなります。

変数 warn_assign_as_truth_valueの説明については，Section 12.1 [The if Statement], 頁 75 を参照してください。

12.4 do-untilステートメント

do-untilステートメントは，条件が真になるまであるステートメントを繰り返し実行することを除き，whileステートメントと同様です。その結果，ループの本体は，少なくとも一度は必ず実行されます。ifステートメントの条件と同じように，do-untilステートメントにおける条件は，その値がゼロでないならば真，ゼロならば偽になります。do-untilステートメントにおける条件式の値がベクトルまたは行列ならば，その要素のすべてがゼロでないときに限り真であると見なされます。

Octave の do-untilステートメントは，以下のように見えます。

```
do
  body
until (condition)
```

body は，ループの本体と呼ぶ（単数または複数の）ステートメントであり，*condition* は，ループがどのくらい続くかをコントロールする式です。

この例では，フィボナッチ数列の 10 番めまでの要素を含む変数 fibを生成します。

```
fib = ones (1, 10);
i = 2;
do
  i++;
  fib (i) = fib (i-1) + fib (i-2);
until (i == 10)
```

条件と本体の間に，改行は必要ありません。しかし，本体が非常に単純でない場合は，改行を入れるとプログラムが読みやすくなります。

変数 warn_assign_as_truth_valueの説明については，Section 12.1 [The if Statement], 頁 75 を参照してください。

12.5 forステートメント

forステートメントは，ループの反復回数をカウントすることを容易にしてくれます。forステートメントの一般的な型は，以下のように見えます。

```
for var = expression
  body
endfor
```

ここで *body* は，何らかのステートメントやステートメントのリストを表し，*expression* は何らかの妥当な式です。また，*var* はいくつかの型を取ることができます。通常，その変数は単純な変数名あるいはインデックス付きの変数です。もし *expression* の値が構造体ならば，*var* もリストになります。以下の Section 12.5.1 [Looping Over Structure Elements], 頁 81 を参照してください。

forステートメントにおける代入式は，Octave の通常の代入ステートメントとは少し異なる動作をします。式の完全な結果を代入するかわりに，反復のたびに *var* に式の各列を代入します。もし *expression* が範囲，行ベクトル，あるいはスカラであれば，*var* の値は，ループ本体を実行するたびに，スカラとなるでしょう。もし *var* が列ベクトルあるいは行列ならば，*var* は，ループの本体を実行するたびに，列ベクトルになるでしょう。

以下の例では，フィボナッチ数列の 10 番めまでの要素を含むベクトルを生成するための別の方法を示しています。今回，forステートメントを用いています。

```
fib = ones (1, 10);
for i = 3:10
  fib (i) = fib (i-1) + fib (i-2);
endfor
```

このコードは，3 から 10 までの値を含む範囲を生成するため，最初に式 3:10 を評価することから動作します。その後，変数 iには，範囲の最初の要素が代入され，ループの本体を一度実行します。ループ本体の終端に達したとき，その範囲の次の値を iに代入し，ループ本体を再度実行します。この過程は，代入すべき要素が存在しなくなるまで継続します。

すべての forループは，whileループとして書き直すことは可能ですが，Octave 言語では両ステートメントをサポートしています。これは，for ループは入力が短くて済み，より自然な考え方ができるという理由によるものです。反復の回数をカウントすることは，ループにおいてとても当たり前のことであり，ループの一部としてカウントを考えることは，ループ内で何らかのカウントを行うよりは容易なことです。

12.5.1 構造体の要素をまたぐループ

forステートメントの特殊な使い方として，構造体の全ての要素にまたがるループができるようになるというものです。

```
for [ val, key ] = expression
  body
endfor
```

forステートメントのこの使い方において，*expression* の値は構造体でなければなりません。もしそうならば，これ以上要素が存在しなくなるまで，*key* と *val* には，要素名と，その反復において対応する値がセットされます。たとえば，以下のように使うことができます。

```
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
```

その要素は，何か特定の順序でアクセスしません。もし，特定の方法によってそのリストを通したループを行う必要があるならば，struct_elements 関数を使用するか，あなた自身でソートを行う必要があります。

key 変数は，省略することができます。もし省略すると，ブラケット [ ] も省略できます。これは，要素名を知る必要がないとき，全ての構造体要素の値を通して循環するときに役立ちます。

12.6 breakステートメント

breakステートメントは，それを囲む最も内側の forまたは while ループの外側にジャンプします。breakステートメントは，ループの本体内でのみ使用することができます。以下の例は，与えられた整数を除すことのできる最小の値を発見し，それが素数かどうかを識別します。

```
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
```

最初の whileステートメントにおいて剰余がゼロのとき，Octave は直ちにループを脱出します。これは，Octaveが直ちにそのループに続くステートメントに進み，処理を続けることを意味します（これは，exitステートメントとは大きく異なります。exitは，Octave プログラム全体を停止します)。

以前のものと等価な，別のプログラムを示します。これは，whileステートメントの *condition* が，if内部の breakに，どのように置き換わるのがよいかを示しています。

```
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
```

12.7 continueステートメント

continueステートメントは，breakのように，forまたは whileループの内部でのみ使用できます。これは，ループ本体の残り部分を飛び越え，直ちに次の循環を開始します。これと break（ループ全体を脱出する）とを対比します。以下に例を示します。

```
# 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
```

vec の要素のひとつが奇数ならば，この例は，その要素の値を表示するステートメントをスキップし，ループの先頭にあるステートメントに戻って継続します。

これは，continueステートメントの実用的な例ではありません。しかし，これがどのように動作するかを，はっきりと理解できるようにはなるはずです。通常は，以下のように書くことになるでしょう。

```
for x = vec
  if (rem (x, 2) == 0)
    printf ("%d\n", x);
  endif
endfor
```

12.8 unwind_protectステートメント

Octave は，Lisp の unwind-protect を真似た例外処理の一部方法についてサポートしています。

unwind_protectブロックの一般的な書き方は，以下のようなものです。

```
unwind_protect
  body
unwind_protect_cleanup
  cleanup
end_unwind_protect
```

ここで *body* と *cleanup* は，ともにオプションであり，任意の Octave 式またはコマンドを含めることができます。*cleanup* における式は，どのように制御が *body* を抜けるかに関わらず，実行されることが保証されます。

これは，グローバル変数を一時的に変更することによって起こりうるエラーを防ぐために役立ちます。たとえば，以下のコードは，たとえインデックス操作を実行している間にエラーが発生したとしても，組み込み変数 warn_fortran_indexing の元の値を復元することになります。

```
save_warn_fortran_indexing = warn_fortran_indexing;
unwind_protect
  warn_fortran_indexing = 1;
  elt = a (idx)
unwind_protect_cleanup
  warn_fortran_indexing = save_warn_fortran_indexing;
end_unwind_protect
```

unwind_protectが無いならば，インデックス操作を実行している間にエラーが発生したときに，組み込み変数 warn_fortran_indexingが元の値を復元することはできません。なぜならば，評価はエラーの時点で停止してしまい，値を復元するためのステートメントは実行されないからです。

12.9 tryステートメント

unwind_protect に加え，Octave では，例外処理の別の（限定された）書き方をサポートしています。

tryブロックの一般的な書き方は，以下のようなものです。

```
try
  body
catch
  cleanup
end_try_catch
```

ここで *body* と *cleanup* は，ともにオプションであり，任意の Octave 式またはコマンドを含めることができます。*cleanup* におけるステートメントは，*body* においてエラーが発生したときにのみ実行されます。

body が実行されているあいだ，警告やエラーメッセージは何も表示されません。*body* の実行中にエラーが発生すると，*cleanup* は，表示されるはずだったメッセージのテキストにアクセスするための関数 lasterrを使用することができます。これは，eval (*try*, *catch*)とすることに同じですが，上の関数がより効率的です。なぜならば，そのコマンドは，*try* および *catch* ステートメントを毎回解釈する必要がないからです。lasterr関数について，より多くの情報は Chapter 14 [Error Handling], 頁 101 を参照してください。

Octave の *try* ブロックは，Lisp の condition-case 形式の非常に限られたバリエーションです（異なるエラーのクラスを別個に処理することができないためです)。おそらく，同じ点において，Octave にはエラーのクラス分けについて，いくつかの種類があり，try-catch は，Lisp の condition-case と同じようにパワフルになり得るでしょう。

12.10 継続行

Octave 言語において，大部分のステートメントは改行文字で終了し，ある行から次の行までステートメントを継続するためには，改行文字を無視するように Octave に伝えねばなりません。行末に...あるいは\という文字がある行は，Octave のパーサによってトークンに分割される前に，次の行を連結します。たとえば，以下のような行について

```
x = long_variable_name ...
    + longer_variable_name \
    - 42
```

これは，1 行を形成します。2 行めのバックスラッシュ文字は，割り算の記号ではなく，継続文字として解釈されます。

文字列定数の内部で発生しない継続行については，継続記号および改行文字の間に，空白文字とコメントが出現してもかまいません。たとえば，

```
x = long_variable_name ...      # コメント 1
    + longer_variable_name \    # コメント 2
    - 42                        # 最後のコメント
```

このステートメントは，上で示した式と等価です。文字定数の内部では，改行の直前の行末に継続記号が出現しなければなりません。

かっこの内部で発生する入力は，継続行を使用する必要なしに，次の行へと続きます。たとえば，面倒な継続記号を付ける必要なく，以下のようなステートメントを書くことは可能です。

```
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
```

13 関数とスクリプトファイル

込み入った Octave プログラムは，関数を定義することにより，しばしば単純化することができます。関数は，対話的 Octave セッションの間にコマンドラインから，あるいは外部ファイル内で直接定義することができ，組み込み関数のように呼び出すことができます。

13.1 関数の定義

その最も単純な形式において，*name* という名前の関数の定義は，以下のように見えます。

```
function name
  body
endfunction
```

妥当な関数名は，妥当な変数名のようなものです。すなわち，連続する文字，数字およびアンダースコアであって，先頭が数字ではないものです。関数は，名前のプールを変数と共有します。

関数の本体 *body* は，Octave ステートメントから構成されます。これは，定義の最も重要な部分です。なぜならば，この部分は，関数が実際に何を実行するべきかを述べているからです。

たとえば，実行時に，端末のベルを鳴らす関数を示します（それが実行可能であると仮定します）。

```
function beep
  printf ("\a");
endfunction
```

`printf`ステートメント（see Chapter 16 [Input and Output], 頁 105 を参照）は，単に Octave に文字列`"\a"`を表示するように伝えます。特殊文字`"\a"` は，警告文字（アスキーコード 7）を表します。See Chapter 5 [Strings], 頁 35. を参照してください。

一度この関数を定義すると，関数名を打ち込むことにより，Octave にそれを評価するように伝えることができます。

普通なら，自分が定義した関数に，何らかの情報を渡したいと思うでしょう。Octave では，ある関数にパラメータを渡すための文法は，

```
function name (arg-list)
  body
endfunction
```

となります。ここで *arg-list* は，関数の引数をカンマで区切ったリストです。関数を呼び出すとき，引数の名前は，その呼び出しにおいて与えた引数値を保持するために使用されます。引数のリストは空でもかまいません。この場合，この形式は，最初に示した形式と同じものになります。

ベルを鳴らすとともにメッセージを表示するには，`beep`関数を，以下に示すように変更すればよいのです。

```
function wakeup (message)
  printf ("\a%s\n", message);
endfunction
```

以下のようなステートメントを用いて，この関数を呼び出すと，

```
wakeup ("Rise and shine!");
```

Octave は，端末のベルを鳴らし，‘`Rise and shine!`’というメッセージを表示して改行（`printf`ステートメントの最初の引数にある‘`\n`’）します。

大部分の場面において，自分で定義した関数から，何らかの情報を得たいこともあるはずです。ある 1 つの値を返す関数を書くための文法は，以下のようなものです。

```
function ret-var = name (arg-list)
  body
endfunction
```

ret-var は，関数によって返される値を保持することになる変数名です。この変数は，関数が値を返すようにするために，関数本体の終わりまでに定義されていなければなりません。

関数の本体において使用される変数は，その関数に対してローカルなものです。*arg-list* および *ret-var* に名前を挙げた変数も，その関数内のローカルなものです。関数内からグローバル変数にアクセスするための方法について，より多くの情報は See Section 9.1 [Global Variables], 頁 51. を参照してください。

たとえば，あるベクトルの要素の平均値を計算する関数を示します。

```
function retval = avg (v)
  retval = sum (v) / length (v);
endfunction
```

もし，代わりに以下のような avgを書いたとして，

```
function retval = avg (v)
  if (isvector (v))
    retval = sum (v) / length (v);
  endif
endfunction
```

さらに，引数としてベクトルの代わりに行列を用いて関数を呼び出すならば，Octave は以下のようなエラーメッセージを表示することになるでしょう。

```
error: ‘retval’ undefined near line 1 column 10
error: evaluating index expression near line 7, column 1
```

なぜならば，ifステートメントの本体は決して実行されず，retval は決して定義されないからです。このような目立たないエラーを避けるために，戻り値が常に値をもつようにいつでも確認をしたり，問題に遭遇したときに，有用なメッセージを出すことは良い心がけです。たとえば，avg関数は，以下のように書いてあれば良かったのです。

```
function retval = avg (v)
  retval = 0;
  if (isvector (v))
    retval = sum (v) / length (v);
  else
    error ("avg: expecting vector argument");
  endif
endfunction
```

この関数には，まだもう一つの問題が存在します。もしこの関数が引数を付けずに起動されたら？ということです。エラーチェックを追加しなくとも，おそらく Octave は，実際にはエラーの原因を突き止める手助けにはならないような，エラーメッセージを表示するでしょう。このようなエラーを捕獲ことができるようにするために，Octave には，各々の関数について nargin なる自動設定変数を提供しています。関数が呼び出されるたびに，narginは自動的に，実際に関数に渡された引数の数に初期化されます。たとえば，avg を以下のように書き換えたとしましょう。

```
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
```

期待したよりも多くの引数を付けてこの関数を呼び出すとき，Octave は自動的にエラーを報告しませんが，おそらく何かがおかしいことを示します。引数の数が少なすぎるときも，Octave はエラーを自動的に報告しませんが，値を与えられていない変数を使用しようという試みは，エラーになるでしょう。このような問題を避け，役立つメッセージを提供するためには，両方の可能性をチェックし，独自のエラーメッセージを出すようにします。

`nargin ()` [Built-in Function]
`nargin (`*fcn_name*`)` [Built-in Function]

ある関数の内部で，その関数に渡された引数の数を返す。トップレベルでは，Octave に渡されたコマンドライン引数の数を返す。もしオプション引数 *fcn_name* を付けて呼び出すならば，その関数が受け入れることのできる引数の最大数を返す。もしその関数が不定数の引数を受け入れるならば，-1 を返す。

`silent_functions` [Built-in Variable]

もし `silent_functions`の値がゼロでなければ，関数からの内部出力が抑制される。そうでなければ，セミコロンで終わっていない関数本体内の式の結果は，その値が出力される。標準状態は 0 である。

たとえば，もし以下の関数

```
function f ()
  2 + 2
endfunction
```

を実行するならば，Octave は `silent_functions`の値に依存して，‘`ans = 4`’と表示したり，何も表示しなかったりする。

`warn_missing_semicolon` [Built-in Variable]

もしこの変数の値がゼロでないならば，関数定義内のステートメントがセミコロンで終わっていないときに警告を表示する。標準状態は 0 である。

13.2 複数の戻り値

多くの他のコンピュータ言語とは異なり，Octave では，1 つ以上の値を返す関数が定義できるようになっています。複数の値を返す関数を定義するための文法は，以下のようなものです。

```
function [ret-list] = name (arg-list)
  body
endfunction
```

ここで *name*，*arg-list* および *body* は，以前のものと同じ意味であり，*ret-list* は，関数から戻った値を保持することになる変数名をカンマで区切ったリストである。戻り値のリストは，少なくとも 1 つの要素を持っていなければなりません。もし *ret-list* が 1 つの要素しか含まないならば，この `function` ステートメントの形式は，前の節で解説した形式と等価です。

あるベクトルの最大の要素と，その値がそのベクトルに最初に出現するインデックスの，2 つの値を返す関数の例を示します。

```
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
```

この具体例において，2 つの値は単一の配列の要素として返されることになります。しかし，これが常に可能であったり便利だったりするわけではありません。返される値は，整合性のある次元でないこともあります。また，個々の返り値に別個の名前を与えることは，しばしば好ましいものです。

関数が呼ばれるたびに `nargin`をセットすることに加え，Octave は，`nargout`を，返されると期待される値の数に初期化します。これは，関数のユーザがリクエストした値の数に依存して，異なる挙動をするような関数を書くことができるようにします。組み込み変数 `ans`への暗黙的な代入は，出力引数のカウントにおいて，判断されません。ゆえに，`nargout`の値はゼロになります。

`svd`と `lu`関数は，`nargout`の値によって異なる挙動をする組み込み関数の例です。

ある戻り値のみをセットする関数を書くことが可能です。たとえば，以下ある関数

```
function [x, y, z] = f ()
  x = 1;
  z = 2;
endfunction
```

について，以下のように呼び出すとします。

```
[a, b, c] = f ()
```

その結果は，

```
a = 1

b = [](0x0)

c = 2
```

であり，組み込み変数 `warn_undefined_return_values`がゼロでないならば，警告が発生します。

`nargout ()` [Built-in Function]
`nargout (`*`fcn_name`*`)` [Built-in Function]

ある関数の内部で，呼び出し側が受け取ことを期待する値の数を返す。もしオプション引数 *fcn_name* を付けて呼び出すならば，その関数が返すことのできる戻り値の最大数を返す。もしその関数が不定数の戻り値を返すならば，-1 を返す。

たとえば，

```
f ()
```

は，関数 fの内部では nargoutが 0 になり，また，

```
[s, t] = f ()
```

この関数 fの内部では，nargoutが 2 となる。

トップレベルでは，nargoutは定義されない。

warn_undefined_return_values [Built-in Variable]

もし warn_undefined_return_valuesがゼロでないならば，ある関数が期待される戻り値のリストにおいて 1 つでも値が定義されていないときに，警告を表示する。

nargchk (*nargin_min*, *nargin_max*, *n*) [Function File]

もし *n* が *nargin_min* から *nargin_max* までの範囲にあるならば，空行列を返す。そうでなければ，*n* が大きすぎるか小さすぎるかどうかを示すメッセージを返す。

この関数は，関数に与えた引数の数が，受け入れられる範囲内にあることを確かめるために便利である。

13.3 可変長の引数リスト

13.4 可変長の戻り値リスト

13.5 関数からのリターン

ユーザ定義関数の本体は，returnステートメントを含むことができます。このステートメントは，制御を Octave プログラムの残り部分に戻します。それは，以下のような使い方をします。

```
return
```

C 言語における returnとは異なり，Octave の returnステートメントは，ある関数から値を返すために使用することはできません。かわりに，functionステートメントの一部である戻り値変数のリストに，値を割り当てなければなりません。returnステートメントは，深く入れ子になったループあるいは条件つきステートメントから，その関数を終了することを単に容易にするものです。

あるベクトルのいずれかの要素がゼロでないかどうかを確認することを，チェックする関数の例です。

```
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
```

この関数は，そのベクトルがゼロでない要素を含んでいるときに，メッセージの表示を回避するための余計なロジックを加えること無しに，一度ゼロでない値を発見した場合に，ループを終了するための break ステートメントを使用して書き直すことができないのです。

`return` [Keyword]
Octave が，ある関数内あるいはスクリプト内で `return`キーワードに遭遇するとき，直ちに制御を呼び出し元に戻します。トップレベルでは，`return` ステートメントは無視されます。`return`ステートメントは，どの関数定義にも末尾に仮定されます。

13.6 関数ファイル

単純な 1 回完結のプログラムを除き，必要なときに毎回，必要とするすべての関数を定義しなければならないことは実用的ではありません。かわりに，通常はそれらをファイルに保存しておきたいと思うでしょう。そうすれば，簡単に編集することができ，後で使うために保存しておくことができます。

Octave は，使用前にファイルから関数定義を読み込む必要はありません。Octave が発見できる場所にファイルを置いておき，単に関数定義名を入力する必要があるだけです。

Octave が未定義の識別子に遭遇すると，すでにコンパイルされて現時点でシンボルテーブルに掲載されている変数または関数を最初に探します。もし，ここでその定義を見つけられなかったならば，未定義の識別子と同じベース名をもち，‘`.m`’で終わるファイルを，組み込み変数 `LOADPATH`によって指定されたディレクトリのリストから検索します。[1] 一度 Octave がマッチするファイルを見つけると，ファイルの中身を読み込みます。もし，そのファイルに *1* 個の関数が定義されているならば，その関数がコンパイルされて実行されます。ある単一のファイルに複数の関数を定義するにはどうするのか，についてさらなる情報は Section 13.7 [Script Files], 頁 94 を参照してください。

Octave が関数ファイルから関数を定義するときには，それを読み込んだファイルの完全な名前とタイムスタンプをを保存します。その後，その関数が必要になったときに毎回，そのファイルのタイムスタンプをチェックします。もしタイムスタンプが，最後に読み込んだ時間以降に変更されたことを示すならば，Octave は再度そのファイルを読み込みます。

タイムスタンプをチェックすることにより，Octave の実行中に，関数の定義を編集することができるようになり，Octave セッションを再起動することなく，新たな関数定義を自動的に使用できます。ある関数を使用するたびにタイムスタンプをチェックすることは非効率ですが，正しい関数定義を使用できるようにするために必要なのです。

変更されていないと思われる関数のタイムスタンプをチェックすることによるパフォーマンスの悪化を避けるために，Octave は，ディレクトリ‘*octave-home*`/share/octave/`*version*`/m`’ に存在する関数ファイルは変更されないと仮定します。ですから，それらのファイルで定義されている関数を使用するたびにチェックは行われません。これは，普通は非常によい仮定であり，Octave とともに配布される関数ファイルについてパフォーマンスを有意に向上させます。

もし Octave を実行しているあいだに，自作の関数ファイルが変更されないことが分かっているならば，変数 `ignore_function_time_stamp`に `"all"` とセットすることにより，パフォーマンスを向上させることができます。この変数に`"system"`をセットすると，標準の挙動になります。もしこれ以外の何かの値をセットするならば，Octave は全ての関数ファイルについてタイムスタンプをチェックするようになります。

`DEFAULT_LOADPATH` [Built-in Variable]
関数ファイルを検索するディレクトリを，コロンで区切って並べたもの。この変数の値は，組み込み変数 `LOADPATH`に現れた先頭，末尾あるいは二重のコロンへと自動的に代入される。

[1] ‘`.m`’という拡張子は，MATLAB との互換性のために選ばれたものです。

LOADPATH [Built-in Variable]

関数ファイルを検索するディレクトリを，コロンで区切って並べたもの。詳細は Chapter 13 [Functions and Scripts], 頁 87 を参照せよ。LOADPATHの値は，環境変数 OCTAVE_PATHを上書きする。これについては付記 C [Installation], 頁 313 を参照のこと。

LOADPATHは，TEX が TEXINPUTSを処理するのと同じ要領で処理される。LOADPATHの先頭，末尾あるいは二重に出現するコロンは，DEFAULT_LOADPATHの値で置き換えられる。LOADPATHの初期値は":"である。これは，DEFAULT_LOADPATHによって指定されたディレクトリを検索するよう Octave に伝えるものである。

さらに，いずれかのパス項目が '//'で終わっているならば，そのディレクトリと，それに含まれる全てのサブディレクトリに対して関数ファイルを再帰的に検索する。これにより，Octave が関数を最初に検索するときに，LOADPATH内で発見したファイルのリストをキャッシュしておくため，わずかに遅くなる。その後，Octave はファイルの内部キャッシュを検索するだけなので，通常はより高速に検索する。

再帰的ディレクトリ検索のパフォーマンスを向上させるために，再帰的に検索される各ディレクトリについて，追加的なサブディレクトリあるいは関数ファイルのどちらか片方を含め，両者を混ぜないのが最善である。

Octave とともに配布される関数ファイルディレクトリの説明は，Section 13.10 [Organization of Functions], 頁 99 を参照せよ。

rehash () [Built-in Function]

Octave のディレクトリキャッシュを再初期化する。

file_in_loadpath (*file*) [Built-in Function]
file_in_loadpath (*file*, "*all*") [Built-in Function]

LOADPATHによって指定されたディレクトリのリストの中に，*file* が見つかるならば，*file* の絶対名を返す。もしファイルが何も見つからないならば，空行列を返す。

もし最初の引数が文字列のセル配列ならば，セル配列の要素について LOADPATHの各ディレクトリを検索し，マッチした最初のファイル名を返す。

もし 2 番めのオプション引数"all"を与えるならば，そのパスにおいて同じ名前を持つ全てのファイル名のリストを含むセル配列を返す。もし何もファイルが見つからなければ，空のセル配列を返す。

ignore_function_time_stamp [Built-in Variable]

この変数は，関数ファイル内で定義された関数を検索するたびに，Octaveがシステムコール stat を実行しないようにするために使用する。もし ignore_function_time_stampが"system" であれば，Octave は '*octave-home*/lib/*version*'のサブディレクトリに存在する関数ファイルを自動的に再コンパイルしないようになる。たとえ，それが最後にコンパイルされたときから変更されていたとしても，である。しかし，LOADPATHに存在するその他の関数は，変更されていれば再コンパイルされる。この変数に"all"をセットすると，関数定義を clearで削除しない限り，Octave はどの関数ファイルも再コンパイルしないようになる。ignore_function_time_stampにそれ以外の値をセットすると，Octave は，関数ファイルで定義された関数を再コンパイルするかどうかを，常にチェックするようになる。ignore_function_time_stamp の初期値は"system" である。

`warn_function_name_clash` [Built-in Variable]

もし`warn_function_name_clash`の値がゼロでないならば，ある関数ファイル内で定義されている関数が，そのファイル名と異なることを発見したときに，警告を発生する（もし名前が合わなければ，ファイル内で宣言した関数名は無視される）。もしこの値が 0 ならば，警告を省略する。初期値は 1 である。

`warn_future_time_stamp` [Built-in Variable]

もしこの変数の値がゼロでないならば，未来のタイムスタンプを持つ関数ファイルを発見したときに警告を表示する。

13.7 スクリプトファイル

スクリプトファイルとは，（大部分が）一連の Octave コマンドを含むファイルのことです。これは，各々のコマンドを Octave プロンプトで打ち込んだかのように読み込まれ，評価されます。また，スクリプトファイルは，関数には論理的になじまないコマンド群を実行するための便利な方法を提供します。

関数ファイルとは異なり，スクリプトファイルは，`function`キーワードで始まりません。もしそうなっていれば，Octave はこれが関数ファイルであって，定義されてすぐに評価されるべき 1 個の関数を定義してあると仮定するでしょう。

スクリプトファイルは，そのファイル内で名付けた変数がローカル変数ではなく，コマンドラインで見ることのできる変数と同じスコープであるという点においても関数ファイルとは異なっています。

あるスクリプトファイルが`function`キーワードで始まることはないとしても，1 つのスクリプトファイル内に複数の関数を定義すること，そして，それらのすべてを一度に読み出す（でも実行はしない）ことは可能です。これを行うには，ファイルの最初のトークン（コメントと空白を無視した部分）が，`function`以外の何かでなければなりません。もし評価すべきステートメントが他に無いならば，以下のように，何も効果のないステートメントを使うと良いでしょう。

```
# Octave に，これが関数ファイルであると見なさない
# ようにする:

1;

# 関数 one の定義:

function one ()
  ...
```

これを Octave に読み込ませ，これらの関数を内部形式にするためには，このファイルを Octave の`LOADPATH`に確実に置く必要があります。その後，このコマンドを含むファイルのベース名を，単純に入力すればよいのです（Octave は，スクリプトファイルを検索するために，関数ファイルの検索と同じルールを使用します）。

もしファイル内の最初のトークン（コメントを除く）が`function`ならば，Octave は，空白ではない文字が関数定義の後に出現することについての警告を表示しつつ，その関数をコンパイルしてそれを実行しようとするでしょう。

評価する必要があるまで，Octave は任意の識別子の定義を調べようとしないことに注意してください。このことは，もし以下のステートメントがスクリプトファイルに現れるか，コマンドラインで入力されるならば，Octave はそれをコンパイルすることを意味します。

```
# 関数ファイルではありません:
1;
function foo ()
  do_something ();
endfunction
function do_something ()
  do_something_else ();
endfunction
```

たとえ，関数 fooで参照する以前に do_somethingが定義されていないとしてもです。この例はエラーになりません。なぜならば，Octave はその関数が実際に実行されるまで，ある関数によって参照される全てのシンボルを解決する必要がないためです。

Octave は，必要があるまで定義を探さないので，以下のコードは，コマンドラインで直接入力されたか，スクリプトファイルから読み込まれたか，あるいは関数の本体であるかにかかわらず，常に‘bar = 3’と表示するでしょう。たとえ，Octave の LOADPATHの ‘bar.m’を呼び出す関数またはスクリプトファイルが存在したとしてもです。

```
eval ("bar = 3");
bar
```

関数本体内で現れるこのようなコードは，もし定義が，その関数がコンパイルされているとして解決されたならば，Octave を惑わします。このコードを矛盾のない方法で評価できるくらいまで Octave を賢くすることは，不可能と言ってもいいでしょう。パーサが，コンパイル時に evalの呼び出しを実行することができ，そして，評価されるべき文字列における全ての参照も解決されない限りは不可能であって，さらに，それは限定的すぎること（その文字列はユーザ入力に由来するかもしれないし，ある関数が評価されるまでは知り得ないことに依存するかもしれない）を要求するからです。

普通なら，Octave は‘*file*.m’なる名前をもつスクリプトファイルからコマンドを実行しますが，任意のファイルからコマンドを実行するには，source関数を使用することになります。

source (*file*) [Built-in Function]
file の内容をパースし，実行する。これはスクリプトファイルからコマンドを実行することと等価であるが，ファイルが ‘*file*.m’という名前である必要がない。

13.8 動的にリンクされる関数

あるシステムでは，Octave は，C++言語で書かれた関数を動的にロードして実行することができます。Octave は C++で書かれた関数を直接呼び出すことだけしかできません。しかし，他の言語で書いた関数も，C++で書いた単純なラッパ関数から呼び出すことにより，ロードすることができます。

Octave がロードすることのできる C++関数を書くための方法の一例を，コメント付きで示します。この関数のソースは，Octave のソース配布において，‘examples/oregonator.cc’として含めています。この例は，Section 23.1 [Ordinary Differential Equations], 頁 177 の例題において使用しているのと同じ微分方程式を定義しています。その例とこのコードを実行することにより，動的リンク関数を使用することによって期待される速度向上の性質を確認するために実行時間を比較できます。

‘oregonator.cc’において定義された関数は，8 つのステートメントのみを含み，対応する M-ファイル（これも Octave とともに，‘examples/oregonator.m’ として配布されています）において定義されたコードと，それほど大きな違いはありません。

‘oregonator.cc’の完全なテキストです:

```
#include <octave/oct.h>

DEFUN_DLD (oregonator, args, ,
  "The ‘oregonator’.")
{
  ColumnVector dx (3);

  ColumnVector x (args(0).vector_value ());

  dx(0) = 77.27 * (x(1) - x(0)*x(1) + x(0)
                   - 8.375e-06*pow (x(0), 2));

  dx(1) = (x(2) - x(0)*x(1) - x(1)) / 77.27;

  dx(2) = 0.161*(x(0) - x(2));

  return octave_value (dx);
}
```

このファイルの最初の行,

```
#include <octave/oct.h>
```

これは，必要になるであろう，Octave の内部関数の全てに対する定義をインクルードしています。もし標準 C++あるいは C ライブラリからの他の関数が必要ならばmここで必要なヘッダファイルをインクルードできます。

次の 2 行,

```
DEFUN_DLD (oregonator, args, ,
  "The ‘oregonator’.")
```

これは，関数を宣言しています。マクロ DEFUN_DLD，およびこのマクロが依存するマクロ群はファイル‘defun-dld.h’, ‘defun.h’および‘defun-int.h’（これらのファイルは，‘octave/oct.h’でインクルードされています）で定義されています。

DEFUN_DLDに対する 3 番めの引数（nargout）は使用されませんので，未使用の関数パラメータに関する gcc からの警告を避けるためには，引数リストから省略します。

次の行,

```
ColumnVector dx (3);
```

これは，単に微分方程式の右辺を格納するためのオブジェクトを宣言しています。また，ステートメント

```
ColumnVector x (args(0).vector_value ());
```

これは，1 番めの入力引数からベクトルを展開します。vector_value メソッドが使用された結果，関数の利用者は行ベクトルまたは列ベクトルのどちらも渡すことができます。ColumnVectorコンストラクタは必要です。なぜならば，ODE クラスは，列ベクトルを必要とするからです。変数 argsは，DEFUN_DLDで octave_value_listオブジェクトとして定義された関数に渡されます。これは，リストの長さを得たり，個々の要素を展開するためのメソッドを含みます。

この例において，エラーをチェックしていませんが，それは難しいことではありません。Octave の組み込み関数の全ては，その引数についてある形式のチェックを行っています。ですから，それら

関数のソースコードをチェックすれば，与えられた引数の数と型を照合する様々な方法の例が得られます。

次のステートメント，

```
dx(0) = 77.27 * (x(1) - x(0)*x(1) + x(0)
                 - 8.375e-06*pow (x(0), 2));

dx(1) = (x(2) - x(0)*x(1) - x(1)) / 77.27;

dx(2) = 0.161*(x(0) - x(2));
```

これは方程式の右辺を定義しています。最後に，dxを返します。

```
return octave_value (dx);
```

実際に戻す型は octave_value_listですが，これは戻す型を octave_valueに変換するためだけに必要です。なぜならば，octave_valueオブジェクトから，その型のオブジェクトを自動的に生成するデフォルトのコンストラクタが存在するからです。ですから，かわりにそれを使うだけなのです。

このファイルを使用するためには，お使いの Octave が動的リンクをサポートしていなければなりません。サポートの有無を確かめるには，Octave プロンプトで *octave_config_info ("dld")*というコマンドを入力してください。もしこれが 1 を返したならば，動的リンクのサポートが含まれています。

このサンプルファイルをコンパイルするためには，シェルのプロンプトで ‘mkoctfile oregonator.cc’と入力してください。mkoctfileというコマンドは，Octave とともにインストールされているはずです。これを実行することにより，Octave によってロードすることができる ‘oregonator.oct’ なるファイルが生成されるでしょう。ファイル ‘oregonator.oct’をテストするには，Octave を起動して，プロンプトから以下のコマンドを打ち込みます。

```
oregonator ([1, 2, 3], 0)
```

Octave は以下の表示をすることによって反応を返すはずです。

```
ans =

   77.269353
   -0.012942
   -0.322000
```

この時点で，この微分方程式を解くために，oregonator.mファイルと同じように ‘oregonator.oct’が使用できるようになりました。

Linux を実行している 133MHz の Pentium マシンでは，Octave は Section 23.1 [Ordinary Differential Equations], 頁 177 で示した問題を，M-ファイルを使用した約 19 秒と比較して，動的リンク関数を使用して約 1.4 秒で解きました。これと同様な実行時間の短縮は，他の関数にも期待できます。特に，ユーザ提供関数を必要とする lsodeのような関数には適しています。

M-ファイルを同じように，動的リンク関数が最後にロードされた時間よりも，より新しくそのファイルが定義されたときには，Octave は動的リンク関数を自動的に再ロードします。もし 1 個の ‘.oct’ ファイル内に複数の関数が定義されているならば，ファイルを再ロードすることによって，他の関数も強制的にクリアして再ロードします。もし与えられた ‘.oct’ファイルからロードした全ての関数をクリアするならば，Octave は自動的に ‘.oct’ファイルをアンロードします。

`warn_reload_forces_clear` [Built-in Variable]

もし同じファイルから複数の関数がロードされているならば，それらのどれか 1 つを再ロードする前に，全ての関数をクリアしなければならない。もし，warn_reload_forces_clearが

ゼロでないならば，これが発生するときに警告を発し，強制的にクリアされるその他の関数のリストを表示する。

`variables_can_hide_functions` [Built-in Variable]
もしこの変数の値がゼロでないならば，変数への代入は，以前に定義された同名の関数を隠すことになる。この変数が負の値ならば警告を表示するようになるが，操作は許される。

動的リンク関数を書くためのさらなる情報は，Octave 配布パッケージの ‘`src`’ ディレクトリにあるファイルで入手できます。現在では，そこには以下のようなファイルが含まれます。

```
balance.cc   fft2.cc      inv.cc       qzval.cc
chol.cc      filter.cc    log.cc       schur.cc
colloc.cc    find.cc      lsode.cc     sort.cc
dassl.cc     fsolve.cc    lu.cc        svd.cc
det.cc       givens.cc    minmax.cc    syl.cc
eig.cc       hess.cc      pinv.cc
expm.cc      ifft.cc      qr.cc
fft.cc       ifft2.cc     quad.cc
```

これらのファイルは，`DEFUN_DLD`の代わりに `DEFUN_DLD_BUILTIN` マクロを使用しています。これら 2 つのマクロの違いは，オペレーティングシステムが動的リンクをサポートしないならば，`DEFUN_DLD_BUILTIN`が，動的にロードされない組み込み関数を定義することになる，という点だけです。

現在のところ，組み込み関数において呼ぶことのできる全ての関数についての，詳細な解説はありません。それが実現するまでのあいだ，Octave のソースコードを読む必要があるでしょう。

13.9 関数ハンドルとインライン関数

ここでは，関数ハンドルとインライン関数の説明のためにとってある場所です。

13.9.1 関数ハンドル

`functions` (*`fcn_handle`*) [Built-in Function]
関数ハンドル *fcn_handle* に関する情報を含む構造体を返す。

`func2str` (*`fcn_handle`*) [Built-in Function]
関数ハンドル *fcn_handle* によって参照される関数名を含む文字列を返す。

`str2func` (*`fcn_name`*) [Built-in Function]
文字列 *fcn_name* から構成される関数ハンドルを返す。

13.9.2 インライン関数

`inline` (*`str`*) [Built-in Function]
`inline` (*`str`*, *`arg1`*, ...) [Built-in Function]
`inline` (*`str`*, *`n`*) [Built-in Function]
文字列 *str* からインライン関数を生成する。もし 1 つの引数を付けて呼ばれたならば，生成された関数の引数は，関数それ自身から展開される。生成された関数の引数は，そのとき，アルファベット順になるだろう。引数として *i* および *j* は無視されることに注意せよ。これは，それが変数として使われているのか，組み込み定数として使われているかが曖昧だからである。かっこに続く全ての引数は，関数になると見なされる。

もし 2 番めおよびそれ以降の引数が文字列ならば，それらは，その関数の引数の名前である。
もし 2 番めの引数が整数 *n* ならば，その引数は，"x", "P1", ..., "P*N*"である。

argnames (*fun*) [Built-in Function]
インライン関数 *fun* の引数の名前を含む文字列のセル配列を返す。

formula (*fun*) [Built-in Function]
インライン関数 *fun* を表す文字列を返す。char (*fun*)は，formula (*fun*)と等しいことに注意せよ。

argnames (*fun*) [Built-in Function]
登場する全ての*や/などを，.*, や./などに置き換えることにより，インライン関数 *fun* のベクトル化バージョンを生成する。

13.10 Octave とともに配布される関数の構成

Octave の標準関数の多くは，関数ファイルとして配布されています。それらは，'*octave-home*/lib/octave/*version*/m'のサブディレクトリ以下にあり，見つけやすくするためにトピックによって緩く構成されています。

以下に示すものは，関数ファイルサブディレクトリと，そこで見つけたいと思う関数の型の全てのリストです。

'audio' 音声を再生したり録音するための関数です。

'control' automatic control systems のシミュレーションをデザインするための関数です。

'elfun' 初等関数です。

'general' flipud, rot90, triuのような雑多な行列操作関数のほか，ismatrixや nargchkなどのような基本的な関数を含みます。

'image' 画像処理ツールです。これらの関数には X Windows System が必要です。

'io' 入出力関数です。

'linear-algebra'
線形代数のための関数です。

'miscellaneous'
このほかのどこにも属さない関数です。

'plot' MATLAB ライクなプロット関数を実装した関数群です。

'polynomial'
多項式を操作するための関数です。

'set' 一意な値の集合を生成したり操作するための関数です。

'signal' signal processing applications のための関数です。

'specfun' 特殊な関数です。

'special-matrix'
特殊な行列構造を作るための関数です。

‘`startup`’ Octave のシステム寄りのスタートアップファイルです。

‘`statistics`’
統計関数です。

‘`strings`’ 雑多な文字列処理関数です。

‘`time`’ 時間保持に関連する関数です。

14 エラー処理

Octave は，エラーや警告メッセージを表示するために，いくつかの関数を持っています。異常な状態に遭遇したときに特別な動作をする必要のある関数を書くとき，この章で説明する関数を用いてエラーメッセージを表示すべきです。

`error` (*template*, ...) [Built-in Function]

`error`関数は，`printf`に類する関数と同じルール（Section 16.2.4 [Formatted Output], 頁 115 を参照）を用いて，テンプレート文字列 *template* の制御下で，オプション引数の書式を整える。生み出されるメッセージの先頭には‘`error: `’が付けられ，`stderr`ストリームに表示される。

`error`を呼び出すことは，これ以上のコマンドを評価することなくトップレベルに制御が戻るような，Octave の内部エラー状態をセットすることにもなる。これは，関数またはスクリプトを停止するには便利である。

もしエラーメッセージが改行文字で終わっていなければ，Octave は，エラーに先だって呼び出された全ての関数のトレースバック（呼び出し履歴）を表示する。たとえば，以下の関数定義が与えられていたとする。

```
function f () g () end
function g () h () end
function h () nargin == 1 || error ("nargin != 1"); end
```

関数 `f`を呼び出すと，エラーの正確な位置をすぐに見つける手がかりとなるようなメッセージリストを出力する。

```
f ()
error: nargin != 1
error: evaluating index expression near line 1, column 30
error: evaluating binary operator ‘||’ near line 1, column 27
error: called from ‘h’
error: called from ‘g’
error: called from ‘f’
```

もしエラーメッセージが改行文字で終わっているならば，Octave はそのメッセージを表示するが，処理をトップレベルに戻すときにはトレースバックメッセージを何も表示しなくなる。たとえば，上の例のエラーメッセージの終端を改行に修正すると，Octave はひとつのメッセージしか表示しなくなる。

```
function h () nargin == 1 || error ("nargin != 1\n"); end
f ()
error: nargin != 1
```

`beep_on_error` [Built-in Variable]

もし `beep_on_error`がゼロでないならば，Octave はエラーメッセージを表示する前に端末のベルを鳴らそうとする。初期値は 0 である。

`warning` (*msg*) [Built-in Function]

文字列‘`warning: `’に続けて警告メッセージ *msg* を表示する。警告メッセージを表示した後，Octave はコマンドの実行を続ける。このコマンドは，ユーザに異常事態を知らせたいときに使用すべきであるが，独自のプログラムが実行を継続する意味があるときに限るべきである。

`usage (msg)` [Built-in Function]

文字列 'usage: ' に続けてメッセージ *msg* を表示し，これ以上のコマンドを評価することなくトップレベルに制御が戻るような，Octave の内部エラー状態をセットする。これは，関数を停止するときに便利である。

usageが評価された後，Octave は，使用法メッセージに先だって呼び出された全ての関数のトレースバック（呼び出し履歴）を表示する。

この関数は，引数の数が正しくなかったり，引数の型が間違った状態で関数を呼び出したときのように，関数の不適切な呼び出しから発生した問題を報告するために使用するべきである。たとえば，Octave とともに配布されている大部分の関数は，以下のようなコードで書き始めている。

```
if (nargin != 2)
  usage ("foo (a, b)");
endif
```

これは適切な引数の数をチェックするためである。

`lasterr ()` [Built-in Function]
`lasterr (msg)` [Built-in Function]

何も引数が無いときには，最後のエラーメッセージを返す。1 つの引数を付けると，最後のエラーメッセージを *msg* にセットする。

`lastwarn ()` [Built-in Function]
`lastwarn (msg)` [Built-in Function]

何も引数が無いときには，最後の警告メッセージを返す。1 つの引数を付けると，最後の警告メッセージを *msg* にセットする。

以下の一対の関数は，使いやすさに制限があり，Octave の将来のバージョンでは削除されるかもしれません。

`perror (name, num)` [Function File]

エラー番号 *num* に対応して，関数 *name* についてエラーメッセージを表示する。この関数は，エラーコードを数値で返すような関数に対するエラーメッセージを便利に表示するために使用することを目的としている。

`strerror (name, num)` [Function File]

関数 *name* について，エラー番号 *num* に対応するエラーメッセージのテキストを返す。この関数は，エラーコードを数値で返すような関数に対するエラーメッセージを便利に表示するために使用することを目的としている。

15 デバッグ

`rline = dbstop` (*func, line*) [Loadable Function]

ある関数内にブレークポイント（breakpoint）を設定する。

`func` 関数名を表す文字列である。すでにデバッグモードに入っているとき，これは省略し，行のみを与えるべきである。

`line` ブレークポイントをセットしたい行を指定する。

返される *rline* は，ブレークポイントが設定された実際の位置である。

`dbclear` (*func, line*) [Loadable Function]

関数内に設定したブレークポイントを削除する。

`func` 関数名を表す文字列である。すでにデバッグモードに入っているとき，これは省略し，行のみを与えるべきである。

`line` ブレークポイントを削除したい行を指定する。

指定した行が本当にブレークポイントかどうかを確かめるチェックは行っていない。もし間違った行を指定すると，何も起こらない。

`lst = dbstatus` ([*func*]) [Loadable Function]

ある関数がもつブレークポイントの行を含むベクトルを返す。

`func` 関数名を表す文字列である。すでにデバッグモードに入っているとき，これは省略すべきである。

`dbwhere` () [Loadable Function]

そのコードのどこにいるかを表示する。

`dbtype` () [Loadable Function]

行番号付きでスクリプトファイルを表示する。

`debug_on_interrupt` [Built-in Variable]

もし `debug_on_interrupt`がゼロでないならば，Octave が割り込み（インタラプト）信号（普通であれば *C-c*で発生する）を受け取ったときにデバッグモードに入ることになる。もしデバッグモードに入る前に 2 回めの割り込みを受け取ったならば，通常の割り込みが発生する。初期値は 0 である。

`debug_on_warning` [Built-in Variable]

もし `debug_on_warning`がゼロでないならば，Octave は警告に遭遇したときにデバッガに入ろうとする。初期値は 0 である。

`debug_on_error` [Built-in Variable]

もし `debug_on_error`がゼロでないならば，Octave はエラーに遭遇したときにデバッガに入ろうとする。これは，通常のトレースバックメッセージの表示を行わないようにもする（トップレベルのエラーメッセージだけが見えることになる）。初期値は 0 である。

16 入力と出力

入出力関数は，明確に 2 つに分類できます。1 群めは，MATLAB で利用できる関数を真似たものです。2 群めは，C 言語で使用されている標準 I/O ライブラリを真似たものであり，さらなる柔軟性と出力の制御を提供してくれます。

対話的に実行しているとき，Octave は通常，端末の 1 画面の長さに収まらない出力を，`less`や`more`といったページャ（1 画面単位で表示するプログラム）に送ります。これにより，多量の出力が，読む前に流れてしまうという問題を回避します。`less`（あるいは`more`の一部のバージョン）を使用すると，前方や後方に移動したり，特定の項目を検索したりできます。

通常，Octave がトップレベルプロンプトの表示を準備する直前，あるいは標準入力から読み込む（たとえば`fscanf`や`scanf`関数を使用する）直前まで，ページャによる出力の表示は行われません。これは，もしあなたが 1 個のコマンドステートメントで，かなりの量の作業を実施しようとするならば，何らかの出力がスクリーンに現れる前に，いくらかの遅れが存在することを意味します。関数`fflush`は，出力を強制的にページャ（あるいは任意の他のストリーム）へと直ちに送るために使えるでしょう。

変数`PAGER`に値をセットすることにより，ページャとして実行するためのプログラムを選ぶことができます。また，変数`page_screen_output`の値を 0 にセットすることにより，ページごとの表示をやめることができます。

`more` [Command]
`more` *on* [Command]
`more` *off* [Command]
出力のページ区切りをするかどうかを切り替える。引数を付けないときは，現在のオンとオフを切り替える。

`PAGER` [Built-in Variable]
初期値は，ふつうは`"less"`，`"more"`あるいは`"pg"` である。これは利用しているシステムにどんなプログラムがインストールされているかに依存する。付記 C [Installation], 頁 313 を参照せよ。

対話的に実行しているとき，Octave は，端末の 1 画面の長さに収まらない出力を，変数`PAGER`の値の名前を持つプログラムに送る。

`page_screen_output` [Built-in Variable]
もし`page_screen_output`がゼロでないならば，1 ページよりも長い全ての出力は，ページャに送られる。これは，一度に 1 画面ずつ眺められるようにするものである。ページャの中には（`less`のような—付記 C [Installation], 頁 313 を参照のこと），出力の後方へさかのぼる機能を持つものがある。初期値は 1 である。

`page_output_immediately` [Built-in Variable]
もし`page_output_immediately`の値がゼロでないならば，Octave は，出力が得られるとすぐにページャに送る。そうでなければ，Octave はその出力をバッファリングし，その出力をページャに流すため，プロンプトが表示される直前まで待機する。

`fflush` (*fid*) [Built-in Function]
出力を *fid* にフラッシュする。これは，全ての保留中の出力を，他のイベントが発生する前にスクリーンへと出そうとするときに便利である。たとえば，`input`を呼ぶ前に標準出力ストリームをフラッシュすることは，いつもよい心がけである。

fflushは成功時に 0 を，エラー時には OS に依存するエラー値（UNIX では −1）を返す。

16.1 基本的な入出力

16.1.1 端末への出力

Octave は通常，式が評価されるとすぐにその値を表示するので，すべての I/O 関数の最も単純なものは，単なる式であるといえます。たとえば，以下の式は piの値を表示するでしょう。

```
pi
     ⊣ pi = 3.1416
```

これは，その値とともに変数名（または‘ans’）が表示されるということを受け入れられるのであれば，問題なく動作しています。変数名を表示させずに，変数の値を表示するためには，関数 dispを使用してください。

formatコマンドは，Octave が，dispや通常の表示メカニズムを通して値を表示する方法をいくらか制御する機能を提供します。

ans [Built-in Variable]

この変数は，最も最近計算した結果で，明示的に変数へ代入していない値を保持する。たとえば，以下の式を実行した後，

```
3^2 + 4^2
```

ansの値は 25 である。

fdisp (*fid*, *x*) [Built-in Function]

x の値を，ストリーム *fid* に表示する。例を示す。

```
fdisp (stdout, "The value of pi is:"), fdisp (stdout, pi)

     ⊣ the value of pi is:
     ⊣ 3.1416
```

fdispによる出力は，常に改行で終わることに注意せよ。

disp (*x*) [Built-in Function]

x の値を画面に表示する。例を示す。

```
disp ("The value of pi is:"), disp (pi)

     ⊣ the value of pi is:
     ⊣ 3.1416
```

dispによる出力は，常に改行で終わることに注意せよ。

もし出力値がリクエストされるならば，dispは何も表示せず，フォーマット済み出力を文字列で返す。

format *options* [Command]

dispおよび Octave の通常の表示メカニズムによって生み出される出力のフォーマットを制御する。妥当なオプションは，以下の表に列挙してある。

`short` Octave は，最大 10 文字幅（行列の列間に挟むスペースはカウントしない）のフィールド内に，少なくとも有効桁が 5 桁で数字を表示しようとする。

もし Octave が行列の書式設定ができず，その結果，列が小数点で並んでおり，全ての数値が最大のフィールド幅内に収まっていれば，‘e’フォーマットに切り替える。

`long` Octave は，最大 20 文字幅（行列の列間に挟むスペースはカウントしない）のフィールド内に，少なくとも有効桁が 15 桁で数字を表示しようとする。

‘short’フォーマットと同様に，もし Octave が行列の書式設定ができず，その結果，列が小数点で並んでおり，全ての数値が最大のフィールド幅内に収まっていれば，‘e’フォーマットに切り替える。

`long e`
`short e` ‘format long’あるいは‘format short’と同様であるが，常に‘e’ フォーマットで出力結果を表示する。たとえば，‘short e’フォーマットについて，piは 3.14e+00と表示される。

`long E`
`short E` ‘format long e’あるいは‘format short e’と同様であるが，常に大文字の‘E’ フォーマットで出力結果を表示する。たとえば，‘long E’ フォーマットでは，pi は 3.14159265358979E+00と表示される。

`long g`
`short g` 数値の大きさに基づいて，通常の‘long’（あるいは‘short’）と‘long e’（あるいは‘short e’）フォーマットを選択する。たとえば，‘short g’フォーマットでは，`pi .^ [2; 4; 8; 16; 32]` は以下のように表示される。

```
ans =

     3.1416
     9.8696
     97.409
     9488.5
  9.0032e+07
  8.1058e+15
```

`long G`
`short G` ‘format long g’または‘format short g’と同じであるが，大文字の‘E’フォーマットを使用する。たとえば，‘short G’フォーマットでは，`pi .^ [2; 4; 8; 16; 32]`は以下のように表示される。

```
ans =

     3.1416
     9.8696
     97.409
     9488.5
  9.0032E+07
  8.1058E+15
```

`free`
`none` 行列の列を小数点でそろえようとはせずに，自由なフォーマットで出力する。これは複素数についても，‘`0.60419 + 0.60709i`’のような出力ではなく，‘`(0.604194, 0.607088)`’のようなフォーマットになる。

`bank` 小数点以下 2 桁までの固定フォーマットで表示する。

`+`
`+` *chars*
`plus`
`plus` *chars*
ゼロでない行列の要素を‘`+`’記号で，ゼロの要素をスペースで表示する。このフォーマットは，巨大な行列の構造を確かめるときに非常に有効である。

オプション引数 *chars* は，それぞれゼロより大きい，ゼロより小さい，ゼロに等しい値を表示するために使用する 3 文字のリストである。たとえば，‘`+ "+-."`’フォーマットでは，`[1, 0, -1; -1, 0, 1]`は以下のように画面に表示される。

```
ans =

+.-
-.+
```

`native-hex`
メモリに格納されている形式どおりに，16 進数を表示する。たとえば，最下位バイトが最初にくる IEEE の 8 バイト実数を利用するワークステーションにおいて，`pi`の値を `hex`フォーマットで表示すると，`400921fb54442d18`となる。このフォーマットは，数値にのみ有効である。

`hex` `native-hex`と同様であるが，常に最上位バイトを最初に表示する。

`native-bit`
メモリに格納されている形式どおりに，2 進数を表示する。たとえば，`pi`の値は，以下のようになる。

```
0100000000001001001000011111011
0101010001000100001011010011000
```

（ここでは折り返しが発生する関係上，32 ビット区切りで 2 段表示している）これは，最下位バイトが最初にくる IEEE の 8 バイト実数を利用するワークステーションにおいて，表示したときのものである。このフォーマットは，数値にのみ有効である。

`bit` `native-bin`と同様であるが，常に最上位バイトを最初に表示する。

`compact` 列数ラベルの周囲に余計な空白を入れない。

`loose` 列数ラベルの上下に空白行を挿入する（これは標準設定である）。

標準設定により，Octave は最大 10 文字幅のフィールド内に，有効桁が 5 桁で数字を表示しようとする。

もし Octave が行列の書式設定ができず，その結果，列が小数点で並んでおり，全ての数値が最大のフィールド幅内に収まっていれば，‘`e`’フォーマットに切り替える。

もし `format`が引数無しで起動されるならば，標準のフォーマット状態を復元する。

`print_answer_id_name` [Built-in Variable]
もし `print_answer_id_name`の値がゼロでなければ，その結果とともに変数名を表示する。そうでなければ，結果の値のみを表示する。初期値は 1 である。

16.1.2 端末からの入力

Octave には，ユーザ入力を促すことを容易にする 3 つの関数が存在します。`input`と `menu`関数は，通常，ユーザとの対話的設計を管理するために使用され，`keyboard`関数は，通常，簡単なデバッグを行うために使用されます。

`input (`*`prompt`*`)` [Built-in Function]
`input (`*`prompt`*`, "`*`s`*`")` [Built-in Function]
プロンプトを表示し，ユーザ入力を待つ。たとえば，

```
input ("Pick a number, any number! ")
```

は，以下のようなプロンプトを表示する。

```
Pick a number, any number!
```

そして，ユーザが値を入力するのを待つ。ユーザによって入力された文字列は式として評価される。ゆえに，それがリテラル定数，変数名，あるいは他の妥当な式となり得る。

現在，`input`は，式を評価することによって生み出された値の数にかかわらず，単に 1 つの値のみを返す。

もし文字列値を得ることだけに興味があるならば，2 番めの引数として文字列`"s"`を付けて `input`を呼び出すと良い。これは，ユーザによって入力された文字列を，評価せずに返すように Octave に命令する。

ページャによって表示される出力が待機されることがあるので，`input` を呼ぶ前に，常に`fflush (stdout)`を呼ぶことは良い心がけである。これは，保留中の出力が，あなたの出すプロンプトより前にスクリーンに表示されるようにするものである。

`menu (`*`title`*`, `*`opt1`*`, ...)` [Function File]
タイトル文字列に続けて一連のオプションを表示する。各々のオプションは，数値とともに表示する。戻り値は，ユーザが選択したオプションの数値である。この関数は，対話的プログラムに役立つ。渡すことのできるオプションの数には制限はないが，簡単には 1 画面に収まらないメニューを出すことは混乱の元である。

`keyboard (`*`prompt`*`)` [Built-in Function]
この関数は，通常は単なるデバッグのために使用される。`keyboard`関数を実行するとき，Octave はプロンプトを表示し，ユーザ入力を待機する。その後，入力文字列が評価され，その結果が表示される。これにより，関数内で変数の値を試験すること，および変数に新たな値を代入することが可能になる。`keyboard`は，何に値も返さない。また，ユーザが‘`quit`’または‘`exit`’と打ち込むまで入力の待ち受けを継続する。

`keyboard`が何も引数を付けずに起動されると，標準のプロンプト‘`debug> `’が使用される。

`input`と `keyboard`の両方とも，プロンプトにおいて通常のコマンドライン履歴および編集関数が利用できます。

Octave には，ユーザに改行を入力してもらう必要なく，キーボードから 1 文字を得ることが可能な関数も用意されています。

`kbhit ()` [Built-in Function]

キーボードから，単一のキー投下を読み込む。1 つの引数を付けて呼び出すと，キー入力を待機しない。たとえば，

```
x = kbhit ();
```

x には，入力が行われるとすぐに，キーボードから入力した次の文字がセットされる。

```
x = kbhit (1);
```

これは上の例と同じであるが，キー投下を待たず，何もキーが得られなかったならば空の文字列を返す。

16.1.3 単純なファイル入出力

`save`と `load`コマンドは，様々なフォーマットでデータをファイルに書き出したり，ファイルから読み込んだりするものです。`save` コマンドによって書き出されるファイルの標準フォーマットは，組み込み変数 `default_save_format`と `save_precision`を使用することでコントロールできます。

Octave は，未だに構造体変数あるいはユーザ定義型を保存（save）したり読み出（load）したりすることはできないことに注意してください。

`save` *options file v1 v2 . . .* [Command]

あるファイル *file* に，変数 *v1*, *v2*, . . . を保存する。特殊なファイル名 '`-`' は，端末に出力内容を書き出す。もし変数名を列記しないならば，Octave は現在のスコープにある全ての変数を書き出す。`save`コマンドで利用できるオプションは，以下の表に列挙してある。出力フォーマットを修正するオプションは，組み込み変数 `default_save_format`によって指定されたフォーマットの設定を上書きする。

`-ascii` Octave のテキストデータフォーマットで，データを保存する。

警告: このオプションの意味は，MATLAB との互換性のため，Octave の将来のバージョンで変更されることがある。この変更があっても，現在のコードの意味を保つためには，かわりに`-text`オプションを使用せよ。

`-text` Octave のテキストデータフォーマットで，データを保存する。

`-binary` Octave のバイナリデータフォーマットで，データを保存する。

`-float-binary`
Octave のバイナリデータフォーマットで，単精度のみを利用してデータを保存する。保存される全ての値が単精度の表現になることを分かっているときにのみ，このオプションを使用するべきである。

`-mat`
`-mat-binary`
MATLAB のバイナリデータフォーマットで，データを保存する。

`-V4`
`-v4`
`-4`
`-mat4-binary`
MATLAB version 4 のバイナリデータフォーマットで，データを保存する。

`-hdf5` HDF5 フォーマットでデータを保存する。(HDF5 は，University of Illinois の National Center for Supercomputing Applications によって開発された，フリーで軽量なバイナリフォーマットである)

Octave の実行形式が HDF5 ライブラリをリンクしていないときには，HDF による保存や読み込みはできない。

`-float-hdf5`
HDF5 フォーマットで，単精度のみを利用してデータを保存する。保存される全ての値が単精度の表現になることを分かっているときにのみ，このオプションを使用するべきである。

`-save-builtins`
組み込み変数も保存するようにする。初期設定では，Octave は組み込み変数を保存しない。

保存すべき変数のリストは，以下の特殊文字から構成されるワイルドカードパターンを含んでも良い。

`?` 任意の 1 文字にマッチする。

`*` 0 文字以上の文字列にマッチする。

`[ list ]` *list* によって指定された文字のいずれかにマッチする。もし最初の文字が`!`あるいは`^`ならば，*list* で指定した以外の全ての文字にマッチする。たとえば，‘`[a-zA-Z]`’なるパターンは，全ての大文字と小文字にマッチする。

MATLAB のバイナリデータフォーマットを使用する場合を除き，グローバル変数はグローバル属性が付いたまま保存される。したがって，もし後になって‘`load`’を使用して読み込むならば，グローバル変数として復元されることになる。

以下のコマンド

```
save -binary data a b*
```

は，変数‘`a`’と‘`b`’で始まる全ての変数を，Octave のバイナリフォーマットで，ファイル‘`data`’に保存する。

`save`の挙動を変更するための 3 つの変数と，Octave が予期せぬ終了を迎えたときに，変数を保存するかどうかをコントロールするための 3 つの変数があります。

`crash_dumps_octave_core` [Built-in Variable]
もしこの変数にゼロでない値がセットされるならば，Octave がクラッシュしたりハングアップや終了などの信号を受け取ったときに，現在の全ての変数を"octave-core"というファイルに保存しようとする。初期値は 1 ある。

`sighup_dumps_octave_core` [Built-in Variable]
もしこの変数にゼロでない値がセットされ，`crash_dumps_octave_core`もゼロでないならば，Octave がハングアップ信号を受け取ったときに，現在のすべての変数を"octave-core"というファイルに保存しようとする。初期値は 1 である。

`sigterm_dumps_octave_core` [Built-in Variable]
もしこの変数にゼロでない値がセットされていれば，Octave が終了信号を受け取ったときに，現在のすべての変数を"octave-core"というファイルに保存しようとする。初期値は 1 である。

`default_save_format` [Built-in Variable]
この変数は，`save`コマンドに対する標準フォーマットを指定する。以下の値のうち，いずれか 1 つをとるべきである：`"ascii"`，`"binary"`，`float-binary`あるいは`"mat-binary"`。標準設定の保存フォーマットは，Octave のテキストフォーマットである。

`save_precision` [Built-in Variable]

This variable specifies the number of digits to keep when saving data in text format. The default value is 17.

`save_header_format_string` [Built-in Variable]

この変数は，Octave によって保存されるテキストフォーマットのファイルの行頭において書き出されるコメント行の書式文字列を指定する。この書式文字列は，`strftime`に渡される。また，‘`#`’で始まるべきであり，改行文字を含んではいけない。もし `save_header_format_string` が空の文字列ならば，テキストデータファイルから先頭行コメントを省略する。初期値は，以下の文字列である。

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

`load` *options file v1 v2 . . .* [Command]

ファイル *file* から，指定した変数をロード（読み込み）する。`save` と同様に，変数名を列記することができるが，`load`はマッチした変数名のみを取り出すことになる。たとえば，ファイル‘`data`’に保存した変数を復元するには，以下のコマンドを使用せよ。

```
load data
```

もしファイルからグローバルのマークが付いた変数が読み出され，同じ名前を持つグローバルシンボルがすでに存在するときには，その変数はグローバルシンボルテーブルにロードされる。また，もしファイル中で変数にグローバルなマークが付いており，ローカルシンボルが存在しているならば，ローカルシンボルはグローバルテーブルに移動し，ファイルからの値が与えられる。これら両方のケースは，ある種のエラーの結果であるように見えることから，それらは警告を発する。

1 個の出力引数をつけて起動すると，Octave はシンボルテーブルに変数を挿入するかわりに，データを返す。もしそのデータファイルが数値のみを含んでいれば（桁をタブまたはスペースで区切ってあれば），それらの値を含む行列を返す。そうでなければ，`load`は，ファイル内の変数名に対応するメンバを持つ構造体を返す。

`load`コマンドは，Octave のテキストおよびバイナリ形式，MATLAB のバイナリ形式で保存されたデータを読むことができる。ファイル形式は自動的に認識され，異なる浮動小数点形式（現在のところ，IEEE ビッグまたはリトルエンディアンのみであり，その他の形式は今後追加されるであろう）からの変換が行われる。

`load`で利用できるオプションを以下の表に示す。

`-force` ‘`-force`’オプションは，後方互換性を無視して受け入れられる。現在，Octave は，その時点でメモリに存在する変数を，ファイル中に存在する同名の変数で上書きする。

`-ascii` そのファイルが Octave のテキストフォーマットであると仮定する。

警告: このオプションの意味は，MATLAB との互換性のため，Octave の将来のバージョンで変更されることがある。この変更があっても，現在のコードの意味を保つためには，かわりに`-text`オプションを使用せよ。

`-binary` そのファイルが Octave のバイナリフォーマットであると仮定する。

`-mat`
`-mat-binary`
そのファイルが MATLAB version 6 のバイナリフォーマットであると仮定する。

`-V4`
`-v4`
`-4`
`-mat4-binary`
: そのファイルが MATLAB version 4 のバイナリフォーマットであると仮定する。

`-hdf5`
: そのファイルが HDF5 フォーマットであると仮定する。(HDF5 は，University of Illinois の National Center for Supercomputing Applications によって開発された，フリーで軽量なバイナリフォーマットである）Octave は，他のソフトウエアが作成した HDF5 ファイルを読むことができる。しかし，サポートしていないデータセットは読み飛ばされる。

Octave の実行形式が HDF5 ライブラリをリンクしていないときには，HDF による保存や読み込みはできない。

`-import`
: '`-import`'は，後方互換性を無視して受け入れられる。現在，Octave は多次元 HDF データをサポートしており，Octave の識別子として妥当でない変数名について，それを自動的に修正する。

`-text`
: そのファイルが Octave のテキストフォーマットであると仮定する。

16.2 C スタイルの入出力関数

Octave の C スタイル入出力関数は，C 言語の標準入出力ライブラリの機能の大部分を提供します。しかし，入力関数のいくつかについて，引数リストはわずかに異なっています。これは，Octave には引数の参照渡しを行う方法がないためです。

これ以降において，*file* はファイル名を参照し，`fid`は `fopen` によって返される整数のファイル番号を指します。

常に利用できる 3 つのファイルが存在します。これらのファイルは，対応するファイル番号を用いてアクセスすることもできますが，以下の表で与えられるシンボル名を常に使用すべきです。なぜならば，これを使用することにより，プログラムが理解しやすくなるからです。

`stdin` [Built-in Variable]
: 標準入力ストリームである（ファイル番号 0)。Octave を対話的に使用しているとき，これはコマンドライン編集関数に渡されて（フィルタ）いる。

`stdout` [Built-in Variable]
: 標準出力ストリームである（ファイル番号 1)。標準出力に書き出される出たは，通常，ページャに渡される。

`stderr` [Built-in Variable]
: 標準エラーストリームである（ファイル番号 2)。たとえページ区切りをオンにしていても，標準エラーはページに送られない。エラーメッセージや待ち受けを出すときに有用である。

16.2.1 ファイルのオープンとクローズ

`[`*`fid`*`,` *`msg`*`] = fopen (`*`name`*`,` *`mode`*`,` *`arch`*`)` [Built-in Function]
`fid_list` `= fopen ("`*all*`")` [Built-in Function]
`file` `= fopen (`*`fid`*`)` [Built-in Function]
: `fopen`の最初の形式では，指定したモード（読み込み限定，読み書き両用など）とアーキテクチャ設定（IEEE ビッグエンディアンや IEEE リトルエンディアンなど）でファイルをオープン

し，そのファイルを後で参照するために用いる整数を返す。もしエラーが発生したならば，*fid* は −1 にセットされ，*msg* には，対応するエラーメッセージが含まれる。*mode* は，そのファイルが読み込み，書き出し，あるいはその両方の，どの目的のためにオープンされるかどうかを指定する 1 文字か 2 文字の文字列である。

`fopen`の 2 番めの形式では，`fopen`関数は，現時点でオープンしている全てのファイル（`stdin`, `stdout`および `stderr`ストリームは除く）に対応するファイル ID のベクトルを返す。

`fopen`の 3 番めの形式では，`fopen`関数は，指定したファイル ID をもっていて，現在のところオープンしているファイル名を返す。

たとえば，

```
myfile = fopen ("splat.dat", "r", "ieee-le");
```

この例は，読み出し専用でファイル‘`splat.dat`’をオープンしている。必要であれば，バイナリ形式の数値は，最下位ビットが最初に来る IEEE フォーマットで格納されていると仮定して読み込まれ，そのマシンで利用される形式に変換される。

すでにオープンされているファイルを開くと，単に再びオープンを行い，別のファイル ID を返す。異なるファイル ID を通して同じファイルに書き出すことは，予期しない結果を招くこともあるが，あるファイルを何度もオープンすることはエラーではない。

‘`mode`’がとり得る値を示す。

‘`r`’ 読み込み専用でファイルをオープンする。

‘`w`’ 書き出し専用でファイルをオープンする。以前の内容は捨てられる（上書き）。

‘`a`’ ファイルの末端から書き出しを行うためにファイルをオープンするか新規作成する。

‘`r+`’ 読み書き両用で，すでに存在しているファイルを開く。

‘`w+`’ 読み書き両用でファイルを開く。以前の内容は捨てられる（上書き）。

‘`a+`’ ファイルの末端から読み書きを行うためにファイルをオープンするか新規作成する。

モード文字列に加えて，`"t"`はテキストモードで，`"b"`はバイナリモードでオープンする。Windows および Macintosh システムでは，テキストモードは改行（LF;linefeed）を，システムにとって適切な行終端文字（Windows では復帰改行 CR-LF，Macintosh では復帰 CR）へと変換する。モードを指定しないときには，バイナリモードになる。

引数 *arch* は，そのファイルについての標準データフォーマットを指定する。*arch* に指定できる値を以下に示す。

‘`native`’ 現在のマシンのフォーマット（これが初期設定である）

‘`ieee-be`’ IEEE ビッグエンディアン

‘`ieee-le`’ IEEE リトルエンディアン

‘`vaxd`’ VAX D floating フォーマット

‘`vaxg`’ VAX G floating フォーマット

‘`cray`’ Cray floating フォーマット

しかしながら，現在のところ，この変換は‘`native`’，‘`ieee-be`’ および‘`ieee-le`’についてのみサポートしている。

`fclose (`*`fid`*`)` [Built-in Function]

指定したファイルをクローズする。成功すると，`fclose`は 0 を返し，そうでなければ −1 を返す。

16.2.2 単純な出力

`fputs` (*fid*, *string*) [Built-in Function]
ある文字列を，フォーマット無しでファイルに書き出す。

成功すると正の数を，エラーの場合は EOF を返す。

`puts` (*string*) [Built-in Function]
ある文字列を，フォーマット無しで標準出力に書き出す。

成功すると正の数を，エラーの場合は EOF を返す。

16.2.3 行単位の入力

`fgetl` (*fid*, *len*) [Built-in Function]
ファイルから，文字を読み込む。改行まで，あるいは EOF まで，または *len* 文字読み込んだ後に停止する。読み込んだ文字を，末尾の改行を除いた上で，文字列として返す。

len を省略すると，`fgetl`は次の改行までの文字を読み込む。

もし読み込むべき文字がこれ以上ないならば，`fgetl`は -1 を返す。

`fgets` (*fid*, *len*) [Built-in Function]
ファイルから，文字を読み込む。改行まで，あるいは EOF まで，または *len* 文字読み込んだ後に停止する。読み込んだ文字を，末尾の改行を付けたまま，文字列として返す。

len を省略すると，`fgetl`は次の改行までの文字を読み込む。

もし読み込むべき文字がこれ以上ないならば，`fgetl`は -1 を返す。

16.2.4 フォーマット付き出力

この節は，`printf`や関連する関数の呼び出し方について説明しています。

以下の関数は，フォーマット付き出力のために利用可能です。これらは，同名の C 言語関数を真似たものですが，それらの関数はベクトルや行列の値を表示するパフォーマンスを向上するために，異なるフォーマットテンプレートを解釈します。

`printf` (*template*, ...) [Built-in Function]
テンプレート文字列 *template* に従って，オプション引数を `stdout`に表示する。

表示した文字数を返す。

`fprintf` (*fid*, *template*, ...) [Built-in Function]
この関数は，出力が `stdout`の代わりにストリーム *fid* に書き出すことを除き，`printf`と同様である。

`sprintf` (*template*, ...) [Built-in Function]
この関数は，出力が文字列として返されることを除き，`printf`と同様である。引数として適切なサイズの文字列を提供する必要があった C ライブラリ関数とは異なり，Octave の `sprintf`関数は，変換したすべての項目が保持するため，自動的にサイズを変更した文字列を返す。

printf関数は，任意の引数の数を表示するために使用することができます。関数の呼び出しのときに与えたテンプレート文字列の引数は，追加的な引数の数についてだけでなく，その型およびそれらの出力のために使用されるべきスタイルについての情報も提供します。

テンプレート文字列内の通常の文字は，単にそのまま出力ストリームに書き出されます。一方，テンプレート内に‘%’文字で始まる変換指定（conversion specifications）は，それに続く引数をフォーマットし，出力ストリームに書き出すようにします。たとえば，

```
pct = 37;
filename = "foo.txt";
printf ("Processing of ‘%s’ is %d%% finished.\nPlease be patient.\n",
        filename, pct);
```

このコードは，以下のように出力します。

```
Processing of ‘foo.txt’ is 37% finished.
Please be patient.
```

この例は，スカラの引数を 10 進数で表示するように指定する‘%d’変換子，文字列の引数を表示するように指定する‘%s’変換子，および文字‘%’そのものを表示するための‘%%’変換子の利用を示したものです。

整数の引数を符号無し 8 進数，10 進数あるいは 16 進数として表示するための変換子もあります（それぞれ‘%o’，‘%u’あるいは‘%x’）。ならには，文字の値（‘%c’）も表示できます。

浮動小数点数値は，通常，‘%f’変換子を使用して固定小数点表記したり，‘%e’変換子を使用して指数表記で表示することができます。‘%g’ 変換子は，ある数値の大きさに対してより適切な表記によって，‘%e’と‘%f’フォーマットを使い分けます。

‘%’と適用する変換を表す文字の間に修正子を書くことにより，より詳細なフォーマットをコントロールすることができます。これは，変換の通常の挙動をわずかに変更します。たとえば，大部分の変換指定は，最小のフィールド幅やフィールド内の左寄せや右寄せにするかどうかを合図できるようにします。

特定のフラグや許容される修正子とその解釈は，特定の変換に依存します。それらは以降の節において，より詳細に解説されています。

16.2.5 行列の出力変換

行列の値が与えられたとき，Octave のフォーマット付き出力関数は，その行列の全ての値が出力されるまで，書式テンプレートを繰り返し適用します。たとえば，以下のようになります。

```
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
```

もし複数の値が，一度の呼び出しで表示されるならば，1 つの値から次の値まで移動するときには，出力関数は書式テンプレートの最初には戻りません。このことは，行列内の要素数が，書式フォーマットにおいて変換する数のちょうど倍数になっていないときには，よく分からない出力になることがあります。以下の例を参照してください。

```
printf ("%4.2f %10.2e %8.4g\n", [1, 2], [3, 4]);
```

```
⊣ 1.00   2.00e+00        3
⊣ 4.00
```

もしこれを望まないならば，1 回の呼び出しで出力しようとせずに，何度も呼び出しを繰り返してください。

16.2.6 出力変換の記述方法

このセクションでは，printfのテンプレート文字列に現れるであろう変換指定子の細かな記述方法についての詳細を述べています。

テンプレート文字列内にあって，変換指定子ではない文字は，そのままストリームに表示されます。

printfのテンプレート文字列における変換指定子は，以下のような一般的な形式をとります。

```
% flags width [ . precision ] type conversion
（% フラグ 幅 [ . 精度 ] 型 変換方法）
```

たとえば，変換指定子 '%-10.8ld'について，'-'はフラグであり，'10'はフィールド幅を指定し，精度は'8'であり，'l'という文字は型修飾子であって，'d'は変換のスタイルである（この例について言えば，数値の引数を，10 進数表記，最低 10 文字幅となるフィールドに 8 桁以内の左寄せで表示せよということである）。

より詳細に言えば，出力変換方法の指定は，最初に '%'で開始し，以下の順で並べた文字で構成されます。

- ゼロ，あるいは変換指定子の通常の挙動を修飾するフラグ文字；
- 最小フィールド幅を指定する 10 進の整数（オプション）；もし通常の変換によってこの幅よりも少ない文字数となれば，そのフィールドは指定した幅になるようにスペースを詰める。これは最小の値である。すなわち，もし通常の変換によってこの幅よりも多い文字数となれば，そのフィールドは丸められない。通常，その出力はフィールド内で右寄せとなる。

  フィールド幅を '*'と指定することもできる。これは，引数リストの次の引数が，(表示すべき実際の値ではなく）フィールド幅として使用する。その値は最も近い整数に丸められる。もしこの値が負ならば，これは '-'フラグ（以下を参照）を指定し，フィールド幅としてその絶対値を使用することを意味する。
- 数値の変換に対して，書き出す桁数を指定するための精度（オプション）；もし精度を指定するならば，10 進整数をピリオド（'.'）の後ろに置く（省略するとゼロとみなす）。

  精度には '*'を指定することもできる。これは，引数リストの次の引数が，(表示すべき実際の値ではなく）精度として使用する。この値は整数でなければならず，負であれば無視される。
- 型修飾文字列（オプション）；この文字は，Octave の printf関数では無視されるが，C 言語の printfとの互換性のために提供されていると考えておくとよい。
- 適用する変換を指定する文字；

許容される正確なオプションと，それがどのように解釈されるのかは，異なる変換指定子のあいだで変化する。使用する特定のオプションについての情報は，個々の変換に関する解説を参照してください。

16.2.7 出力変換の表

全てのさまざまな変換が行うことをまとめた表を示します。

'%d', '%i' 整数を符号付き 10 進数として表示します。詳細は Section 16.2.8 [Integer Conversions], 頁 118 を参照してください。'%d'と '%i'は，出力に対して同じ意味ですが，入力に対する scanfで使用するときには異なる意味になります（Section 16.2.13 [Table of Input Conversions], 頁 121 を参照してください）。

'%o' 整数を符号なし 8 進数として表示します。詳細は Section 16.2.8 [Integer Conversions], 頁 118 を参照してください。

'%u' 整数を符号なし 10 進数として表示します。詳細は Section 16.2.8 [Integer Conversions], 頁 118 を参照してください。

‘%x’, ‘%X’ 整数を符号なし 16 進数として表示します。‘%x’は小文字を，‘%X’は大文字を使用します。詳細は Section 16.2.8 [Integer Conversions], 頁 118 を参照してください。

‘%f’ 浮動小数点数を通常の（固定小数点）表記で表示します。詳細は Section 16.2.9 [Floating-Point Conversions], 頁 119 を参照してください。

‘%e’, ‘%E’ 浮動小数点数を指数表記で表示します。‘%e’は小文字を，‘%E’は大文字を使用します。詳細は Section 16.2.9 [Floating-Point Conversions], 頁 119 を参照してください。

‘%g’, ‘%G’ 浮動小数点数を通常の（固定小数点）表記または指数表記のどちらかで表示します。数値の大きさに対してより適切な方法で表示します。‘%g’は小文字を，‘%G’は大文字を使用します。詳細は Section 16.2.9 [Floating-Point Conversions], 頁 119 を参照してください。

‘%c’ 1 文字を表示します。Section 16.2.10 [Other Output Conversions], 頁 119 を参照してください。

‘%s’ 文字列を表示します。Section 16.2.10 [Other Output Conversions], 頁 119 を参照してください。

‘%%’ ‘%’という文字そのものを表示します。Section 16.2.10 [Other Output Conversions], 頁 119 を参照してください。

もし変換識別子の記述方法が妥当なものでなければ，予測できないことが起こり，表示が行われないでしょう。もしテンプレート文字列内の全ての変換指定子について値を与えるために提供する引数が充分でなければ，あるいはその引数が正しい型でなければ，結果は予測不能です。もし変換指定子よりも多くの引数を与えるならば，余分な引数値は単に無視されます；これは，ときに有用です。

16.2.8 整数の変換

この節では，‘%d’，‘%i’，‘%o’，‘%u’，‘%x’および‘%X’変換指定子のオプションについて解説します。これらの変換では，さまざまなフォーマットで整数を表示します。

‘%d’と‘%i’変換指定子は，数値引数を符号付き 10 進数として表示します；一方で，‘%o’，‘%u’および‘%x’は数値引数を，それぞれ符号なし 8 進数，10 進数，16 進数として表示します。‘%X’変換指定子は，‘%x’と同じですが，桁として‘abcdef’のかわりに‘ABCDEF’という文字を使用することが異なっています。

以降のフラグが意味を持ちます。

‘-’ そのフィールドで左寄せにします（通常の右寄せに代えて）。

‘+’ 符号付きの‘%d’および‘%i’変換指定子について，その値が正であればプラス記号を表示します。

‘ ’ 符号付きの‘%d’および‘%i’変換指定子について，その結果がプラスまたはマイナス記号で開始しなければ，かわりに空白文字を先頭に付けます。‘+’フラグは結果に記号が含まれるようにしますが，そのフラグとこれを両方とも与えるならば，このフラグは無視されます。

‘#’ ‘%o’変換子について，このフラグは，あたかも精度が増えたかのように，先頭の桁が‘0’になるようにします。‘%x’または‘%X’について，このフラグは，結果の先頭にそれぞれ‘0x’または‘0X’を付加します。これは，‘%d’，‘%i’あるいは‘%u’変換指定子については何も有効ではありません。

‘0’ スペースの代わりに，フィールドをゼロで埋めます。符号あるいはベースを認識した後にゼロを配置します。‘-’フラグも指定されているか，精度が指定されていれば，このフラグは無視されます。

もし精度が与えられていれば，これは表示する最小桁数を指定します。もし必要であれば，先頭にゼロが含まれます。もし精度を指定しないならば，その数値は，必要になるできるだけ多くの桁数で表示します。もし精度ゼロで 0 という値を変換すると，何も文字が表示されません。

16.2.9 浮動小数点数の変換

この節では，浮動小数点数に対する変換指定子，すなわち ‘%f’，‘%e’，‘%E’，‘%g’および ‘%G’ について論じます。

‘%f’変換子は，その引数を，$[-]ddd.ddd$ なる形式の固定小数点表記で表示します。ここで小数点以下の桁数は，指定した精度によってコントロールされます。

‘%e’変換子は，その引数を，$[-]d.ddde[+|-]dd$ なる形式の指数表記で表記します。また，小数点以下の桁数は，指定した精度によってコントロールされます。指数は，常に少なくも 2 桁が含まれます。‘%E’ 変換子も同様ですが，指数を ‘e’の代わりに ‘E’で表すことが異なります。

‘%g’および ‘%G’変換子は，指数部分が-4 以下もしくは精度以上であるときには，それぞれ ‘%e’または ‘%E’の形式で表示します。そうでなければ，‘%f’形式を使用します。末尾のゼロは結果の分数部分から取り除かれ，後ろに桁が続くときにのみ小数点が表示される。

以下のフラグは，その挙動を修飾するために使用できます。

‘-’ そのフィールドで左寄せにします。通常，結果は右寄せになります。

‘+’ 結果に，常にプラスまたはマイナスの符号を含めます。

‘ ’ もしその結果がプラスまたはマイナスの符号で開始しなければ，かわりに先頭にスペースを付加します。‘+’フラグは，その結果に符号を含むようにするので，このフラグと両方を当てると，このフラグは無視されます。

‘#’ たとえ小数点以下に桁が無くとも，結果が常に小数点を含むように指定します。‘%g’および ‘%G’変換子について，このフラグは，小数点以降に続くゼロが，取り除かれずにそのままにする。

‘0’ スペースの代わりに，フィールドをゼロで埋めます。符号あるいはベースを認識した後にゼロを配置します。‘-’フラグも指定されているか，精度が指定されていれば，このフラグは無視されます。

精度指定子は ‘%f’，‘%e’および ‘%E’変換子に対して，小数点以下に何桁続くかを指定します。これらの変換子について，初期設定の精度は 6です。もしその精度が明示的に 0であれば，小数点文字を表示しません。‘%g’および ‘%G’変換子について，その精度は，表示すべき有効桁が何桁かを指定します。有効桁は小数点以前の最初の桁と，小数点以下の全ての桁です。‘%g’および ‘%G’に対して，精度が 0または未指定ならば，1という値のように扱われます。もし表示される値が指定した桁数では詳細に表現できないならば，その値は適合する最も近い数に丸められます。

16.2.10 他の出力の変換

この節では，printfに対するその他の変換子について解説します。

‘%c’変換子は 1 文字を表示します。‘-’フラグは，そのフィールドで左寄せを指定するために使用できますが，他のフラグは何も指定されず，精度あるいは型修飾子は何も与えることはできません。たとえば，

```
printf ("%c%c%c%c%c", "h", "e", "l", "l", "o");
```

これは‘hello’と表示します。

‘%s’変換子は文字列を表示します。対応する引数は，文字列でなければなりません。精度は，書き出す最大の文字数を指示するために指定することができます。一方，文字列内の null で終わるまでの文字（null は含まない）は，出力ストリームに書き出されます。‘-’フラグは，そのフィールドで左寄せを指定するために使用できますが，他のフラグは何も指定されず，精度あるいは型修飾子は何も与えることはできません。たとえば，

```
printf ("%3s%-6s", "no", "where");
```

は，‘ nowhere ’（先頭と末尾にスペースを含みます）と表示します。

16.2.11 フォーマット付き入力

Octave は，フォーマット付き入力を読み込むために，scanf，fscanfおよび sscanf関数を提供しています。これらの関数の各々には，2 つの使い方が存在します。1 つめは，ファイルからデータのベクトルを展開するため，もうひとつは，より「C ライク」なものです。

`[`*val*`,` *count*`] = fscanf (`*fid*`,` *template*`,` *size*`)` [Built-in Function]
`[`*v1*`,` *v2*`, ..., ` *count*`] = fscanf (`*fid*`,` *template*`, "`*C*`")` [Built-in Function]

1 番めの形式において，*template* にならって *fid* から読み込み，行列 *val* として結果を返す。

オプション引数 *size* は，読み込むべきデータの総量を指定する。これは，以下のうちどれか 1 つをとる。

`Inf` 可能な限り読み込み，列ベクトルを返す。

nr *nr* 個めの要素まで読み込み，列ベクトルを返す。

`[`*nr*`, Inf]` 可能な限り読み込み，*nr* 行の行列を返す。もし読み込んだ要素数がきちんと *nr* の倍数になっていないならば，最後の列はゼロで埋められる。

`[`*nr*`,` *nc*`]` *nr* * *nc*個の要素まで読み込み，*nr* 行の行列を返す。もし読み込んだ要素数がきちんと *nr* の倍数になっていないならば，最後の列はゼロで埋められる。

size を省略すると，Infという値を仮定する。

もし *template* に文字変換のみを指定すると，文字列を返す。

正常に読み込めた項目の数は，*count* に返る。

2 番めの形式において，単一スカラの戻り値に対応する *template* の各変換識別子として，*template* にならって *fid* から読み込む。この形式は，より「C 言語寄り」であり，Octave の以前のバージョンと互換性がある。変換に成功した数は，*count* に返される。

`[`*val*`,` *count*`] = sscanf (`*string*`,` *template*`,` *size*`)` [Built-in Function]
`[`*v1*`,` *v2*`, ..., ` *count*`] = sscanf (`*string*`,` *template*`, "`*C*`")` [Built-in Function]

これはfscanfのような関数であるが，文字をストリームの代わりに文字列 *string* から得ることが異なっている。その文字列の終端に達することは，ファイルの末端（end-of-file）の状態であるとして扱われる。

scanfを呼び出すことは，任意の引数がテンプレート文字列のコントロール下で読み込まれるということについて，表面的には printfを呼ぶことと等価です。テンプレート文字列内の変換識別子の表記法が printfのそれと非常に似ている一方で，テンプレートの解釈は，固定フィールドではなく，フォーマットなし入力で単にパターンマッチング向きです。たとえば，大部分の scanf変換では，

入力ファイル内の任意の数の「ホワイトスペース」（空白文字，タブおよび改行）を読み飛ばし，さらに，数値入力については，出力変換に存在するような精度の概念がありません。通常は，テンプレート内のホワイトスペース以外の文字は，入力ストリーム内の文字に正確に一致すると期待されます。

マッチングの失敗が発生するとき，scanfは直ちに終了し，マッチしなかった最初の文字をそのストリームから読み込むべき次の文字として残します。そして scanfはうまく変換した全ての項目を返します。

フォーマット付き入力関数は，フォーマット付き出力関数と同じ頻度で使用されることはありません。理由の一部としては，それら関数を適切に使うには注意を要するからです。別の理由は，マッチエラーから復帰することが難しいことです。

16.2.12 入力変換の記述方法

scanfのテンプレート文字列は，‘%’で始まる変換識別子を交えた通常の文字列を含みます。

テンプレートにある任意のホワイトスペース文字は，入力ストリームにおける任意の数のホワイトスペースを読み込んで捨てます。マッチするホワイトスペース文字は，テンプレート文字列に登場するホワイトスペースと厳密に同じである必要はありません。たとえば，前後にホワイトスペースを伴うカンマを認識させるには，テンプレートに‘ , ’を書いてください。

テンプレート文字列内の，変換識別子の一部ではない他の文字は，入力ストリームにおける文字列に厳密に一致しなければならない。もしこのケースに合わなければ，マッチングの失敗が発生します。

scanfのテンプレート文字列内の変換指定子は，以下のような一般的な形式になります。

```
% flags width type conversion
```

より詳細に言えば，入力変換方法の指定は，最初に‘%’で開始し，以下の順で並べた文字で構成されます。

- フラグ文字‘*’（オプション）；これは，この識別子によって読み込んだテキストを無視するということである。scanfがこのフラグを用いた識別子を見つけたときには，変換識別子の残りによって指示されたように入力から読み込むが，この入力は捨てて何の値も返さない。また，うまく代入できた階数のカウントをインクリメントされない。
- 最大フィールド幅を指定する 10 進の整数（オプション）；この最大数に達したとき，あるいはマッチしない文字が見つかったときには，入力ストリームからの読み込みを停止する。大部分の変換子は，最初のホワイトスペース文字を捨てられ，これら読み捨てた文字は最大フィールド幅に向けてカウントとされない。最初のホワイトスペースを読み捨てない変換は，きちんと説明する。
- 型修飾文字（オプション）；この文字は，Octave の scanf関数では無視されるが，C 言語の scanf との互換性のために提供されていると考えておくとよい。
- 適用する変換を指定する文字；

許容される正確なオプションと，それがどのように解釈されるのかは，異なる変換指定子のあいだで変化する。使用する特定のオプションについての情報は，個々の変換に関する解説を参照してください。

16.2.13 入力変換の表

各種変換指定子をまとめた表を示します。

‘%d’ 10 進数で書かれた符号付き（あるいは符号なし）整数にマッチします。Section 16.2.14 [Numeric Input Conversions], 頁 122 を参照してください。

‘%i’ C 言語で整数の定数を指定するための任意のフォーマットによる符号付き（あるいは符号なし）整数にマッチします。Section 16.2.14 [Numeric Input Conversions], 頁 122 を参照してください。

‘%o’ 8進数表記で書かれた符号付き（あるいは符号なし）整数にマッチします。Section 16.2.14 [Numeric Input Conversions], 頁 122 を参照してください。

‘%u’ 10 進数で書かれた符号なし整数にマッチします。Section 16.2.14 [Numeric Input Conversions], 頁 122 を参照してください。

‘%x’, ‘%X’ 16 進数で書かれた符号なし整数にマッチします。Section 16.2.14 [Numeric Input Conversions], 頁 122 を参照してください。

‘%e’, ‘%f’, ‘%g’, ‘%E’, ‘%G’
符号付き（あるいは符号なし）浮動小数点数にマッチします。Section 16.2.14 [Numeric Input Conversions], 頁 122 を参照してください。

‘%s’ ホワイトスペース以外の文字のみを含む文字列にマッチします。Section 16.2.15 [String Input Conversions], 頁 122 を参照してください。

‘%c’ 1 文字以上の文字列にマッチします。読み込んだ文字数は，変換子に与えられた最大フィールド幅によってコントロールされます。Section 16.2.15 [String Input Conversions], 頁 122 を参照してください。

‘%%’ これは，入力ストリーム内で ‘%’文字そのものにマッチします。対応する引数はありません。

もし変換指定子の記述が妥当なものでなければ，その挙動は定義されません。代入を実行するテンプレート文字列における全ての変換指定子についてアドレスを提供するために与えられる十分な関数がない，あるいはその引数が正しい型でないならば，その挙動も定義されません。一方で，余分な引数は単に無視されます。

16.2.14 数値入力の変換子

この節は数値を読み込むための scanf変換子について説明します。

‘%d’変換子は，10 進の符号付き（あるいは符号なし）整数にマッチします。

C 言語で整数の定数を指定するための任意のフォーマットによる符号付き（あるいは符号なし）整数にマッチします。

たとえば，‘%i’変換子のもとでは，‘10’，‘0xa’あるいは ‘012’のどの文字列も整数として読み込むことができます。これら各々は 10 進数 10という数値を指定します。

‘%o’，‘%u’および ‘%x’は，それぞれ 8 進数，10 進数および 16 進数の符号なし整数にマッチします。

‘%X’変換子は ‘%x’と同一です。それらは桁として大文字と小文字のどちらも許容します。

C 言語の scanfとは異なり，Octave は ‘h’，‘l’および ‘L’ 修飾子を無視します。

16.2.15 文字列入力の変換子

この節では，文字列と文字を読み込むための scanf入力変換子（‘%s’ と ‘%c’）について説明しています。

‘%c’変換子は最もシンプルです。これは常に固定文字数にマッチします。最大フィールドは，何文字読み込むかを伝えます。もし最大値を指定しなければ，初期値は 1 です。この変換子は，最初にあるホワイトスペース文字を読み飛ばしません。この変換子は，精密に次の n 文字を読み込み，読み込めないならば失敗します。

‘%c’変換子は，ホワイトスペース以外の文字列にマッチします。これは最初にあるホワイトスペース文字を読み飛ばしますが，読み込んだ文字列の後ろでホワイトスペースに遭遇すると停止します。

たとえば，以下の入力を読み込みます。

```
hello, world
```

これを‘%10c’で変換すると，" hello, wo"という文字列になりますが，同じ入力を‘%10s’変換子は"hello,"を生み出します。

16.2.16 バイナリ入出力

Octave は freadと fwrite関数を使用することにより，バイナリデータを読み書きすることができます。これは同名の標準 C 言語を真似たものです。この関数は，整数データのバイト順を自動的に入れ替えることができ，データが読み込まれたときにサポートする浮動小数点フォーマット間で変換することができます。

[*val*, *count*] = fread (*fid*, *size*, *precision*, *skip*, *arch*) [Built-in Function]

指定したファイル ID*fid* から，型 *precision* のバイナリデータを読み込む。

オプション引数 *size* は，読み込むべきデータの総量を指定する。これは，以下のうちの 1 つをとる。

Inf 可能な限り読み込み，列ベクトルを返す。

nr *nr* 個めの要素まで読み込み，列ベクトルを返す。

[*nr*, Inf] 可能な限り読み込み，*nr* 行の行列を返す。もし読み込んだ要素数がきちんと *nr* の倍数になっていないならば，最後の列はゼロで埋められる。

[*nr*, *nc*] *nr* * *nc*個の要素まで読み込み，*nr* 行の行列を返す。もし読み込んだ要素数がきちんと *nr* の倍数になっていないならば，最後の列はゼロで埋められる。

size を省略すると，Infという値を仮定する。

追加の引数 *precision* は，読み込むべきデータの型を指定する文字列であり，以下のうちの 1 つをとる。

"schar"
"signed char"
符号付き文字

"uchar"
"unsigned char"
符号なし文字

"int8"
"integer*1"
8 ビット符号付き整数

"int16"
"integer*2"
16 ビット符号付き整数

"int32"
"integer*4"
32 ビット符号付き整数

"int64"
"integer*8"
64 ビット符号付き整数

"uint8"　8 ビット符号なし整数

"uint16"　16 ビット符号なし整数

"uint32"　32 ビット符号なし整数

"uint64"　64 ビット符号なし整数

"single"
"float32"
"real*4"　32 ビット浮動小数点数

"double"
"float64"
"real*8"　64 ビット浮動小数点数

"char"
"char*1"　単一文字

"short"　単精度整数（サイズはプラットフォーム依存）

"int"　整数（サイズはプラットフォーム依存）

"long"　倍精度整数（サイズはプラットフォーム依存）

"ushort"
"unsigned short"
　　符号なし単精度整数（サイズはプラットフォーム依存）

"uint"
"unsigned int"
　　符号なし整数（サイズはプラットフォーム依存）

"ulong"
"unsigned long"
　　符号なし倍精度整数（サイズはプラットフォーム依存）

"float"　単精度浮動小数点数（サイズはプラットフォーム依存）

初期設定の精度は，"uchar"である。

引数 *precision* は，任意の反復カウントを指定することもできる。たとえば，'32*single'は 32 個の単精度浮動小数点数のブロックを読み込むことになる。ブロックによる読み込みは，引数 *skip* と組み合わせると有用である。

引数 *precision* は，型変換も指定することができる。たとえば，'int16=>int32'は 16 ビット整数値を読み込んで 32 ビット整数の配列を返す。標準設定により，freadは倍精度配列を返す。特別な表記 '*TYPE'は 'TYPE=>TYPE'を短縮したものである。

変換と反復回数は組み合わせることができる。たとえば，'32*single=>single'は単精度浮動小数点値のブロックを読み込み，標準の倍精度値配列のかわりに単精度値を返すようになる。

追加の引数 *skip* は，各々の要素（あるいは要素のブロック）を読み込んだ後，読み飛ばすべきバイト数を指定する。もしこれを指定しないならば，0 が指定されたものと仮定する。もし読み込んだ最後のブロックが完了しないならば，最初のスキップは省略される。たとえば，

```
fread (f, 10, "3*single=>single", 8)
```

この式は，最後の 8 バイトのスキップを省略する。なぜならば，最後の読み込みが，3 値の完全なブロックにはならないからである。

オプション引数 *arch* は，そのファイルについてのデータフォーマットを指定する文字列である。以下の値が利用できる。

`"native"` そのマシンのフォーマット

`"ieee-be"`
IEEE ビッグエンディアン

`"ieee-le"`
IEEE リトルエンディアン

`"vaxd"` VAX D floating フォーマット

`"vaxg"` VAX G floating フォーマット

`"cray"` Cray floating フォーマット

現在のところ，この変換は ‘`native`’，‘`ieee-be`’および ‘`ieee-le`’ についてのみサポートしている。

そのファイルから読み込んだデータは，*val* に返される。また，読み込んだ値の数は `count`に返される。

count = **fwrite** (*fid*, *data*, *precision*, *skip*, *arch*) [Built-in Function]
型 *precision* のバイナリデータを，指定したファイル ID*fid* に書き出し，正常にファイルへ書き込んだ値の数を返す。

引数 *data* は，ファイルに書き込まれることになる値の行列である。この値は，行方向に要素が連続しているように展開される（Fortran の配列順）。

残りの引数 *precision*，*skip* および *arch* は，オプションであり，`fread`の解説にあるように解釈される。

もし *data* の値が大きすぎて指定した精度に当てはまらないならば，`fwrite`の挙動は定義されない。

16.2.17 テンポラリファイル

[*fid*, *name*, *msg*] = **mkstemp** (*template*, *delete*) [Built-in Function]
template から生成される一意な名前を持つ，新規テンポラリファイルに対応するファイル ID を返す。*template* の最後の 6 文字は `XXXXXX`でなければならず，それらは一意なファイル名を作るための文字列に置き換えられる。そのファイルは，読み書き両用モードで，システムに依存したパーミッション（GNU/Linux システムでは，glibc 2.0.7 以降に対して 0600 となる）で生成される。このファイルは，`O_EXCL`フラグ付きでオープンされる。

もしオプション引数 *delete* が与えられ，これが真ならば，そのファイルは Octave の終了時，または `purge_tmp_files`関数が呼ばれたときに自動的に削除される。

この関数が成功すると，*fid* は妥当なファイル ID，*name* はファイル名，および *msg* は空文字列となる。そうでなければ，*fid* は −1，*name* は空に，そして *msg* はシステム依存のエラーメッセージとなる。

[*fid*, *msg*] = **tmpfile** () [Built-in Function]
一意な名前を持つ新規テンポラリファイルに対応するファイル ID を返す。このファイルはバイナリの読み書き両用モード（`"w+b"`）でオープンされる。このファイルは，クローズされるか Octave が終了するときに自動的に削除される。

この関数が成功すると，*fid* は妥当なファイル ID，*msg* は空文字列となる。そうでなければ，*fid* は −1，*msg* はシステム依存のエラーメッセージとなる。

`tmpnam` (*dir*, *prefix*) [Built-in Function]

一意なテンポラリファイル名を文字列として返す。

prefix を省略すると，`"oct-"`という値が使用される。*dir* も省略すると，テンポラリファイルを作成するための標準的なディレクトリが使用される。もし *dir* を与えるならば，それが存在していなければならない。存在していなければ，標準的なディレクトリが使用される。`tmpnam` によって名付けられたファイルはオープンされていないので，自分でオープンしようとするときには，すでに利用可能では無くなっているという可能性がある（それほど起こりえないことだろうが）。

16.2.18 ファイルの終端とエラー

`feof` (*fid*) [Built-in Function]

与えられたファイルが，すでに読み終わった状態（end-of-file）にあれば 1 を，そうでなければ 0 を返す。これは，そのファイルがすでにファイル末尾まで読み終わっているときに 1 を返すものであり，次の操作によってその状態になるだろうということではないことに注意せよ。

`ferror` (*fid*) [Built-in Function]

与えられたファイルについてエラー状態にあれば 1 を，そうでなければ 0 を返す。これはエラーがすでに起こったときにのみ 1 を返すことになり，次の操作がエラー状態に陥るだろうということではないことに注意せよ。

`freport` () [Built-in Function]

オープンされているファイル名，読み込み専用，書き出し専用，あるいは読み書き両用かどうかのリストを表示する。例を示す。

```
freport ()

  -|  number  mode  name
  -|
  -|       0     r  stdin
  -|       1     w  stdout
  -|       2     w  stderr
  -|       3     r  myfile
```

16.2.19 ファイル内のポジション指定

与えられたファイルに対するファイルポインタのポジションをセットし，決定するために 3 つの関数が利用できます。

`ftell` (*fid*) [Built-in Function]

ファイルポインタの位置を，ファイル *fid* の先頭からの文字数として返す。

`fseek` (*fid*, *offset*, *origin*) [Built-in Function]

ファイルポインタを，ファイル *fid* 内の任意の位置へセットする。

そのポインタは，*origin* から *offset* 文字めに配置する。*origin* は，あらかじめ定義された変数 `SEEK_CUR`（現在の位置），`SEEK_SET` （ファイルの先頭），`SEEK_END`（ファイルの末端），あ

るいは文字列"cof", "bof", "eof"のいずれか 1 つを使うのがよい。オフセットはゼロでなければならず，そうでなければ `ftell`によって返される値とする（その場合，*origin* は `SEEK_SET` でなければならない)。

成功すると 0，エラーの場合は −1 を返す。

`SEEK_SET` [Built-in Variable]
`SEEK_CUR` [Built-in Variable]
`SEEK_END` [Built-in Variable]

これらの変数は，関数 `fseek`の 3 番めの引数として使用されることになる。

`SEEK_SET` ファイルの先頭からの相対位置を指定する。

`SEEK_CUR` 現在位置からの相対位置を指定する。

`SEEK_END` ファイルの末端からの相対位置を指定する。

`frewind (`*fid*`)` [Built-in Function]

ファイル *fid* のファイルポインタを，先頭に移動する。成功すると 0 を，エラーが発生すると −1 を返す。これは `fseek (`*fid*`, 0, SEEK_SET)`と等価である。

以下の例は，現在のファイルポジションを変数 `marker`に格納し，ファイルの先頭にポインタを移動させ，4 文字読み込み，もとのポジションを返します。

```
marker = ftell (myfile);
frewind (myfile);
fourch = fgets (myfile, 4);
fseek (myfile, marker, SEEK_SET);
```

17 プロット

Octave の全てのプロット関数は，実際のグラフィックを扱うために gnuplot を使用します。大部分のプロット型は，基本的なプロット関数を使用して生成されます。これは MATLAB における同等の関数を真似て作られています。これらの関数を利用することは，一般には率直であり，プロットを生成するための好ましい方法です。しかしながら，gnuplotに馴染んだユーザや，基本的なコマンドが適切でないような場面での特殊な応用に対しては，Octave は 2 つの低水準関数，gplotと gsplot も提供しています。これらは，対応する gnuplotの関数 plotと splotのようにほぼ正確に動作します。また，最近のバージョンから拡張された MATLAB の機能，たとえば handle-graphics と関連する関数は実装されていないことに注意してください。

17.1 2 次元プロット

MATLAB スタイルの 2 次元プロットコマンドは，以下のようなものです。

plot (*args*) [Function File]

この関数は，2 次元プロットを出力します。引数の多数の異なる組み合わせが可能です。最も単純な形式は，

```
plot (y)
```

である。ここで，その引数は *y* に連動する集合として受け入れられ，*x* に連動する値は，1 から開始する要素のインデックスであると受け入れられる。

1 つ以上の引数を与えると，それらは以下のように解釈される。

```
plot (x, y, fmt ...)
```

ここで *y* と *fmt* はオプションであり，任意の数の引数セットが現れてもよい。*x* と *y* の値は，以下のように解釈される。

- もし 1 個のデータ引数が与えられれば，それが *y* に連動する集合として受け入れられ，*x* に連動する値は，1 から開始する要素のインデックスであると受け入れられる。
- もし 1 番めの引数がベクトルであり，2 番めが行列ならば，そのベクトルは，行列の列（あるいは行）と対でプロットとされる（最初に列について試し，組み合わせがマッチするものを使用する）。
- もし 1 番めの引数が行列であり，2 番めがベクトルならば，行列の列（あるいは行）はそのベクトルと対でプロットされる（最初に列について試し，組み合わせがマッチするものを使用する）。
- もし両方ともベクトルならば，*y* の要素は *x* の要素と対でプロットされる。
- もし両方とも行列ならば，*y* の列は *x* の列と対でプロットされる。このケースにおいて，両方の行列は同じ行数と列数でなければならず，行数を合わせるために引数を転置しようと試みることはない。

  もし両方の引数がスカラならば，1 個の点がプロットされる。

もし *fmt* 引数を与えるならば，以下のように解釈される。もし *fmt* が与えられなければ，標準の gnuplot 直線スタイルを仮定する。

‘-’ 直線プロットスタイルをセットする（初期値）。

‘.’ ドットプロットスタイルをセットする。

‘@’ ポイントプロットスタイルをセットする。

‘-@’ 直線・ポイントプロットスタイルをセットする。

‘^’ インパルスプロットスタイルをセットする。

‘L’ ステッププロットスタイルをセットする。

‘*n*’ もし *n* が 1 から 6 の範囲にあれば，プロット色として解釈する。

‘*nm*’ もし *nm* が 2 桁の整数であり，*m* が 1 から 6 の範囲にあれば，*m* がポイントスタイルとして解釈される。これは，@あるいは-@指定子を組み合わせるときのみ妥当である。

‘*c*’ もし *c* が，"r"，"g"，"b"，"m"，"c"，"w"のうちの 1 つならば，プロット色として解釈される（赤，緑，青，マゼンタ，シアン，白）。

‘";title;"’
"title"はキーについてのラベルである。

‘+’
‘*’
‘o’
‘x’ ポイントあるいはラインポイントスタイルと組み合わせるとき，ポイントスタイルをセットする。

カラーラインスタイルは，色をサポートするターミナルにおいて，以下のような意味を持つ。

```
Number  Gnuplot colors  (lines)points style
  1       red                   *
  2       green                 +
  3       blue                  o
  4       magenta               x
  5       cyan                house
  6       brown            there exists
```

引数 *fmt* はキータイトルを割り当てるためにも使用できる。そうするには，上で説明したフォーマット項目の後ろに，希望するタイトルをセミコロンで挟んで含める。たとえば"+3;Key Title;"のようにする。最後のセミコロンは必要であり，付けないとエラーが発生する。

いくつかのプロット例を示す：

```
plot (x, y, "@12", x, y2, x, y3, "4", x, y4, "+")
```

このコマンドは，yをタイプ 2（‘+’で表示される）の点と色 1 （赤）で，y2を直線で，y3を色 4（マゼンタ）で，y4を ‘+’で表示される点でプロットする。

```
plot (b, "*")
```

個のコマンドは，ベクトル b内のデータが ‘*’で表示される点でプロットする。

```
t = 0:0.1:6.3;
plot (t, cos(t), "-;cos(t);", t, sin(t), "+3;sin(t);");
```

これは余弦関数および正弦関数と，キーに対応するラベルをプロットする。

hold *args* [Built-in Function]

直後にプロットコマンドを実行するときに，プロット上の現在のデータを「ホールド」することを Octave に指示する。これは，一連のプロットを実行できるようにし，同じ図にすべてのラインを描画する。初期設定は，各々の新たなプロットコマンドについて，プロットするデバイスを最初にクリアするようにする。たとえば，

```
hold on
```

このコマンドは，ホールド状態をオンにする。offという引数は，ホールド状態をオフにし，何も引数を付けずにholdを実行すると，オンとオフを切り替える。

ishold [Built-in Function]
もし次のラインが現在のプロットに追加されるならば 1 を，次のラインを描画する前にプロットデバイスがクリアされるならば 0 を返す。

clearplot [Built-in Function]
clg [Built-in Function]
プロットウインドウと，タイトル，軸ラベルをクリアする。clgは，MATLAB との互換性のための，clearplotの短縮名である。

gplot clear，*gsplot clear*および *replot clear*コマンドは，clearplotに等しい（以前には，*gplot clear*のようなコマンドは，clearとして評価され，全ての可視変数をクリアすることになっていた）。

shg [Function File]
グラフウインドウを表示する。現在のところ，これは再プロットを実行することと同じである。

closeplot [Built-in Function]
gnuplotサブプロセスへのストリームをクローズする。もし X11 を使用しているならば，これはプロットウインドウをクローズする。

purge_tmp_files [Built-in Function]
プロットコマンドによって作成されたテンポラリファイルを削除する。

Octave は gnuplotのためにテンポラリデータファイルを作成し，パイプを通して gnuplotにコマンドを送る。Octave は終了時にテンポラリファイルを削除するが，多くのプロットを実行するならば，セッションの途中でファイルを削除したいと思うかもしれない。

Octave の将来のバージョンでは，プロットデータを保持するためのテンポラリファイルを使用する必要性がなくなるであろう。

axis (*limits*) [Function File]
プロットについて軸の限界値をセットする。

引数 *limits* は，2，4 または 6 個の要素をもつベクトルとするべきである。1 番めと 2 番めの要素は，x 軸の下限と上限を指定する。3 番めと 4 番めの要素は，y 軸の限界値を指定する。また，5 番めと 6 番めは z 軸の限界値を指定する。

何も引数を付けないときには，axisは自動スケール化をオンにします。

出力引数を 1 つ付けたときには，x=axisは現在の軸を返す（これは自動軸設定については実装されていない）。

限界値を指定するベクトル引数はオプションであり，追加的な文字列引数は様々な軸を適切に指定するために使用できる。たとえば，

```
axis ([1, 2, 3, 4], "square");
```

このコードは，正方アスペクト比とし，

```
axis ("labely", "tic");
```

これは，全ての軸に目盛りマークを付け，y 軸にのみ目盛りラベルを付ける。

以下のオプションは，軸のアスペクト比をコントロールする。

`"square"` 正方アスペクト比とする。

`"equal"` x の距離を y の距離に等しくする。

`"normal"` バランスを戻す。

以下のオプションは，軸の限界値が解釈される方法をコントロールする。

`"auto"` データ周辺で良いとおぼしき限界値となるように設定した軸をセットする。あるいは軸が何も設定されていないときにはすべてセットする。

`"manual"` 現在の軸限界値を修正する。

`"tight"` データの限界値に軸を合わせる（まだ実装されていない）。

オプション`"image"`は，`"tight"`または`"equal"` と等価である。

以下のオプションは，目盛りの外観に影響する。

`"on"` 目盛りおよび全ての軸のラベルをオンにする。

`"off"` 目盛りおよび全ての軸のラベルをオフにする。

`"tic[xyz]"`
目盛りをオンにする。あるいは指定した軸についてはオン，それ以外についてはオフにする。

`"label[xyz]"`
ラベルをオンにする。あるいは指定した軸についてはオン，それ以外についてはオフにする。

`"nolabel"`
全ての軸のラベルをオフにする。

もし軸に何も目盛りを付けないならば，ラベルも付加されない。

以下のオプションは，軸における値の増分の方向に影響する。

`"ij"` y 軸を逆にし，低い値が上側に来る。

`"xy"` y 軸を戻し，高い値が上側に来る。

17.2 専門的な 2 次元プロット

`bar` (*x*, *y*) [Function File]

x-y データの 2 つのベクトルを与え，バーグラフを描画する。

もし 1 つの引数のみが与えられるならば，y 値のベクトルとして採用され，x 軸の値は要素のインデックスとして与えられる。

もし 2 つの出力引数が指定されたならば，そのデータは生成されるがプロットされる。たとえば，

```
bar (x, y);
```

および

```
[xb, yb] = bar (x, y);
plot (xb, yb);
```

の 2 つは等価である。

`contour (z, n)` [Function File]
`contour (x, y, z, n)` [Function File]

z によって記述された 3 次元外観の等高線プロットをする。この関数が非常に有用になる前には，gnuplotの等高線ルーチンを高める必要がある。

`hist (y, x, norm)` [Function File]

ヒストグラムカウントあるいはプロットをする。

入力の引数に 1 個のベクトルを付けると，10 個の bin で，その値のヒストグラムをプロットする。ヒストグラム bin の幅は，データの範囲によって決定される。

2 番めの引数にスカラを与えると，bin の数として使用する。

2 番めの引数にベクトルを与えると，bin の中心値として使用し，そのベクトルにおいて調整された値から決定された bin の幅を使用する。

もし 3 番めの引数が与えられるならば，そのヒストグラムは，バーの和が *norm* に等しくなるように標準化される。

外れ値は，最初および最後の bin にまとめられる。

2 つの出力引数を付けると，`bar (xx, nn)`がヒストグラムとなるような *nn* と *xx* を返す。

`loglog (args)` [Function File]

両軸に対して対数スケールを使用して 2 次元プロットする。`loglog` 関数が受け付ける引数の説明は，`plot`の解説を参照せよ。

`polar (theta, rho, fmt)` [Function File]

極座標 *theta* と *rho* を与えて 2 次元プロットをする。

オプションの 3 番めの引数は，ラインのスタイルを指定する。

`semilogx (args)` [Function File]

x 軸を対数スケールにして 2 次元プロットをする。`semilogx` 関数が受け付ける引数の説明は，`plot`の解説を参照せよ。

`semilogy (args)` [Function File]

x 軸を対数スケールにして 2 次元プロットをする。`semilogy` 関数が受け付ける引数の説明は，`plot`の解説を参照せよ。

`stairs (x, y)` [Function File]

x-y データの 2 つのベクトルを与え,「階段」プロットをする。

もし 1 つの引数のみが与えられるならば，y 値のベクトルとして採用され，x 軸の値は要素のインデックスとして与えられる。

もし 2 つの出力引数が指定されたならば，そのデータは生成されるがプロットされる。たとえば,

```
stairs (x, y);
```

および

```
[xs, ys] = stairs (x, y);
plot (xs, ys);
```

の 2 つは等価である。

`errorbar (args)` [Function File]

この関数は，エラーバー付きの 2 次元プロットをする。引数の多くの組み合わせが可能である。最も単純な使用方法は，以下のようになる。

```
errorbar (y, ey)
```

ここで最初の引数は *y* 軸値の集合として採用され，2 番めの引数 *ey* は，*y* 値のエラーとして受け入れられる。*x* 軸の値は，1 から始まる要素のインデックスとなる。

もし 2 つ以上の引数が与えられるならば，それらは以下のように解釈される。

```
errorbar (x, y, ..., fmt ...)
```

ここで *x* と *y* の後ろには，*ey*，*ex*，*ly*，*uy* などのように，プロットの種類によって，4 つまでのエラーパラメータがあっても良い。フォーマット文字列で分けられるまで，任意の数の引数セットを指定することができる。

もし *y* が行列ならば，*x* およびエラーパラメータも同じ次数である行列でなければならない。*y* の列は，*x* の対応する列に対してプロットされ，エラーバーはエラーパラメータの対応する列から描かれる。

もし *fmt* が指定されていなければ，y 軸エラーバー (`"~"`) プロットスタイルを仮定する。もし *fmt* 引数が与えられるなら，通常のプロットとして解釈される（`__pltopt__`を参照)。さらに，以下のプロットスタイルがエラーバー付きで利用できる。

‘`~`’ yerrorbars をセットする。

‘`>`’ xerrorbars をセットする。

‘`~>`’ xyerrorbars をセットする。

‘`#`’ boxerrorbars をセットする。

‘`#~`’ boxerrorbars をセットする。

‘`#~>`’ boxxyerrorbars をセットする。

例:

```
errorbar(x, y, ex, ">")
```

x-*ex* から *x*+*ex* まで描かれる *x* エラーバーを付けて *x* の *y* に対する xerrorbar プロットする。

```
errorbar(x, y1, ey, "~", x, y2, ly, uy)
```

x の *y1* と *y2* に対する 2 つの yerrorbar プロットする。*y1* についてのエラーバーは，*y1*-*ey* から *y1*+*ey* までで描かれ，*y2* についての *y2*-*ly* から *y2*+*uy* までで描かれる。

```
errorbar(x, y, lx, ux, ly, uy, "~>")
```

x エラーバーは，*x*-*lx* から *x*+*ux* までで描かれ，*y* エラーバーは，*y*-*ly* から *y*+*uy* までで描き，*x* の *y* に対する xyerrorbar プロットする。

`loglogerr (args)` [Function File]

この関数は，エラーバー付きで両軸が対数軸における 2 次元プロットをする。引数のさまざまな組み合わせが可能である。もっとも利用される使い方は，以下のようなものである。

```
loglogerr (x, y, ey, fmt)
```

これは，*ey* で定義される *y* スケールの誤差と，*fmt* によって定義されるプロットフォーマットを用いて，*x* の *y* に対する両対数軸プロットをつくる。利用可能なフォーマットに対するエラーバーと追加の情報を参照せよ。

`semilogxerr (args)` [Function File]

この関数は，エラーバー付きで両軸が対数軸における 2 次元プロットをする。引数のさまざまな組み合わせが可能である。もっとも利用される使い方は，以下のようなものである。

```
semilogxerr (x, y, ey, fmt)
```

これは，*ey* で定義される *y* スケールの誤差と，*fmt* によって定義されるプロットフォーマットを用いて，*x* の *y* に対する両対数軸プロットをつくる。利用可能なフォーマットに対するエラーバーと追加の情報を参照せよ。

`semilogyerr (args)` [Function File]

この関数は，エラーバー付きで両軸が対数軸における 2 次元プロットをする。引数のさまざまな組み合わせが可能である。もっとも利用される使い方は，以下のようなものである。

```
semilogyerr (x, y, ey, fmt)
```

これは，*ey* で定義される *y* スケールの誤差と，*fmt* によって定義されるプロットフォーマットを用いて，*x* の *y* に対する両対数軸プロットをつくる。利用可能なフォーマットに対するエラーバーと追加の情報を参照せよ。

17.3 3 次元プロット

MATLAB スタイルの 3 次元プロットコマンドは，以下のようなものがある。

`mesh (x, y, z)` [Function File]

meshdomから行列 *x* と y，そしてメッシュの *x* と *y* 座標に対応する行列 *z* を与えて，メッシュをプロットする。もし *x* と *y* がベクトルならば，典型的な頂点は，(x(j), *y*(i), *z*(i,j)) となる。従って，*z* の列は，さまざまな *x* 値に対応し，*z* の行はさまざまな *y* 値に対応する。

`[xx, yy] = meshgrid (x, y)` [Function File]

`[xx, yy] = meshgrid (x)` [Function File]

x および *y* 座標のベクトルを与え，メッシュの *x* および *y* 座標に対応する 2 つの行列を返す。*xx* の行は *x* のコピーであり，*yy* の列は *y* のコピーである。

`meshdom (x, y)` [Function File]

x および *y* 座標のベクトルを与え，メッシュの *x* および *y* 座標に対応する 2 つの行列を返す。

注意: この関数は MATLAB の旧バージョンとの互換性のために提供されているものです。かわりに meshgridを使用すべきである。

17.4 注釈のプロット

`grid (arg)` [Function File]

2 次元プロットについて，プロットにグリッドを表示させる。引数は，`"on"`または`"off"`のどちらかをとる。もしこれを省略すると，`"on"`であると仮定する。

`title (string)` [Function File]

プロットのタイトルを指定する。

`xlabel` (*string*) [Function File]
`ylabel` (*string*) [Function File]
`zlabel` (*string*) [Function File]
プロットの x 軸，y 軸および z 軸ラベルを指定する。もしすでにプロットを表示しているならば，新たなラベル付きのプロットを表示するために，`replot` コマンドを使用する。

`top_title` (*string*) [Function File]
`bottom_title` (*string*) [Function File]
プロットの上部（下部）に *string* なるテキストをタイトルとする。

17.5 1 ページ上の複数プロット

以下の関数は全て，マルチプロット機能をサポートする gnuplotのバージョンを必要としています。

`mplot` (*x*, *y*) [Function File]
`mplot` (*x*, *y*, *fmt*) [Function File]
`mplot` (*x1*, *y1*, *x2*, *y2*) [Function File]
これは，1 ページに複数のグラフをプロットできるような gnuplotのマルチプロット版で動作するような plot関数の修正版である。このプロット版は，各々の引数セットを処理した後で，次のサブプロット位置に自動的に進める。

さまざまなオプションについて，*plot* 関数の解説を参照せよ。

`multiplot` (*xn*, *yn*) [Function File]
マルチプロットモードをセットしたりリセットする。

もし引数がゼロでないならば，`multiplot`は *x* および *y* 軸に沿って *xn* および *yn* サブプロットでマルチプロットモードをセットアップする。もし両方のモードがゼロならば，`multiplot` はマルチプロットモードをクローズする。

`oneplot` () [Function File]
もしマルチプロットモードに入っていれば，シングルプロットモードに切り替わる。

`plot_border` (...) [Function File]
境界を示す側を指定するために，複数の引数を指定できる。許容される引数は，以下のようなものである。

`"blank"` 境界を表示しない。

`"all"` 全ての境界を表示する。All borders displayed

`"north"` 上側の境界を表示する。

`"south"` 下側の境界を表示する。

`"east"` 右側の境界を表示する。

`"west"` 左側の境界を表示する。

この引数は，1 文字に短縮してもよい。何も引数を付けないときは，`plot_border`は境界をオフにする。

`subplot (rows, cols, index)` [Function File]
`subplot (rcn)` [Function File]

gnuplotをマルチプロットモードにして，インデックスによって与えられる位置にプロットする（*cols* × *rows* のサブウインドウが存在する）。

subplotを呼び出す前に，コマンド`__gnuplot_set__ size xsize, ysize` が使用されるときに，グローバル変数*__multiplot_scale__*が使用される。

*__multiplot_scale__*の値は，2 つの要素を持つベクトルとなるべきであり，最初の要素は *xsize* に等しく，2 番めは *ysize* に等しい値をセットする。

入力：

rows サブプロット枠における行の数である。

columns サブプロット枠における列の数である。

index 次のプロットを行うサブプロットのインデックスである。

もし 1 個の引数を与えるならば，3 桁の値でなければならない。その値は，1 桁目に行数，2 桁目に列数，3 桁目にプロットインデックスを指定する。

プロットインデックスは，行方向に並ぶ。ある行における全ての列を最初に埋められ，続いて次の行が埋められる。

たとえば，4 行 2 列のマス目をもつプロットは，以下のように並ぶプロットインデックスをもつ。

1	2	3	4
5	6	7	8

`subwindow (xn, yn)` [Function File]

マルチプロットモードにおいて，サブウインドウの位置を次の枠にセットする。マルチプロットモードは，この実行前に`multiplot`関数を使用して初期化されている必要がある。一方，このコマンドは `multiplot` への短縮名である。

17.6 マルチプロットウインドウ

`figure (n)` [Function File]

現在のプロットウインドウを，ウインドウ *n* にセットする。現在のところ，この関数は，X11 と，マルチフレームをサポートする gnuplot のバージョンを必要とする。もし *n* が指定されなければ，次に利用可能なウインドウ番号が選ばれる。

17.7 低水準プロットコマンド

`gplot ranges expression using title style` [Command]

2 次元プロットを生成する。

ranges，*using*，*title* および *style* 引数はオプションであり，*using*，*title* および *style* の量指定子は，その式の後に，任意のオーダで指定することができる。カンマで分けることによって，1 個のコマンドで複数の式をプロットすることができる。各々の式が，それぞれの量指定子をもつことができる。

オプション項目 *ranges* は，以下の記述方法をもつ。

```
[ x_lo : x_up ] [ y_lo : y_up ]
```

また，この記述方法は，実際のデータ範囲とは独立に，プロットの軸の範囲を指定するために使用できる。y 軸の範囲および個々の限界値のいずれかは省略することができる。範囲 `[:]`は，使用すべき標準の限界値を示すものである。これは，普通，全てのデータ点を含むくらい充分に大きな範囲が使用される。

プロットされるべき式は，何らかの定数行列（たとえば `[ 1, 2; 3, 4 ]`）を含んではいけない。なぜならば，データ行列からのプロット範囲を識別することはほぼ不可能であるからである。

オプション項目に対する記述法の解説については，gnuplotのヘルプを参照のこと。

初期設定により，gplotコマンドは行列の 1 列めと 2 列めに対してプロットする。もしその行列が 1 列だけしか含まないならば，y 座標のベクトルとして受け入れられ，x 軸はゼロから開始する要素のインデックスとなる。たとえば，

```
gplot rand (100,1) with linespoints
```

このコマンドは，100 個の乱数値とそれらを繋ぐ直線をプロットする。gplotが列ベクトルをプロットするために使用されるときには，その要素のインデックスは x 値として受け入れられる。

もし 2 つ以上の列が存在するならば，*using* 量指定子を用いて，プロットすべき列を選択することができる。たとえば，以下のデータが与えられているとする。

```
x = (-10:0.1:10)';
data = [x, sin(x), cos(x)];
```

このとき，

```
gplot [-11:11] [-1.1:1.1] \
  data with lines, data using 1:3 with impulses
```

このコマンドは，2 つのラインをプロットする。最初のラインは，`data with lines`コマンドによって生成されており，−10 から 10 までの範囲にわたる正弦関数のグラフである。データは，その行列の最初の 2 列から受け入れられる。なぜならば，プロットすべき列は，*using* 量指定子では指定できないからである。

このプロットの 2 番めの部分における項目`using 1:3`は，行列 dataの 1 番めと 3 番めの列がプロットすべき値として受け取るということを指定する。

この例において，その範囲は，実際のデータ範囲よりも少し大きめに明示的に指定されていた。結果として，その曲線はプロットの境界に触れることはない。

gsplot *ranges expression using title style* [Command]

3 次元プロットを生成する。

ranges，*using*，*title* および *style* 引数はオプションであり，*using*，*title* および *style* の量指定子は，その式の後に，任意のオーダで指定することができる。カンマで分けることによって，1 個のコマンドで複数の式をプロットすることができる。各々の式が，それぞれの量指定子をもつことができる。

オプション項目 *ranges* は，以下の記述方法をもつ。

```
[ x_lo : x_up ] [ y_lo : y_up ] [ z_lo : z_up ]
```

また，この記述方法は，実際のデータ範囲とは独立に，プロットの軸の範囲を指定するために使用できる。y 軸と z 軸の範囲および個々の限界値のいずれかは省略することができる。範囲`[:]`は，使用すべき標準の限界値を示すものである。これは，普通，全てのデータ点を含むくらい充分に大きな範囲が使用される。

プロットされるべき式は，何らかの定数行列（たとえば `[ 1, 2; 3, 4 ]`）を含んではいけない。なぜならば，データ行列からのプロット範囲を識別することはほぼ不可能であるからである。

オプション項目に対する記述法の解説については，gnuplotのヘルプを参照のこと。

初期設定により，gsplotコマンドは，行インデクスをx値として，列インデクスをy値として用い，式の各列の値をz値としてプロットする。このインデクスは，1ではなく0からカウントされる。たとえば，

```
gsplot rand (5, 2)
```

このコマンドは，その行列の行および列から受け入れたxおよびy値を用いて，ランダムな表面をプロットする。

パラメトリックプロットモードがセットされていれば（*gset parametric* なるコマンドを実行すれば），gsplotは3つの領域におけるラインを定義するx，yおよびzの値として，同時に3つの行列の列を受け取る。そのほかの余分な列は無視され，xとyの値はソートされていると期待される。たとえば，parametricをセットすると，以下のような行列をプロットするという意味になる。

$$\begin{bmatrix} 1 & 1 & 3 & 2 & 1 & 6 & 3 & 1 & 9 \\ 1 & 2 & 2 & 2 & 2 & 5 & 3 & 2 & 8 \\ 1 & 3 & 1 & 2 & 3 & 4 & 3 & 3 & 7 \end{bmatrix}$$

このとき rand (5, 30)ではない。

gset *options* [Command]
gshow *options* [Command]
replot *options* [Command]

基本的なプロットコマンドに加えて，gnuplotからのコマンド gsetと gshowの全範囲は，replotと同じように利用可能である。

Octave 2.0では，setと showというコマンド名は，gsetと gshowコマンドに変更された。これは，Octaveの将来のバージョンにおいて，MATLABのグラフィックとGUIコマンドと互換性を待たせるようにするためである（現在のところ，以前の setと showコマンドは動作するが，これらのコマンドは，gsetと gshowコマンドを使うように勧める邪魔な警告メッセージを表示する）。

gsetと gshowコマンドは，gnuplotのパラメータをセットしたり表示したりできるようにする。gsetと gshowコマンドに関するさらなる情報については，gnuplotユーザーズガイドにある set と showに関する解説を参照せよ（Octaveから実行する代わりに，gnuplotを直接実行しているならば，オンラインでも入手できる）。

replotコマンドは，プロットを強制的に再描画する。これは，プロットについて，タイトルまたは軸ラベルなど何らかの変更を行ったときに役立つ。replotコマンドは，gplotや gsplotと同じコマンドを受け入れる（データ範囲を除く）。従って，存在するプロットに追加のラインを加えることができる。

たとえば，

```
gset term tek40
gset output "/dev/plotter"
gset title "sine with lines and cosine with impulses"
replot "sin (x) w l"
```

これらのコマンドは，プロットを行う端末のタイプを変更し，現在のプロットにタイトルを追加し，以下のグラフを追加する。$\sin(x)$ そして，プロットするデバイスに，新たなプロットを送りつける。この最終ステップは，通常，プロットを更新するために必要である。この標準設定は，反応の遅い端末，あるいはハードコピー出力機器にとっては合理的である。なぜならば，

再プロットコマンドで追加のラインを加えるときでさえも，gnuplot は常にプロット全体を再描画するので，軸ラベルの変更程度の少しの更新のたびに，毎回，生成された完全に新しいプロットが欲しいとは思わないはずだからである。

コマンド `shg`は，何も引数を付けない `replot`コマンドを実行することに等しい。

gnuplot を呼び出す前に，プロットデータ中の NaN 値は自動的に省略され，Inf 値は非常に大きな値へと変換されます。

17.8 gnuplotとの連携

`gnuplot_binary` [Built-in Variable]

プロットコマンドによって起動されるプログラムの名前である。初期値は，`"gnuplot"`である。
付記 C [Installation], 頁 313 を参照せよ。

`gnuplot_has_frames` [Built-in Variable]

もしこの変数の値がゼロでないならば，Ovtave は，使用している gnuplot が複数フレームをサポートしていると仮定する（この機能は最近の 3.6beta リリースに含められている）。その初期値は configure によって決定されるが，この設定が間違っている場合，あるいは gnuplot のインストールを更新したときには，スタートアップスクリプトまたはコマンドラインにて変更することができる。

`gnuplot_command_plot` [Built-in Variable]

`gnuplot_command_replot` [Built-in Variable]

`gnuplot_command_splot` [Built-in Variable]

`gnuplot_command_using` [Built-in Variable]

`gnuplot_command_with` [Built-in Variable]

`gnuplot_command_axes` [Built-in Variable]

`gnuplot_command_title` [Built-in Variable]

`gnuplot_command_end` [Built-in Variable]

18 行列の操作

行列の要素が，ある条件に合うかどうかを確かめるチェックをしたり，行列の要素を並び替えるための多くの関数が利用可能です。たとえば，Octave では，ある行列の全要素が有限値がどうか，あるいは特定の値よりも小さいかどうかを簡単に知ることができます。Octave では，一部の要素を回転したり，上三角るいは下三角部分を抜き出したり，行列の列をソートしたりもできます。

18.1 要素の検出と条件のチェック

anyと allfunction関数は，行列の要素の「いずれか」あるいは「すべて」がある条件を満たすかどうかを判定するときに便利です。findfunction関数も，行列の要素が特定の条件に合っていることを判定するときに便利です。

any (*x*, *dim*) [Built-in Function]

ベクトルの引数については，そのベクトルのどれかの要素がゼロでないときに 1 を返す。

行列の引数については，各要素が 1 と 0 を含む行ベクトルを返す。その要素は，対応する行列の列の要素のいずれかがゼロでないかどうかを示している。以下に例を示す。

```
any (eye (2, 4))
     ⇒ [ 1, 1, 0, 0 ]
```

オプション引数 *dim* を与えると，次元 *dim* にそって動作する。たとえば，以下のようになる。

```
any (eye (2, 4), 2)
     ⇒ [ 1; 1 ]
```

all (*x*, *dim*) [Built-in Function]

関数 allは，anyfunction関数と似た挙動をするが，ベクトルの全ての要素，あるいは行列の *dim* 次元方向の全ての要素がゼロでないときにのみ，真を返す点が異なっている。

比較演算子（Section 10.4 [Comparison Ops], 頁 66 を参照）は，1 と 0 のみを含む行列を返すので，その要素がゼロでないかどうかといった簡単なことではなく，多くのことに関して行列をテストすることは容易です。たとえば，以下の式は，

```
all (all (rand (5) < 0.9))
     ⇒ 0
```

ランダムな 5 行 5 列の行列について，その要素の全てが 0.9 よりも小さいかどうかを確認するためにテストします。

条件文（ifや whileステートメントのテスト部分）では，Octave は，all (all (condition))と打ち込んだかのようにテストを扱います。

xor (*x*, *y*) [Mapping Function]

x と *y* の全体の「排他的論理和」を返す。*x* と *y* のブール表現では， *x* または *y* が真であり，*x* かつ *y* が真でないときに限り，真である。

is_duplicate_entry (*x*) [Function File]

もし *x* の要素が，別の要素と重複しているならば，ゼロでない値を返す。

`diff (x, k, dim)` [Function File]

もし x が長さ n のベクトルならば，`diff (x)`は 1 回目の差分（first difference）のベクトルである: $x_2 - x_1, \ldots, x_n - x_{n-1}$.

もし x が行列ならば，`diff (x)`は first non-singleton dimension に沿った列の差の行列となる。

2 番めの引数はオプションである。もしこれを与えるならば，`diff (x,k)`となり，ここで k は非負の整数である。この関数は，k 回目の差分（k-th differences）を返す。k が行列の最初の non-singleton dimension よりも大きいことも可能である。このケースにおいて，`diff`は次の non-singleton dimension に沿って差をとることを継続する。

差をとるべき次元は，オプション変数 *dim* によって明示的に指定することができる。この場合，k 回目の差分は，この次元に沿って計算される。k が `size (x, dim)`を越える場合は，空行列が返される。

`isinf (x)` [Mapping Function]

x 内の無限大（Inf）要素について 1 を，それ以外について 0 を返す。以下に例を示す。

```
isinf ([13, Inf, NA, NaN])
     ⇒ [ 0, 1, 0, 0 ]
```

`isnan (x)` [Mapping Function]

x の要素が NaN であれば 1，それ以外は 0 を返す。以下に例を示す。

```
isnan ([13, Inf, NA, NaN])
     ⇒ [ 0, 0, 0, 1 ]
```

`finite (x)` [Mapping Function]

有限値である x の要素について 1 を返し，そうでなければ 0 を返す。以下に例を示す。

```
finite ([13, Inf, NA, NaN])
     ⇒ [ 1, 0, 0, 0 ]
```

`find (x)` [Loadable Function]

ある行列のゼロでない要素のインデックスのベクトルを返す。行列の各要素について 1 個のインデックスを得るために，Octave は，行列の列が（Fortran 配列が格納されているように）1 個の長いベクトルであると偽装する。たとえば，以下のようになる。

```
find (eye (2))
     ⇒ [ 1; 4 ]
```

もし 2 つの出力が要求されたならば，`find`は，行列のゼロでない要素の行と列のインデックスを返す。以下に例を挙げる。

```
[i, j] = find (2 * eye (2))
     ⇒ i = [ 1; 2 ]
     ⇒ j = [ 1; 2 ]
```

もし 3 つの出力が要求されたならば，`find`は，ゼロでない値を含むベクトルも返す。

```
[i, j, v] = find (3 * eye (2))
     ⇒ i = [ 1; 2 ]
     ⇒ j = [ 1; 2 ]
     ⇒ v = [ 3; 3 ]
```

`[err, y1, ...] = common_size (x1, ...)` [Function File]

全ての入力引数がスカラ，あるいは共通のサイズかどうかを判別する。もしそうであれば *err* はゼロであり，*yi* は，入力引数がスカラならば全要素が *xi* に等しい共通サイズの行列となり，そうでなければ *xi* である。もし入力が共通のサイズでなければ，エラーコードは 1 であり，*yi* は *xi* となる。以下に例を挙げる。

```
[errorcode, a, b] = common_size ([1 2; 3 4], 5)
⇒ errorcode = 0
⇒ a = [ 1, 2; 3, 4 ]
⇒ b = [ 5, 5; 5, 5 ]
```

これは，引数がスカラあるいは共通サイズをとるような関数の実装に役立つ。

18.2 行列の整形

`fliplr (x)` [Function File]

列の順序を逆にした *x* のコピーを返す。以下に例を挙げる。

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

`fliplr`は 2 次元配列についてのみ動作する。N 次元の配列を反転するには，このかわりに`flipdim`を使用せよ。

`flipud (x)` [Function File]

行の順序を逆にした *x* のコピーを返す。以下に例を挙げる。

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

行列を反転するための軸を定義することが困難なため，`flipud`は 2 次元配列に限って動作する。N 次元の配列を反転するには，このかわりに`flipdim`を使用せよ。

`flipdim (x, dim)` [Function File]

Return a copy of *x* flipped about the dimension *dim*. For example

```
flipdim ([1, 2; 3, 4], 2)
⇒  2  1
        4  3
```

`rot90 (x, n)` [Function File]

x の要素を反時計回りに 90 度回転させた行列を返す。2 番めの引数は，オプションであり，90 度回転を何回行うかを指定する（初期値は 1 である）。*n* に負の値を指定すると，行列を時計回りに回転させる。たとえば，

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

この式は，与えられた行列を時計回りに 90 度回転させる。以下の式は，すべて等価なステートメントである。

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

行列を回転するための軸を定義することが困難なため，rot90は 2 次元配列に限って動作する。N 次元の配列を反転するには，このかわりに rotdimを使用せよ。

rotdim (*x*, *n*, *plane*) [Function File]

x の要素を反時計回りに 90 度回転させた行列を返す。2 番めの引数は，オプションであり，90 度回転を何回行うかを指定する（初期値は 1 である）。3 番めの引数もオプションであり，回転の次元を定義する。*plane* は，その行列の 2 つの異なる妥当な次元を含む，2 要素のベクトルである。もし *plane* が与えられなければ，最初の 2 つの non-singleton dimensions が使用される。

n に負の値を指定すると，行列を時計回りに回転させる。たとえば，

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

この式は，与えられた行列を時計回りに 90 度回転させる。以下の式は，すべて等価なステートメントである。

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

cat (*dim*, *array1*, *array2*, . . . , *arrayN*) [Built-in Function]

N 個の配列オブジェクト，*array1*，*array2*，. . . ，*arrayN* を次元 *dim* に沿って連結したものを返す。

```
A = ones (2, 2);
B = zeros (2, 2);
cat (2, A, B)
⇒ ans =

     1 1 0 0
     1 1 0 0
```

これとは別に，以下のように 2 番めの次元に沿って A と B を連結することができる。

```
[A, B]
```

dim は N 個の配列オブジェクトの次元より大きくても良く，その結果は，以下の例が示すように，*dim* 次元となる。

```
cat (4, ones(2, 2), zeros (2, 2))
⇒ ans =

   ans(:,:,1,1) =

     1 1
     1 1

   ans(:,:,1,2) =
     0 0
     0 0
```

horzcat (*array1*, *array2*, . . . , *arrayN*) [Built-in Function]

N 個の配列オブジェクト，*array1*，*array2*，. . . ，*arrayN* を次元 2 に沿って水平方向へ連結したものを返す。

vertcat (*array1*, *array2*, . . . , *arrayN*) [Built-in Function]

N 個の配列オブジェクト，*array1*，*array2*，. . . ，*arrayN* を次元 1 に沿って垂直方向へ連結したものを返す。

permute (a, *perm*) [Built-in Function]

N 次元の配列オブジェクト *a* に対する generalized transpose を返す。置換ベクトル *perm* は，1:ndims(a)の要素を含んでいなければならない（どのような順でも良いが，各要素は 1 度だけ現れなければならない）。

ipermute (a, *iperm*) [Built-in Function]

permuteの逆関数である。以下の式，

```
ipermute (permute (a, perm), perm)
```

は，元の配列 *a* を返す。

reshape (a, *m*, *n*, . . .) [Function File]
reshape (a, *siz*) [Function File]

行列 *a* の各要素を組み替え，与えられた次元となる行列を返す。その行列の要素は，行要素が連続であるもの（column-major；Fortran の配列が格納されているような順序）としてアクセスされる。

以下に例を挙げる。

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

もとの行列における全要素数は，新しい行列の総要素数と一致していなければならない。

返される行列が 1 つの次元しか持たないことは起こり得ず，引数が空であるという警告がなされる。

y = circshift (*x*, *n*) [Function File]

配列 *x* を循環シフトする。*n* は，*x* の次元数よりも大きくない整数のベクトルでなければならない。*n* の値は，正にも負にもなる。これは，ある値または *x* がシフトされる方向を決定する。もし *n* の要素がゼロならば，対応する *x* の次元はシフトされない。たとえば，以下のようになる。

```
x = [1, 2, 3; 4, 5, 6, 7, 8, 9];
circshift (x, 1)
⇒  7, 8, 9
    1, 2, 3
    4, 5, 6
circshift (x, -2)
⇒  7, 8, 9
    1, 2, 3
    4, 5, 6
circshift (x, [0,1])
⇒  3, 1, 2
    6, 4, 5
    9, 7, 8
```

y = shiftdim (*x*, *n*) [Function File]

[*y*, *ns*] = shiftdim (*x*) [Function File]

x の次元を，*n* だけずらす（シフトする）。ここで *n* は整数のスカラでなければならない。*n* が正のとき，*x* の次元は左にシフトされ，前方の次元は末尾側に循環する。もし *n* が負ならば,, *x* の次元は右にシフトされ，*n* 個の先立つ singleton dimensions を加える。

たとえば，以下のようになる。

```
x = ones (1, 2, 3);
size (shiftdim (x, -1))
⇒ [2, 3, 1]
size (shiftdim (x, 1))
⇒ [1, 1, 2, 3]
[b, ns] = shiftdim (x);
⇒ b =  [1, 1, 1; 1, 1, 1]
⇒ ns = 1
```

shift (*x*, *b*) [Function File]

shift (*x*, *b*, *dim*) [Function File]

もし *x* がベクトルならば，*x* 要素の長さ *b* だけ循環シフトする。

もし *x* が行列ならば，*x* の各列に対して同じことをする。もしオプション引数 *dim* を与えると，この次元に沿って処理を行う。

[*s*, *i*] = sort (*x*) [Loadable Function]

[*s*, *i*] = sort (*x*, *dim*) [Loadable Function]

[*s*, *i*] = sort (*x*, *mode*) [Loadable Function]

[*s*, *i*] = sort (*x*, *dim*, *mode*) [Loadable Function]

x の要素を昇順で並び替えたコピーを返す。行列に対しては，sortは各列について要素を並びかえす。

以下に例を挙げる。

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

sort関数は，ソートされた行列における要素，元の行インデックスを含む行列を作るためにも使用できるだろう。たとえば，以下のようである。

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

オプション引数 *dim* を与えると，その行列は *dim* によって定義された次元に沿ってソートされる。オプション引数 modeは，値がソートされる並び順を定義する。modeがとりうる値は，'ascend' あるいは 'descend' である。

等しい要素について，このインデックスは，等しい要素が元のリストに現れる順序となる。

sort関数は，文字列や文字列のセル配列をソートするために使用することもできる。その場合は，文字列の辞書順となる。

sortにおいて使用されているアルゴリズムは，部分的に並び替えられたリストをソートすることに最適化されている。

sort関数は，指定すべきソートキーを受け入れないので，1 回の呼び出しでは，さまざまな列にある要素の値によって，ある行列の行を並び替えることはできません。[1] しかし，2 番めの出力を使用することにより，与えられた列における値に基づいて全行をソートすることが可能です。以下に，2 列めにある値に基づいて，行列の行をソートする例を示します。

```
a = [1, 2; 2, 3; 3, 1];
[s, i] = sort (a (:, 2));
a (i, :)
     ⇒  3  1
        1  2
        2  3
```

tril (a, *k*) [Function File]
triu (a, *k*) [Function File]

行列 *a* の下三角（tril）あるいは上三角（triu）部分を抜き出し，その他の全ての要素に 0 をセットすることで作られる新たな行列を返す。2 番めの引数はオプションであり，主対角のどのくらい上の対角または下の対角にもゼロするかを指定する。

k の初期値はゼロであり，結果として triuと trilは，通常は，返される行列の一部に主対角を含む。

もし *k* が負であれば，主対角の上側（tril）または下側（triu）の要素も選択する。

k の絶対値は，下側対角または上側対角の数よりも大きくなければならない。

[1] たとえば，最初に 1 列めの値に基づいてソートし，次に，この並び順を保ったまま，2 列めにある値に基づいてソートすることなどです。

たとえば,

```
tril (ones (3), -1)
⇒  0  0  0
         1  0  0
         1  1  0
```

および

```
tril (ones (3), 1)
⇒  1  1  0
         1  1  1
         1  1  1
```

となる。

vec (*x*) [Function File]

行列 *x* の列を，1 列に積み上げたベクトルにして返す。

統計学ならびに経済学における行列の微分についての応用は，Magnus and Neudecker (1988) を参照せよ。

vech (*x*) [Function File]

正方行列 *x* の右上三角の要素を削除し，残りの要素を 1 列に積み上げたベクトルにして返す。

統計学ならびに経済学における行列の微分についての応用は，Magnus and Neudecker (1988) を参照せよ。

prepad (*x*, *l*, *c*) [Function File]
postpad (*x*, *l*, *c*) [Function File]
postpad (*x*, *l*, *c*, *dim*) [Function File]

スカラ *c* をベクトル *x* の長さが *l* になるまで，その先頭（末尾）に追加する。もし 3 番めの引数が与えられなければ，0 という値が使用される。

もし `length (x) > l` ならば，*x* の始まり（終わり）からの要素は，長さ *l* のベクトルが得られるまで削除される。

もし *x* が行列ならば，要素は各行に追加されたり取り除かれたりする。

もしオプション引数 *dim* が与えられるならば，この次元に沿って操作を行う。

18.3 特殊なユーティリティ行列

eye (*x*) [Built-in Function]
eye (*n*, *m*) [Built-in Function]
eye (..., *class*) [Built-in Function]

単位行列を返す。1 個のスカラ引数を付けて呼び出すと，eye関数は指定した次元の正方行列を返す。もし 2 個のスカラを与えると，それらを行数と列数として受け入れる。2 個の要素を持つベクトルを与えると，その要素の値をそれぞれ行数および列数として受け入れる。たとえば，以下のようになる。

```
eye (3)
     ⇒  1  0  0
         0  1  0
         0  0  1
```

以下の式は，全て同じ結果になる。

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

オプション引数 *class* は，指定したタイプの配列を返すようにする。

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

MATLAB との互換性について，引数をつけずに eyeを呼び出すことは，1 という引数を付けて呼び出すことに等しい。

`ones (`*x*`)` [Built-in Function]
`ones (`*n*`,` *m*`)` [Built-in Function]
`ones (`*n*`,` *m*`,` *k*`, ...)` [Built-in Function]
`ones (..., class)` [Built-in Function]

その要素が全て 1 であるような行列または N 次元の配列を返す。その引数は，eyeの引数とと同じように扱われる。

全要素が全て同じ値をもつ行列を作るには，以下のような式を使うのがよい。

```
val_matrix = val * ones (n, m)
```

オプション引数 *class* は，指定したタイプの配列を返すようにする。

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

`zeros (`*x*`)` [Built-in Function]
`zeros (`*n*`,` *m*`)` [Built-in Function]
`zeros (`*n*`,` *m*`,` *k*`, ...)` [Built-in Function]
`zeros (..., class)` [Built-in Function]

その要素が全て 0 であるような行列または N 次元の配列を返す。その引数は，eyeの引数とと同じように扱われる。

オプション引数 *class* は，指定したタイプの配列を返すようにする。

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

`repmat (`*A*`,` *m*`,` *n*`)` [Function File]
`repmat (`*A*`, [`*m n*`])` [Function File]

行列 A を，行方向に m 個，列方向に n 個に並べたブロック行列を作る。もし n が指定されなければ，$m \times m$ のブロック行列を作る。

`rand (`*x*`)` [Loadable Function]
`rand (`*n*`,` *m*`)` [Loadable Function]
`rand ("seed",` *x*`)` [Loadable Function]

範囲 (0, 1) に一様分布する乱数を要素に持つ行列を返す。その引数は，eyeに対する引数と同じように処理される。さらに，以下の式を使用することにより，乱数生成器に関するシードをセットすることができる。

```
rand ("seed", x)
```

ここで x はスカラの値である。もし，

```
rand ("seed")
```

のように呼び出すと，randは現在のシード値を返す。

randn (*x*) [Loadable Function]
randn (*n*, *m*) [Loadable Function]
randn ("seed", *x*) [Loadable Function]

正規分布する乱数を要素に持つ行列を返す。その引数は，eyeに対する引数と同じように処理される。さらに，以下の式を使用することにより，乱数生成器に関するシードをセットすることができる。

```
randn ("seed", x)
```

ここで *x* はスカラの値である。もし，

```
randn ("seed")
```

のように呼び出すと，randは現在のシード値を返す。

randおよびrandn関数は，別の生成器を使用しています。これは，以下のように確かめることができます。

```
rand ("seed", 13);
randn ("seed", 13);
u = rand (100, 1);
n = randn (100, 1);
```

および

```
rand ("seed", 13);
randn ("seed", 13);
u = zeros (100, 1);
n = zeros (100, 1);
for i = 1:100
  u(i) = rand ();
  n(i) = randn ();
end
```

は，同じ結果を返します。

通常，randおよびrandnは，その初期シードをシステムクロックから得ています。その結果，乱数列は Octave を実行するたびに同じではありません。もし正確に乱数列を再生成する必要が本当にあるならば，シードに特定の値をセットすればよいのです。

引数なしで呼び出すと，randとrandnは 1 個の乱数を返します。

randとrandn関数は，RANLIB からの Fortran コードを使用します。これは，Department of Biomathematics at The University of Texas, M.D. Anderson Cancer Center, Houston, TX 77030. の Barry W. Brown と James Lovato によってまとめられた，乱数値生成のための Fortran ルーチンのライブラリです。

randperm (*n*) [Function File]

1 から *n* までの整数をランダムに並べた行ベクトルを返す。

diag (*v*, *k*) [Built-in Function]

ベクトル *v* を対角 *k* に配置した対角行列を返す。2 番めの引数はオプションである。もしその引数が正であれば，そのベクトルは *k* 番めの上対角（super-diagonal）に配置する。もしこれ

が負の値ならば，k 番めの下対角（sub-diagonal）に配置する。k の標準設定は 0 であり，そのベクトルは対角に配置される。たとえば，以下のようになる。

```
diag ([1, 2, 3], 1)
     ⇒  0  1  0  0
        0  0  2  0
        0  0  0  3
        0  0  0  0
```

関数 `linspace`と `logspace`は，等間隔あるいは対数的に区切られた要素をもつベクトルを生成することを容易にします。Section 4.2 [Ranges], 頁 32 を参照してください。

`linspace (`*`base`*`, `*`limit`*`, `*`n`*`)` [Built-in Function]

base と *limit* の間を，等間隔で n 個に区切った要素をもつ行ベクトルを返す。要素数 n は，1 より大きくなければならない。*base* と *limit* は，その範囲に常に含まれる。もし *base* が *limit* よりも大きければ，その要素は降順に格納される。もし点の数が指定されなければ，100 という値を使用する。

`linspace`関数は，常に行ベクトルを返す。

`logspace (`*`base`*`, `*`limit`*`, `*`n`*`)` [Function File]

`linspace`と同様であるが，その値が 10^{base} から 10^{limit} まで対数的に区切られているところが異なっている。

もし *limit* が π に等しいならば，その点は 10^{base} and 10^{π} ではなく，10^{base} and π の範囲になる。これは MATLAB 関数に対応した互換性を保つためのものである。

`warn_neg_dim_as_zero` [Built-in Variable]

もし `warn_neg_dim_as_zero`の値がゼロでないならば，以下のような式に対して警告を表示する:

```
eye (-1)
```

初期値は 0 である。

`warn_imag_to_real` [Built-in Variable]

もし `warn_imag_to_real`がゼロでないならば，複素数を実数に暗黙的に変換するときに警告を表示する。初期値は 0 である。

18.4 有名な行列

The following functions return famous matrix forms.

`hankel (`*`c`*`, `*`r`*`)` [Function File]

Return the Hankel matrix constructed given the first column c, and (optionally) the last row r. If the last element of c is not the same as the first element of r, the last element of c is used. If the second argument is omitted, it is assumed to be a vector of zeros with the same size as c.

A Hankel matrix formed from an m-vector c, and an n-vector r, has the elements

$$H(i,j) = \begin{cases} c_{i+j-1}, & i+j-1 \leq m; \\ r_{i+j-m}, & \text{otherwise.} \end{cases}$$

`hilb (n)` [Function File]

Return the Hilbert matrix of order n. The i, j element of a Hilbert matrix is defined as

$$H(i,j) = \frac{1}{(i+j-1)}$$

`invhilb (n)` [Function File]

Return the inverse of a Hilbert matrix of order n. This can be computed computed exactly using

$$\begin{aligned} A_{ij} &= -1^{i+j}(i+j-1)\binom{n+i-1}{n-j}\binom{n+j-1}{n-i}\binom{i+j-2}{i-2}^2 \\ &= \frac{p(i)p(j)}{(i+j-1)} \end{aligned}$$

where

$$p(k) = -1^k \binom{k+n-1}{k-1}\binom{n}{k}$$

The validity of this formula can easily be checked by expanding the binomial coefficients in both formulas as factorials. It can be derived more directly via the theory of Cauchy matrices: see J. W. Demmel, Applied Numerical Linear Algebra, page 92.

Compare this with the numerical calculation of `inverse (hilb (n))`, which suffers from the ill-conditioning of the Hilbert matrix, and the finite precision of your computer's floating point arithmetic.

`sylvester_matrix (k)` [Function File]

Return the Sylvester matrix of order $n = 2^k$.

`toeplitz (c, r)` [Function File]

Return the Toeplitz matrix constructed given the first column c, and (optionally) the first row r. If the first element of c is not the same as the first element of r, the first element of c is used. If the second argument is omitted, the first row is taken to be the same as the first column.

A square Toeplitz matrix has the form:

$$\begin{bmatrix} c_0 & r_1 & r_2 & \cdots & r_n \\ c_1 & c_0 & r_1 & \cdots & r_{n-1} \\ c_2 & c_1 & c_0 & \cdots & r_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_n & c_{n-1} & c_{n-2} & \cdots & c_0 \end{bmatrix}$$

`vander (c)` [Function File]

Return the Vandermonde matrix whose next to last column is c.

A Vandermonde matrix has the form:

$$\begin{bmatrix} c_1^{n-1} & \cdots & c_1^2 & c_1 & 1 \\ c_2^{n-1} & \cdots & c_2^2 & c_2 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ c_n^{n-1} & \cdots & c_n^2 & c_n & 1 \end{bmatrix}$$

19 算術演算

以下の解説において特に注釈がない場合，この章で紹介するすべての関数は，実数型と複素数型のスカラおよび行列のいずれにも使用することが出来ます。

19.1 ユーティリティ関数

ここで紹介する関数群は，複素数に対しても使用することができます。各関数とも，1 つの引数を与えるように作られています。これらの関数群を，マッピング関数（*mapping function*）と呼びます。この名称は，引数として行列を与えたとき，行列の各要素に対して同じ処理をすることに由来するものです。

`ceil (x)` [Mapping Function]
: x よりも小さくない最小の整数を返す。x が複素数のときは，`ceil (real (x)) + ceil (imag (x)) * I`を返す。

`exp (x)` [Mapping Function]
: x の指数関数を計算する。行列に対して指数行列を計算するには，Chapter 20 [Linear Algebra], 頁 165 の章を参照せよ。

`fix (x)` [Mapping Function]
: x をゼロに向かって丸める。x が複素数のときは，`fix (real (x)) + fix (imag (x)) * I` を返す。

`floor (x)` [Mapping Function]
: x よりも大きくない最大の整数を返す。x が複素数のときは，`floor (real (x)) + floor (imag (x)) * I`を返す。

g `= gcd (`*a1*`, ...)` [Loadable Function]
`[`*g*`, `*v1*`, ...] = gcd (`*a1*`, ...)` [Loadable Function]
: 引数をひとつだけ与えると，その最大公約数を返す。2 つ以上の引数を指定するときは，それらが同じサイズのベクトルもしくはスカラでなければならない。この場合，対応する要素についてそれぞれ最大公約数を計算する。指定する各要素は，整数でなければならない。以下に例を示す。

```
gcd ([15, 20])
    ⇒  5
```

あるいは，以下のようにすることもできる。

```
gcd ([15, 9], [20 18])
    ⇒  5  9
```

v1 のように返り値を追加すると，以下のような整数のベクトルを返す。

$g = v_1 a_1 + v_2 a_2 + \cdots$

以前の Octave における後方互換性を確保するため，全ての引数がスカラであるときには，単一の返り値 *v1* にすべての値（*v1*, ...）を含めるようになっている。

`lcm (x, ...)` [Mapping Function]

x あるいは指定した全引数について，その最小公倍数を計算する。たとえば，

```
lcm (a1, ..., ak)
```

と

```
lcm ([a1, ..., ak]).
```

は同じ意味である。全ての要素は同じ大きさであるか，スカラでなければならない。

`log (x)` [Mapping Function]

x の各要素について，自然対数を計算する。対数行列を計算するには，Chapter 20 [Linear Algebra], 頁 165 の項を参照のこと。

`log10 (x)` [Mapping Function]

x の常用対数（10 を底とする対数）を計算する。

`log2 (x)` [Mapping Function]

`[f, e] = log2 (x)` [Mapping Function]

x について，2 を底とする対数を計算する。返り値を 2 つ指定すると，$1/2 <= |f| < 1$ および $x = f \cdot 2^e$

を満たす f と e を返す。

`max (x, y, dim)` [Mapping Function]

`[w, iw] = max (x)` [Mapping Function]

引数としてベクトルを渡すと，その要素の最大値を返す。行列を渡すと，各列ごとに最大値を返すので，結果は行ベクトルとなる。*dim* を指定すると，その次数を指定することができる。2 つの行列（あるいは行列とスカラ）を渡すと，対で比較した結果を返す。以下に例を挙げる。

```
max (max (x))
```

この結果は，x に含まれる要素の最大値を返す。

```
max (2:5, pi)
    ⇒  3.1416  3.1416  4.0000  5.0000
```

これは，`2:5`の範囲にある各要素と `pi`とを比較し，最大値を含む行ベクトルを返す。

引数に複素数を渡すと，要素の大きさで比較を行う。

1 つの入力に対して 2 つの返り値を受け取るとき，`max`関数は，最大値に対応する要素の位置も返す。以下の例を参照されたい。

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

`min (x, y, dim)` [Mapping Function]

`[w, iw] = min (x)` [Mapping Function]

引数としてベクトルを渡すと，その要素の最小値を返す。行列を渡すと，各列ごとに最小値を返すので，結果は行ベクトルとなる。*dim* を指定すると，その次数を指定することができる。2 つの行列（あるいは行列とスカラ）を渡すと，対で比較した結果を返す。以下に例を挙げる。

```
min (min (x))
```

この結果は，x に含まれる要素の最小値を返す。

```
min (2:5, pi)
    ⇒  2.0000  3.0000  3.1416  3.1416
```

これは，2:5の範囲にある各要素と piとを比較し，最小値を含む行ベクトルを返す。

引数に複素数を渡すと，要素の大きさで比較を行う。

1 つの入力に対して 2 つの返り値を受け取るとき，min関数は，最小値に対応する要素の位置も返す。以下の例を参照されたい。

```
[x, ix] = min ([1, 3, 0, 2, 5])
    ⇒  x = 0
       ix = 3
```

mod (*x*, *y*) [Mapping Function]

以下の計算式により，剰余（割り算の余り）を計算する。

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

負の値を正しく扱うことに注意されたい。つまり，mod (-1, 3)の結果は 2 であり，-1 ではない。rem (-1, 3)の結果は-1 となる。また，mod (x, 0)は，x を返す。

引数の次数が合わないとき，あるいは引数に複素数を指定したときはエラーメッセージを表示する。

nextpow2 (*x*) [Function File]

x がスカラであれば，$2^n \geq |x|$.

を満たす最初の整数 n を返す。

x がベクトルであれば，nextpow2 (length (x))を返す。

pow2 (*x*) [Mapping Function]

pow2 (*f*, *e*) [Mapping Function]

引数を 1 つだけ指定したときには，x の各要素について 2^x を計算する。2 つの引数に対しては，$f \cdot 2^e$. を計算する。

rem (*x*, *y*) [Mapping Function]

以下の式によって，x / yの余りを計算する。

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

引数の次数が合わないとき，あるいは複素数が渡されたときにはエラーメッセージを表示する。

round (*x*) [Mapping Function]

x に最も近い整数を返す。x が複素数のときには，round (real (x)) + round (imag (x)) * Iを返す。

sign (*x*) [Mapping Function]

符号関数（*signum* function）を計算する。この関数は，以下のように定義される。

$$\mathrm{sign}(x) = \begin{cases} 1, & x > 0; \\ 0, & x = 0; \\ -1, & x < 0. \end{cases}$$

複素数については，x ./ abs (x)を返す。

`sqrt (x)` [Mapping Function]
: x の正の平方根（ルート）を計算する。x が負のとき，複素数を返す。行列の平方根については，Chapter 20 [Linear Algebra], 頁 165 を参照せよ。

19.2 複素数演算

ここで紹介する関数群は，複素数に対しても使用することができます。各関数とも，1 つの引数を与えるように作られています。引数として行列を与えたとき，行列の各要素ごとに同じ処理をします。以降の解説において，z は複素数 $x + iy$ であり，i は虚数単位（$\sqrt{-1}$）です。

`abs (z)` [Mapping Function]
: z の大きさを，$|z| = \sqrt{x^2 + y^2}$. から計算する。

  以下に例を示す。

```
abs (3 + 4i)
     ⇒ 5
```

`conj (z)` [Mapping Function]
: z の共役複素数 $\bar{z} = x - iy$ を返す。

`imag (z)` [Mapping Function]
: z の虚部を実数で返す。

`real (z)` [Mapping Function]
: z の実部を返す。

19.3 三角関数

Octave では以下の三角関数を提供しています。引数はラジアンで指定します。角度（degree）をラジアンに変換するには，角度に $\pi/180$ を乗じてください（たとえば，`sin (30 * pi/180)`は，30 度に対する正弦の値を返す）。

`sin (x)` [Mapping Function]
: x の各要素について，正弦（サイン）を計算する。

`cos (x)` [Mapping Function]
: x の各要素について，余弦（コサイン）を計算する。

`tan (z)` [Mapping Function]
: x の各要素について，正接（タンジェント）を計算する。

`sec (x)` [Mapping Function]
: x の各要素について，余割（セカント）を計算する。

`csc (x)` [Mapping Function]
: x の各要素について，余割（コセカント）を計算する。

`cot (x)` [Mapping Function]
: x の各要素について，余接（コタンジェント）を計算する。

`asin (x)` [Mapping Function]
x の各要素について，逆正弦（アークサイン）を計算する。

`acos (x)` [Mapping Function]
x の各要素について，逆余弦（アークコサイン）を計算する。

`atan (x)` [Mapping Function]
x の各要素について，逆正接（アークタンジェント）を計算する。

`asec (x)` [Mapping Function]
x の各要素について，逆正割（アークセカント）を計算する。

`acsc (x)` [Mapping Function]
x の各要素について，逆余割（アークコセカント）を計算する。

`acot (x)` [Mapping Function]
x の各要素について，逆余接（アークコタンジェント）を計算する。

`sinh (x)` [Mapping Function]
x の各要素について，双曲線正弦（ハイパボリックサイン）を計算する。

`cosh (x)` [Mapping Function]
x の各要素について，双曲線余弦（ハイパボリックコサイン）を計算する。

`tanh (x)` [Mapping Function]
x の各要素について，双曲線正接（ハイパボリックタンジェント）を計算する。

`sech (x)` [Mapping Function]
x の各要素について，双曲線正割（ハイパボリックセカント）を計算する。

`csch (x)` [Mapping Function]
x の各要素について，双曲線余割（ハイパボリックコセカント）を計算する。

`coth (x)` [Mapping Function]
x の各要素について，逆双曲線余接（逆ハイパボリックコタンジェント）を計算する。

`asinh (x)` [Mapping Function]
x の各要素について，逆双曲線正弦（逆ハイパボリックサイン）を計算する。

`acosh (x)` [Mapping Function]
x の各要素について，逆双曲線正接（逆ハイパボリックコサイン）を計算する。

`atanh (x)` [Mapping Function]
x の各要素について，逆双曲線正接（逆ハイパボリックタンジェント）を計算する。

asech (*x*) [Mapping Function]
x の各要素について，逆双曲線正割（逆ハイパボリックセカント）を計算する。

acsch (*x*) [Mapping Function]
x の各要素について，逆双曲線余割（逆ハイパボリックコセカント）を計算する。

acoth (*x*) [Mapping Function]
x の各要素について，逆双曲線余接（逆ハイパボリックコタンジェント）を計算する。

以上の関数は，引数が 1 つであることを想定しています。引数に行列を与えたときは，以下に示すように，各要素ごとに計算を行います。

```
sin ([1, 2; 3, 4])
     ⇒  0.84147   0.90930
         0.14112  -0.75680
```

atan2 (*y*, *x*) [Mapping Function]
x と *y* の対応する要素について，*y* / *x* の逆正接（アークタンジェント）を計算する。結果を $-\pi$ から π の範囲で返す。

19.4 和と積

sum (*x*, *dim*) [Built-in Function]
次元 *dim* 方向の和を計算する。*dim* が省略されたときは，1 を指定した（列方向に和をとる）とみなす。

例外として，*x* がベクトルで *dim* を省略したとき，要素の和を返す。

prod (*x*, *dim*) [Built-in Function]
次元 *dim* 方向の積を計算する。*dim* が省略されたときは，1 を指定した（列方向に積をとる）とみなす。

例外として，*x* がベクトルで *dim* を省略したとき，*x* の要素の積を返す。

cumsum (*x*, *dim*) [Built-in Function]
次元 *dim* 方向の累積和を計算する。*dim* が省略されたときは，1 を指定した（列方向に累積和をとる）とみなす。

例外として，*x* がベクトルで *dim* を省略したとき，*x* の要素の累積和を返す。

cumprod (*x*, *dim*) [Built-in Function]
次元 *dim* 方向の累積積を計算する。*dim* が省略されたときは，1 を指定した（列方向に累積積をとる）とみなす。

例外として，*x* がベクトルで *dim* を省略したとき，*x* の要素の累積積を返す。

sumsq (*x*, *dim*) [Built-in Function]
次元 *dim* 方向の積和を計算する。*dim* が省略されたときは，1 を指定した（列方向に積和をとる）とみなす。

例外として，*x* がベクトルで *dim* を省略したとき，要素の積和を返す。

この関数は，概念的には以下の計算を行っているに等しい。

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

しかし，この関数の方が使用メモリが少なく，x が実数の場合に conjを呼び出さないので，より効率的である。

19.5 特殊な関数

[*j*, *ierr*] = besselj (*alpha*, *x*, *opt*) [Loadable Function]
[*y*, *ierr*] = bessely (*alpha*, *x*, *opt*) [Loadable Function]
[*i*, *ierr*] = besseli (*alpha*, *x*, *opt*) [Loadable Function]
[*k*, *ierr*] = besselk (*alpha*, *x*, *opt*) [Loadable Function]
[*h*, *ierr*] = besselh (*alpha*, *k*, *x*, *opt*) [Loadable Function]

さまざまな Bessel（あるいは Hankel）関数を計算する。

besselj　第 1 種の Bessel 関数

bessely　第 2 種の Bessel 関数

besseli　第 1 種の修正 Bessel 関数

besselk　第 2 種の修正 Bessel 関数

besselh　第 1 種（k=1）あるいは第 2 種（k=2）の Hankel 関数

引数 *opt* を与えたとき，$k = 1$ に対して exp (-I*x)，$k = 2$ に対して exp (I*x)によってスケール化した値を返す。

alpha がスカラであれば，結果は x と同じサイズになる。x がスカラであれば，結果は *alpha* と同じサイズになる。*alpha* が行ベクトルで x が列ベクトルであれば，返す結果は length (x) 行 length (alpha)列の行列となる。一方，*alpha* と x は整合性がなければならず，結果は同じサイズとなる。

alpha の値は実数でなければならない。x は複素数でもよい。

必要に応じて *ierr* を指定することにより，以下のような実行結果を得ることができる。これは結果と同じサイズになる。

0. 正常終了
1. 入力エラーにより NaNを返した
2. オーバーフローにより Infを返した
3. argument reduction による有効桁のロスにより，計算機精度の半分以下の精度となった
4. argument reduction によって有効桁が完全にロスした
5. エラー—計算は行われず，収束せずに NaNを返した

[a, *ierr*] = airy (*k*, *z*, *opt*) [Loadable Function]

第 1 種および第 2 種の Airy 関数，およびその導関数を計算する。

```
 K   Function   Scale factor (if a third argument is supplied)
---  --------   -------------------------------------------
 0   Ai (Z)     exp ((2/3) * Z * sqrt (Z))
 1   dAi(Z)/dZ  exp ((2/3) * Z * sqrt (Z))
 2   Bi (Z)     exp (-abs (real ((2/3) * Z *sqrt (Z))))
 3   dBi(Z)/dZ  exp (-abs (real ((2/3) * Z *sqrt (Z))))
```

関数を airy (*z*)の形式で呼び出すと，airy (0, *z*) として実行したものと見なす。

結果は z と同じサイズになる。

必要に応じて *ierr* を指定することにより，以下のような実行結果を得ることができる。これは結果と同じサイズになる。

0. 正常終了
1. 入力エラーにより `NaN`を返した
2. オーバーフローにより `Inf`を返した
3. argument reduction による有効桁のロスにより，計算機精度の半分以下の精度となった
4. argument reduction によって有効桁が完全にロスした
5. エラー—計算は行われず，収束せずに `NaN`を返した

`beta (a, b)` [Mapping Function]

ベータ関数を返す。

$$B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}.$$

`betainc (x, a, b)` [Mapping Function]

不完全ベータ関数を返す。

$$\beta(x,a,b) = B(a,b)^{-1} \int_0^x t^{(a-z)}(1-t)^{(b-1)}dt.$$

x が複数の成分を含むならば，a と b の両方ともスカラでなければならない。x がスカラならば，a と b は同じ次数でなければならない。

`bincoeff (n, k)` [Mapping Function]

以下の計算式において n と k を与えたときの二項係数を返す。

$$\binom{n}{k} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{k!}$$

たとえば，

```
bincoeff (5, 2)
⇒ 10
```

となる。

`erf (z)` [Mapping Function]

誤差関数を計算する。

$$\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$$

`erfc (z)` [Mapping Function]

補誤差関数 $1 - \mathrm{erf}(z)$ を計算する。

`erfinv (z)` [Mapping Function]

誤差関数の逆関数を計算する。

`gamma (z)` [Mapping Function]

ガンマ関数を計算する。

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt.$$

`gammainc (x, a)` [Mapping Function]

不完全ガンマ関数を計算する。

$$\gamma(x, a) = \frac{\int_0^x e^{-t} t^{a-1} dt}{\Gamma(a)}$$

a がスカラならば，`gammainc (x, a)`は x の各要素に対して結果を返す。

x と a の両方ともスカラではないならば，x と a は同じサイズでなければならない。このとき，要素ごとに結果を返す。

`lgamma (a, x)` [Mapping Function]
`gammaln (a, x)` [Mapping Function]

ガンマ関数の自然対数を返す。

`cross (x, y, dim)` [Function File]

3 次元のベクトル x と y の cross product を計算する。

```
cross ([1,1,0], [0,1,1])
⇒ [ 1; -1; 1 ]
```

x と y が行列ならば，最初の次元の 3 要素について cross product を計算する。オプション引数 *dim* を指定すると，*dim* 次元の方向に沿って cross product を計算する。

`commutation_matrix (m, n)` [Function File]

交換行列（commutation matrix）$K_{m,n}$ を返す。これは，$m \times n$ の行列 A に対して，$K_{m,n} \cdot \mathrm{vec}(A) = \mathrm{vec}(A^T)$ となる $mn \times mn$ の行列である。

引数として m のみを与えるならば，$K_{m,m}$ を返す。

統計学ならびに経済学における行列の微分についての応用は，Magnus and Neudecker (1988) を参照せよ。

`duplication_matrix (n)` [Function File]

duplication matrix D_n を返す。これは，$n \times n$ の対称行列 matrices A について，$D_n * \mathrm{vech}(A) = \mathrm{vec}(A)$ なる $n^2 \times n(n+1)/2$ の行列である。

統計学ならびに経済学における行列の微分についての応用は，Magnus and Neudecker (1988) を参照せよ。

19.6 座標変換

`[theta, r] = cart2pol (x, y)` [Function File]
`[theta, r, z] = cart2pol (x, y, z)` [Function File]
Transform cartesian to polar or cylindrical coordinates. *x*, *y* (and *z*) must be of same shape. *theta* describes the angle relative to the x - axis. *r* is the distance to the z - axis (0, 0, z).

`[x, y] = pol2cart (theta, r)` [Function File]
`[x, y, z] = pol2cart (theta, r, z)` [Function File]
Transform polar or cylindrical to cartesian coordinates. *theta*, *r* (and *z*) must be of same shape. *theta* describes the angle relative to the x - axis. *r* is the distance to the z - axis (0, 0, z).

`[theta, phi, r] = cart2sph (x, y, z)` [Function File]
Transform cartesian to spherical coordinates. *x*, *y* and *z* must be of same shape. *theta* describes the angle relative to the x - axis. *phi* is the angle relative to the xy - plane. *r* is the distance to the origin (0, 0, 0).

`[x, y, z] = sph2cart (theta, phi, r)` [Function File]
Transform spherical to cartesian coordinates. *x*, *y* and *z* must be of same shape. *theta* describes the angle relative to the x-axis. *phi* is the angle relative to the xy-plane. *r* is the distance to the origin (0, 0, 0).

19.7 数学的定数

`I` [Built-in Variable]
`J` [Built-in Variable]
`i` [Built-in Variable]
`j` [Built-in Variable]
虚数単位であり，$\sqrt{-1}$ と定義される。これらの組み込み変数は，関数のような挙動をする。ゆえに，その名前を別の目的に使用することができる。もし，これらを変数名として使用して値を代入し，その後クリアするならば，それらは再びあらかじめ特別に定義してあったものとして機能する。詳細は，Section 9.3 [Status of Variables], 頁 53 を参照のこと。

`Inf` [Built-in Variable]
`inf` [Built-in Variable]
無限大を表す。これは，1/0 のような演算，あるいは浮動小数点オーバーフローの結果として返される。

`NaN` [Built-in Variable]
`nan` [Built-in Variable]
非数を表す。これは，0/0，あるいは $\infty - \infty$，さらには NaN を含む演算を行った場合に返される。

ある演算で返された NaN は，別の NaN と比較できない。この挙動は，浮動小数点演算の IEEE 標準において明記されている。ある値が NaN かどうかを判定するには，`isnan`関数を使用すること。

`pi` [Built-in Variable]

円周率である。内部的には，`pi`は‘`4.0 * atan (1.0)`’として計算される。

`e` [Built-in Variable]

自然対数の底である。この定数 e は，$\log(e) = 1.$ を満たす。

`eps` [Built-in Variable]

計算機の精度を表す。正確に言えば，`eps`は計算機の浮動小数点演算機能において，任意の隣り合うふたつの数の相対間隔の最大値である。この数値は，明らかにシステムに依存する。64 ビット IEEE 浮動小数点数値をサポートする計算機において，`eps`は近似的に 2.2204×10^{-16} となる。

`realmax` [Built-in Variable]

浮動小数点値として表すことのできる最も大きな値である。実際の値は，システムに依存する。64 ビット IEEE 浮動小数点数値をサポートする計算機において，`realmax`は近似的に $1.7977 \times 10^{308}.$ となる。

`realmin` [Built-in Variable]

浮動小数点値として表すことのできる最も小さな値である。実際の値は，システムに依存する。64 ビット IEEE 浮動小数点数値をサポートする計算機において，`realmax`は近似的に $2.2251 \times 10^{-308}.$ となる。

20 線形代数

この章では，Octave の線形代数に関する関数について述べています。これら関数の多くに対してのリファレンスは，以下を参照してください：Golub and Van Loan, *Matrix Computations, 2nd Ed.*, Johns Hopkins, 1989, and in LAPACK *Users' Guide*, SIAM, 1992.

20.1 基本的な行列関数

`aa = balance (a, opt)` [Loadable Function]
`[dd, aa] = balance (a, opt)` [Loadable Function]
`[cc, dd, aa, bb] = balance (a, b, opt)` [Loadable Function]

`[dd, aa] = balance (a)` returns `aa = dd \ a * dd`. `aa` is a matrix whose row and column norms are roughly equal in magnitude, and `dd = p * d`, where `p` is a permutation matrix and `d` is a diagonal matrix of powers of two. This allows the equilibration to be computed without roundoff. Results of eigenvalue calculation are typically improved by balancing first.

`[cc, dd, aa, bb] = balance (a, b)` returns `aa = cc*a*dd` and `bb = cc*b*dd`), where `aa` and `bb` have non-zero elements of approximately the same magnitude and `cc` and `dd` are permuted diagonal matrices as in `dd` for the algebraic eigenvalue problem.

The eigenvalue balancing option `opt` is selected as follows:

`"N"`, `"n"`	No balancing; arguments copied, transformation(s) set to identity.
`"P"`, `"p"`	Permute argument(s) to isolate eigenvalues where possible.
`"S"`, `"s"`	Scale to improve accuracy of computed eigenvalues.
`"B"`, `"b"`	Permute and scale, in that order. Rows/columns of a (and b) that are isolated by permutation are not scaled. This is the default behavior.

Algebraic eigenvalue balancing uses standard LAPACK routines.

Generalized eigenvalue problem balancing uses Ward's algorithm (SIAM Journal on Scientific and Statistical Computing, 1981).

`cond (a)` [Function File]
行列の 2 ノルム条件数（condition number）を計算する。`cond (a)`は `norm (a) * norm (inv (a))`と定義され，特異値分解を通して計算される。

`[d, rcond] = det (a)` [Loadable Function]
LAPACK を用いて *a* の行列式（determinant）を計算する。必要であれば，reciprocal condition number の推定値を返す。

`dmult (a, b)` [Function File]
a が長さ `rows (b)`のベクトルであれば，`diag (a) * b`を返す（この関数の方がずっと効率がよい）。

`dot (x, y, dim)` [Function File]
2つのベクトルの内積（dot product）を計算する。x と y が行列であれば，first non-singleton dimension に沿って内積を計算する。オプション引数 *dim* を与えるならば，この次元に沿って内積を計算する。

`lambda = eig (a)` [Loadable Function]
`[v, lambda] = eig (a)` [Loadable Function]
行列の固有値（および固有ベクトル）を計算する。この計算には，最初に Hessenberg 分解を行い，次に固有値が求まるまで Schur 分解を行う。もし望むなら，Schur 分解をさらに実行することにより，固有ベクトルを計算する。

`g = givens (x, y)` [Loadable Function]
`[c, s] = givens (x, y)` [Loadable Function]
スカラ x と y について

$$G \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}$$

となるような，2×2 の直交行列

$$G = \begin{bmatrix} c & s \\ -s' & c \end{bmatrix}$$

を返す。
以下に例を示す。

```
givens (1, 1)
     ⇒   0.70711   0.70711
        -0.70711   0.70711
```

`[x, rcond] = inv (a)` [Loadable Function]
`[x, rcond] = inverse (a)` [Loadable Function]
正方行列 a の逆行列を計算する。必要であれば，reciprocal condition number を推定する。一方，reciprocal condition number が小さいならば，条件が悪いという警告を表示する。

`norm (a, p)` [Function File]
行列 a の p-ノルムを計算する。もし 2 つめの引数を指定しないならば，`p = 2`と仮定する。
a が行列ならば：

$p =$ `1` 1-ノルム，a の絶対値の最大の列和

$p =$ `2` a の最大の特異値

$p =$ `Inf` 無限大ノルム，a の絶対値の最大の行和

$p =$ `"fro"`
Frobenius ノルム，`sqrt (sum (diag (a' * a)))`

a がベクトルまたはスカラならば：

$p =$ `Inf` `max (abs (a))`.

$p =$ `-Inf` `min (abs (a))`.

その他 a の p-ノルム，`(sum (abs (a) .^ p)) ^ (1/p)`

`null (a, tol)` [Function File]

a の零空間（null space）の直交基底を返す。

零空間の次元は，a の特異値をとり，tol よりも大きくない。tol を指定しないならば，それは以下のように計算する。

```
max (size (a)) * max (svd (a)) * eps
```

`orth (a, tol)` [Function File]

a の range space の直交基底を返す。

range space の次元は，a の特異値をとり，tol よりも大きくない。tol を指定しないならば，それは以下のように計算する。

```
max (size (a)) * max (svd (a)) * eps
```

`pinv (x, tol)` [Loadable Function]

x の疑似逆行列（一般化逆行列）を返す。tol より小さな特異値は無視される。

もし 2 番めの引数を省略したならば，以下のように仮定する。

```
tol = max (size (x)) * sigma_max (x) * eps,
```

ここで `sigma_max (x)`は，x は最大の特異値である。

`rank (a, tol)` [Function File]

特異値分解を用いて，a の階数（rank）を計算する。階数は，指定した基準値 tol よりも大きな a の特異値とする。2 番めの引数を省略したならば，これは以下のように計算する。

```
tol = max (size (a)) * sigma(1) * eps;
```

ここで `eps`は計算機の精度であり，`sigma(1)`は a の最大の特異値である。

`trace (a)` [Function File]

a の対角和 `sum (diag (a))`を計算する。

20.2 行列の分解

`chol (a)` [Loadable Function]

正定値である対称行列 a を，以下のように Cholesky 分解する。$R^T R = A$

`h = hess (a)` [Loadable Function]

`[p, h] = hess (a)` [Loadable Function]

行列 a の Hessenberg 分解を行う。

通常，Hessenberg 分解は固有値計算の最初のステップとして使用される。しかし，他にも充分に応用できる（詳しくは Golub, Nash, and Van Loan, IEEE Transactions on Automatic Control, 1979 を参照せよ）。Hessenberg 分解は，以下のような分解である。

$$A = PHP^T$$

ここで P は正方ユニタリ行列（unitary matrix）であり，H は upper Hessenberg（$H_{i,j} = 0, \forall i \geq j + 1$）である。

`[l, u, p] = lu (a)` [Loadable Function]

LAPACK からのサブルーチンを用いて a の LU 分解を計算する。その結果は，オプション戻り値 p による permuted 形式で返される。たとえば，行列 `a = [1, 2; 3, 4]`を与えると，以下のようになる。

```
[l, u, p] = lu (a)
```

returns

```
l =

  1.00000  0.00000
  0.33333  1.00000

u =

  3.00000  4.00000
  0.00000  0.66667

p =

  0  1
  1  0
```

この行列は正方行列である必要はない。

`[q, r, p] = qr (a)` [Loadable Function]

LAPACK からのサブルーチンを用いて a の QR 分解を計算する。その結果は，オプション戻り値 p による permuted 形式で返される。たとえば，行列 `a = [1, 2; 3, 4]`を与えると，以下のようになる。

```
[q, r] = qr (a)
```

returns

```
q =

  -0.31623  -0.94868
  -0.94868   0.31623

r =

  -3.16228  -4.42719
   0.00000  -0.63246
```

`qr`分解は，過剰決定方程式系に対する最小二乗問題の解法の応用である。

$$\min_x \|Ax - b\|_2$$

（過剰決定方程式系とはすなわち，A が tall，thin 行列である。）QR 分解は，$QR = A$ である。ここで Q は直行行列であり，R は上三角行列である。

permuted QR 分解 `[q, r, p] = qr (a)`は，`r`の対角要素がその大きさにおいて減少するような QR 分解を形成する。たとえば，行列 `a = [1, 2; 3, 4]`を与えると，以下のようになる。

```
[q, r, p] = qr(a)
```

returns

```
q =

  -0.44721  -0.89443
  -0.89443   0.44721

r =

  -4.47214  -3.13050
   0.00000   0.44721

p =

   0  1
   1  0
```

permuted qr分解（`[q, r, p] = qr (a)`）は，`span (a)`の直交基底を構築する。

lambda = qz (*a*, *b*) [Loadable Function]

Generalized eigenvalue problem $Ax = sBx$, QZ decomposition. There are three ways to call this function:

1. `lambda = qz(A,B)`

   Computes the generalized eigenvalues λ of $(A - sB)$.

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

   Computes qz decomposition, generalized eigenvectors, and generalized eigenvalues of $(A - sB)$

   $$AV = BV\mathrm{diag}(\lambda)$$
   $$W^T A = \mathrm{diag}(\lambda)W^T B$$
   $$AA = Q^T AZ, BB = Q^T BZ$$

   with Q and Z orthogonal (unitary)$= I$

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

   As in form [2], but allows ordering of generalized eigenpairs for (e.g.) solution of discrete time algebraic Riccati equations. Form 3 is not available for complex matrices, and does not compute the generalized eigenvectors V, W, nor the orthogonal matrix Q.

   opt for ordering eigenvalues of the GEP pencil. The leading block of the revised pencil contains all eigenvalues that satisfy:

   - `"N"` = unordered (default)
   - `"S"` = small: leading block has all |lambda| <=1
   - `"B"` = big: leading block has all |lambda >= 1
   - `"-"` = negative real part: leading block has all eigenvalues in the open left half-plant

`"+"` = nonnegative real part: leading block has all eigenvalues in the closed right half-plane

Note: qz performs permutation balancing, but not scaling (see balance). Order of output arguments was selected for compatibility with MATLAB

See also: balance, dare, eig, schur

`[aa, bb, q, z] = qzhess (a, b)` [Function File]

Compute the Hessenberg-triangular decomposition of the matrix pencil `(a, b)`, returning `aa = q * a * z`, `bb = q * b * z`, with q and z orthogonal. For example,

```
[aa, bb, q, z] = qzhess ([1, 2; 3, 4], [5, 6; 7, 8])
⇒ aa = [ -3.02244, -4.41741;  0.92998,  0.69749 ]
⇒ bb = [ -8.60233, -9.99730;  0.00000, -0.23250 ]
⇒  q = [ -0.58124, -0.81373; -0.81373,  0.58124 ]
⇒  z = [ 1, 0; 0, 1 ]
```

The Hessenberg-triangular decomposition is the first step in Moler and Stewart's QZ decomposition algorithm.

Algorithm taken from Golub and Van Loan, *Matrix Computations, 2nd edition.*

`s = schur (a)` [Loadable Function]

`[u, s] = schur (a, opt)` [Loadable Function]

The Schur decomposition is used to compute eigenvalues of a square matrix, and has applications in the solution of algebraic Riccati equations in control (see `are` and `dare`). `schur` always returns $S = U^T A U$ where U is a unitary matrix ($U^T U$ is identity) and S is upper triangular. The eigenvalues of A (and S) are the diagonal elements of S If the matrix A is real, then the real Schur decomposition is computed, in which the matrix U is orthogonal and S is block upper triangular with blocks of size at most 2×2 along the diagonal. The diagonal elements of S (or the eigenvalues of the 2×2 blocks, when appropriate) are the eigenvalues of A and S.

The eigenvalues are optionally ordered along the diagonal according to the value of `opt`. `opt = "a"` indicates that all eigenvalues with negative real parts should be moved to the leading block of S (used in `are`), `opt = "d"` indicates that all eigenvalues with magnitude less than one should be moved to the leading block of S (used in `dare`), and `opt = "u"`, the default, indicates that no ordering of eigenvalues should occur. The leading k columns of U always span the A-invariant subspace corresponding to the k leading eigenvalues of S.

`s = svd (a)` [Loadable Function]

`[u, s, v] = svd (a)` [Loadable Function]

Compute the singular value decomposition of a

$$A = U \Sigma V^H$$

The function `svd` normally returns the vector of singular values. If asked for three return values, it computes U, S, and V. For example,

```
svd (hilb (3))
```

returns

```
ans =

  1.4083189
  0.1223271
  0.0026873
```

and

```
[u, s, v] = svd (hilb (3))
```

returns

```
u =

  -0.82704   0.54745   0.12766
  -0.45986  -0.52829  -0.71375
  -0.32330  -0.64901   0.68867

s =

  1.40832  0.00000  0.00000
  0.00000  0.12233  0.00000
  0.00000  0.00000  0.00269

v =

  -0.82704   0.54745   0.12766
  -0.45986  -0.52829  -0.71375
  -0.32330  -0.64901   0.68867
```

If given a second argument, `svd` returns an economy-sized decomposition, eliminating the unnecessary rows or columns of u or v.

`[`*housv*`, `*beta*`, `*zer*`] = housh (`*x*`, `*j*`, `*z*`)` [Function File]

Computes householder reflection vector housv to reflect x to be jth column of identity, i.e., (I - beta*housv*housv')x =e(j) inputs x: vector j: index into vector z: threshold for zero (usually should be the number 0) outputs: (see Golub and Van Loan) beta: If beta = 0, then no reflection need be applied (zer set to 0) housv: householder vector

`[`*u*`, `*h*`, `*nu*`] = krylov (`*a*`, `*v*`, `*k*`, `*eps1*`, `*pflg*`);` [Function File]

construct orthogonal basis U of block Krylov subspace; [v a*v a^2*v ... a^(k+1)*v]; method used: householder reflections to guard against loss of orthogonality eps1: threshhold for 0 (default: 1e-12) pflg: flag to use row pivoting (improves numerical behavior) 0 [default]: no pivoting; prints a warning message if trivial null space is corrupted 1 : pivoting performed

outputs: u: orthogonal basis of block krylov subspace h: Hessenberg matrix; if v is a vector then a u = u h otherwise h is meaningless nu: dimension of span of

krylov subspace (based on eps1) if b is a vector and k > m-1, krylov returns h = the Hessenberg decompostion of a.

Reference: Hodel and Misra, "Partial Pivoting in the Computation of Krylov Subspaces", to be submitted to Linear Algebra and its Applications

20.3 行列に対する関数

`expm (a)` [Loadable Function]

Return the exponential of a matrix, defined as the infinite Taylor series

$$\exp(A) = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots$$

The Taylor series is *not* the way to compute the matrix exponential; see Moler and Van Loan, *Nineteen Dubious Ways to Compute the Exponential of a Matrix*, SIAM Review, 1978. This routine uses Ward's diagonal Padé approximation method with three step preconditioning (SIAM Journal on Numerical Analysis, 1977). Diagonal Padé approximations are rational polynomials of matrices $D_q(a)^{-1}N_q(a)$ whose Taylor series matches the first $2q + 1$ terms of the Taylor series above; direct evaluation of the Taylor series (with the same preconditioning steps) may be desirable in lieu of the Padé approximation when $D_q(a)$ is ill-conditioned.

`logm (a)` [Function File]

Compute the matrix logarithm of the square matrix *a*. Note that this is currently implemented in terms of an eigenvalue expansion and needs to be improved to be more robust.

`[result, error_estimate] = sqrtm (a)` [Loadable Function]

Compute the matrix square root of the square matrix *a*.

Ref: Nicholas J. Higham. A new sqrtm for MATLAB. Numerical Analysis Report No. 336, Manchester Centre for Computational Mathematics, Manchester, England, January 1999.

`kron (a, b)` [Function File]

2 つの行列の Kronecker 積を計算する。ブロック同士の積は，以下のように定義する。

```
x = [a(i, j) b]
```

以下に例を示す。

```
kron (1:4, ones (3, 1))
      ⇒  1  2  3  4
         1  2  3  4
         1  2  3  4
```

`x = syl (a, b, c)` [Loadable Function]

Solve the Sylvester equation

$$AX + XB + C = 0$$

using standard LAPACK subroutines. For example,

```
syl ([1, 2; 3, 4], [5, 6; 7, 8], [9, 10; 11, 12])
     ⇒ [ -0.50000, -0.66667; -0.66667, -0.50000 ]
```

21 Nonlinear Equations

Octave can solve sets of nonlinear equations of the form

$$f(x) = 0$$

using the function `fsolve`, which is based on the MINPACK subroutine `hybrd`.

`[x, info, msg] = fsolve (fcn, x0)` [Loadable Function]

Given *fcn*, the name of a function of the form `f (x)` and an initial starting point *x0*, `fsolve` solves the set of equations such that `f(x) == 0`.

If *fcn* is a two-element string array, the first element names the function f described above, and the second element names a function of the form `j (x)` to compute the Jacobian matrix with elements

$$J = \frac{\partial f_i}{\partial x_j}$$

You can use the function `fsolve_options` to set optional parameters for `fsolve`.

`fsolve_options (opt, val)` [Loadable Function]

When called with two arguments, this function allows you set options parameters for the function `fsolve`. Given one argument, `fsolve_options` returns the value of the corresponding option. If no arguments are supplied, the names of all the available options and their current values are displayed.

Options include

`"tolerance"`

Nonnegative relative tolerance.

Here is a complete example. To solve the set of equations

$$-2x^2 + 3xy + 4\sin(y) - 6 = 0$$
$$3x^2 - 2xy^2 + 3\cos(x) + 4 = 0$$

you first need to write a function to compute the value of the given function. For example:

```
function y = f (x)
  y(1) = -2*x(1)^2 + 3*x(1)*x(2)   + 4*sin(x(2)) - 6;
  y(2) =  3*x(1)^2 - 2*x(1)*x(2)^2 + 3*cos(x(1)) + 4;
endfunction
```

Then, call `fsolve` with a specified initial condition to find the roots of the system of equations. For example, given the function `f` defined above,

```
[x, info] = fsolve ("f", [1; 2])
```

results in the solution

```
x =

  0.57983
  2.54621

info = 1
```

A value of `info = 1` indicates that the solution has converged.

The function `perror` may be used to print English messages corresponding to the numeric error codes. For example,

```
perror ("fsolve", 1)
     ⊣ solution converged to requested tolerance
```

22 Quadrature

22.1 Functions of One Variable

`[v, ier, nfun, err] = quad (f, a, b, tol, sing)` [Loadable Function]

Integrate a nonlinear function of one variable using Quadpack. The first argument is the name of the function, the function handle or the inline function to call to compute the value of the integrand. It must have the form

```
y = f (x)
```

where y and x are scalars.

The second and third arguments are limits of integration. Either or both may be infinite.

The optional argument *tol* is a vector that specifies the desired accuracy of the result. The first element of the vector is the desired absolute tolerance, and the second element is the desired relative tolerance. To choose a relative test only, set the absolute tolerance to zero. To choose an absolute test only, set the relative tolerance to zero.

The optional argument *sing* is a vector of values at which the integrand is known to be singular.

The result of the integration is returned in *v* and *ier* contains an integer error code (0 indicates a successful integration). The value of *nfun* indicates how many function evaluations were required, and *err* contains an estimate of the error in the solution.

You can use the function `quad_options` to set optional parameters for `quad`.

`quad_options (opt, val)` [Loadable Function]

When called with two arguments, this function allows you set options parameters for the function `quad`. Given one argument, `quad_options` returns the value of the corresponding option. If no arguments are supplied, the names of all the available options and their current values are displayed.

Options include

`"absolute tolerance"`
: Absolute tolerance; may be zero for pure relative error test.

`"relative tolerance"`
: Nonnegative relative tolerance. If the absolute tolerance is zero, the relative tolerance must be greater than or equal to `max (50*eps, 0.5e-28)`.

Here is an example of using `quad` to integrate the function

$$f(x) = x \sin(1/x) \sqrt{|1 - x|}$$

from $x = 0$ to $x = 3$.

This is a fairly difficult integration (plot the function over the range of integration to see why).

The first step is to define the function:

```
function y = f (x)
  y = x .* sin (1 ./ x) .* sqrt (abs (1 - x));
endfunction
```

Note the use of the 'dot' forms of the operators. This is not necessary for the call to `quad`, but it makes it much easier to generate a set of points for plotting (because it makes it possible to call the function with a vector argument to produce a vector result).

Then we simply call quad:

```
[v, ier, nfun, err] = quad ("f", 0, 3)
     ⇒ 1.9819
     ⇒ 1
     ⇒ 5061
     ⇒ 1.1522e-07
```

Although `quad` returns a nonzero value for *ier*, the result is reasonably accurate (to see why, examine what happens to the result if you move the lower bound to 0.1, then 0.01, then 0.001, etc.).

22.2 Orthogonal Collocation

`[r, amat, bmat, q] = colloc (n, "left", "right")` [Loadable Function]

Compute derivative and integral weight matrices for orthogonal collocation using the subroutines given in J. Villadsen and M. L. Michelsen, *Solution of Differential Equation Models by Polynomial Approximation*.

Here is an example of using `colloc` to generate weight matrices for solving the second order differential equation $u' - \alpha u'' = 0$ with the boundary conditions $u(0) = 0$ and $u(1) = 1$.

First, we can generate the weight matrices for n points (including the endpoints of the interval), and incorporate the boundary conditions in the right hand side (for a specific value of α).

```
n = 7;
alpha = 0.1;
[r, a, b] = colloc (n-2, "left", "right");
at = a(2:n-1,2:n-1);
bt = b(2:n-1,2:n-1);
rhs = alpha * b(2:n-1,n) - a(2:n-1,n);
```

Then the solution at the roots r is

```
u = [ 0; (at - alpha * bt) \ rhs; 1]
     ⇒ [ 0.00; 0.004; 0.01 0.00; 0.12; 0.62; 1.00 ]
```

23 Differential Equations

Octave has two built-in functions for solving differential equations. Both are based on reliable ODE solvers written in Fortran.

23.1 Ordinary Differential Equations

The function `lsode` can be used to solve ODEs of the form

$$\frac{dx}{dt} = f(x, t)$$

using Hindmarsh's ODE solver LSODE.

`[x, istate, msg] lsode (fcn, x_0, t, t_crit)` [Loadable Function]

Solve the set of differential equations

$$\frac{dx}{dt} = f(x, t)$$

with

$$x(t_0) = x_0$$

The solution is returned in the matrix *x*, with each row corresponding to an element of the vector *t*. The first element of *t* should be t_0 and should correspond to the initial state of the system *x_0*, so that the first row of the output is *x_0*.

The first argument, *fcn*, is a string that names the function to call to compute the vector of right hand sides for the set of equations. The function must have the form

```
xdot = f (x, t)
```

in which *xdot* and *x* are vectors and *t* is a scalar.

If *fcn* is a two-element string array, the first element names the function *f* described above, and the second element names a function to compute the Jacobian of *f*. The Jacobian function must have the form

```
jac = j (x, t)
```

in which *jac* is the matrix of partial derivatives

$$J = \frac{\partial f_i}{\partial x_j} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \cdots & \frac{\partial f_3}{\partial x_N} \end{bmatrix}$$

The second and third arguments specify the intial state of the system, x_0, and the initial value of the independent variable t_0.

The fourth argument is optional, and may be used to specify a set of times that the ODE solver should not integrate past. It is useful for avoiding difficulties with singularities and points where there is a discontinuity in the derivative.

After a successful computation, the value of *istate* will be 2 (consistent with the Fortran version of LSODE).

If the computation is not successful, *istate* will be something other than 2 and *msg* will contain additional information.

You can use the function `lsode_options` to set optional parameters for `lsode`.

`lsode_options` (*opt*, *val*) [Loadable Function]

When called with two arguments, this function allows you set options parameters for the function `lsode`. Given one argument, `lsode_options` returns the value of the corresponding option. If no arguments are supplied, the names of all the available options and their current values are displayed.

Options include

`"absolute tolerance"`
: Absolute tolerance. May be either vector or scalar. If a vector, it must match the dimension of the state vector.

`"relative tolerance"`
: Relative tolerance parameter. Unlike the absolute tolerance, this parameter may only be a scalar.

  The local error test applied at each integration step is

```
abs (local error in x(i)) <= rtol * abs (y(i)) + atol(i)
```

`"integration method"`
: A string specifing the method of integration to use to solve the ODE system. Valid values are

  `"adams"`
  `"non-stiff"`
  : No Jacobian used (even if it is available).

  `"bdf"`
  `"stiff"`
  : Use stiff backward differentiation formula (BDF) method. If a function to compute the Jacobian is not supplied, `lsode` will compute a finite difference approximation of the Jacobian matrix.

`"initial step size"`
: The step size to be attempted on the first step (default is determined automatically).

`"maximum order"`
: Restrict the maximum order of the solution method. If using the Adams method, this option must be between 1 and 12. Otherwise, it must be between 1 and 5, inclusive.

`"maximum step size"`
: Setting the maximum stepsize will avoid passing over very large regions (default is not specified).

`"minimum step size"`
: The minimum absolute step size allowed (default is 0).

`"step limit"`
: Maximum number of steps allowed (default is 100000).

Here is an example of solving a set of three differential equations using `lsode`. Given the function

```
function xdot = f (x, t)

  xdot = zeros (3,1);

  xdot(1) = 77.27 * (x(2) - x(1)*x(2) + x(1) \
            - 8.375e-06*x(1)^2);
  xdot(2) = (x(3) - x(1)*x(2) - x(2)) / 77.27;
  xdot(3) = 0.161*(x(1) - x(3));

endfunction
```

and the initial condition `x0 = [ 4; 1.1; 4 ]`, the set of equations can be integrated using the command

```
t = linspace (0, 500, 1000);

y = lsode ("f", x0, t);
```

If you try this, you will see that the value of the result changes dramatically between $t = 0$ and 5, and again around $t = 305$. A more efficient set of output points might be

```
t = [0, logspace (-1, log10(303), 150), \
       logspace (log10(304), log10(500), 150)];
```

See Alan C. Hindmarsh, *ODEPACK, A Systematized Collection of ODE Solvers*, in Scientific Computing, R. S. Stepleman, editor, (1983) for more information about the inner workings of `lsode`.

23.2 Differential-Algebraic Equations

The function `daspk` can be used to solve DAEs of the form

$$0 = f(\dot{x}, x, t), \qquad x(t = 0) = x_0, \dot{x}(t = 0) = \dot{x}_0$$

using Petzold's DAE solver DASPK.

`[x, xdot, istate, msg] = daspk (fcn, x_0, xdot_0, t, t_crit)` [Loadable Function]

Solve the set of differential-algebraic equations

$$0 = f(x, \dot{x}, t)$$

with

$$x(t_0) = x_0, \dot{x}(t_0) = \dot{x}_0$$

The solution is returned in the matrices x and *xdot*, with each row in the result matrices corresponding to one of the elements in the vector t. The first element of t should be t_0 and correspond to the initial state of the system *x_0* and its derivative *xdot_0*, so that the first row of the output x is *x_0* and the first row of the output *xdot* is *xdot_0*.

The first argument, *fcn*, is a string that names the function to call to compute the vector of residuals for the set of equations. It must have the form

```
res = f (x, xdot, t)
```

in which *x*, *xdot*, and *res* are vectors, and *t* is a scalar.

If *fcn* is a two-element string array, the first element names the function f described above, and the second element names a function to compute the modified Jacobian

$$J = \frac{\partial f}{\partial x} + c\frac{\partial f}{\partial \dot{x}}$$

The modified Jacobian function must have the form

```
jac = j (x, xdot, t, c)
```

The second and third arguments to `daspk` specify the initial condition of the states and their derivatives, and the fourth argument specifies a vector of output times at which the solution is desired, including the time corresponding to the initial condition.

The set of initial states and derivatives are not strictly required to be consistent. If they are not consistent, you must use the `daspk_options` function to provide additional information so that `daspk` can compute a consistent starting point.

The fifth argument is optional, and may be used to specify a set of times that the DAE solver should not integrate past. It is useful for avoiding difficulties with singularities and points where there is a discontinuity in the derivative.

After a successful computation, the value of *istate* will be greater than zero (consistent with the Fortran version of DASPK).

If the computation is not successful, the value of *istate* will be less than zero and *msg* will contain additional information.

You can use the function `daspk_options` to set optional parameters for `daspk`.

`daspk_options (opt, val)` [Loadable Function]

When called with two arguments, this function allows you set options parameters for the function `daspk`. Given one argument, `daspk_options` returns the value of the corresponding option. If no arguments are supplied, the names of all the available options and their current values are displayed.

Options include

`"absolute tolerance"`

Absolute tolerance. May be either vector or scalar. If a vector, it must match the dimension of the state vector, and the relative tolerance must also be a vector of the same length.

`"relative tolerance"`

Relative tolerance. May be either vector or scalar. If a vector, it must match the dimension of the state vector, and the absolute tolerance must also be a vector of the same length.

The local error test applied at each integration step is

```
abs (local error in x(i))
     <= rtol(i) * abs (Y(i)) + atol(i)
```

`"compute consistent initial condition"`

Denoting the differential variables in the state vector by '`Y_d`' and the algebraic variables by '`Y_a`', `ddaspk` can solve one of two initialization problems:

1. Given Y_d, calculate Y_a and Y'_d
2. Given Y', calculate Y.

In either case, initial values for the given components are input, and initial guesses for the unknown components must also be provided as input. Set this option to 1 to solve the first problem, or 2 to solve the second (the default default is 0, so you must provide a set of initial conditions that are consistent).

If this option is set to a nonzero value, you must also set the `"algebraic variables"` option to declare which variables in the problem are algebraic.

`"use initial condition heuristics"`

Set to a nonzero value to use the initial condition heuristics options described below.

`"initial condition heuristics"`

A vector of the following parameters that can be used to control the initial condition calculation.

`MXNIT` Maximum number of Newton iterations (default is 5).

`MXNJ` Maximum number of Jacobian evaluations (default is 6).

`MXNH` Maximum number of values of the artificial stepsize parameter to be tried if the `"compute consistent initial condition"` option has been set to 1 (default is 5).

Note that the maximum number of Newton iterations allowed in all is `MXNIT*MXNJ*MXNH` if the `"compute consistent initial condition"` option has been set to 1 and `MXNIT*MXNJ` if it is set to 2.

`LSOFF` Set to a nonzero value to disable the linesearch algorithm (default is 0).

`STPTOL` Minimum scaled step in linesearch algorithm (default is eps^(2/3)).

`EPINIT` Swing factor in the Newton iteration convergence test. The test is applied to the residual vector, premultiplied by the approximate Jacobian. For convergence, the weighted RMS norm of this vector (scaled by the error weights) must be less than `EPINIT*EPCON`, where `EPCON` = 0.33 is the analogous test constant used in the time steps. The default is `EPINIT` = 0.01.

`"print initial condition info"`

Set this option to a nonzero value to display detailed information about the initial condition calculation (default is 0).

`"exclude algebraic variables from error test"`
: Set to a nonzero value to exclude algebraic variables from the error test. You must also set the `"algebraic variables"` option to declare which variables in the problem are algebraic (default is 0).

`"algebraic variables"`
: A vector of the same length as the state vector. A nonzero element indicates that the corresponding element of the state vector is an algebraic variable (i.e., its derivative does not appear explicitly in the equation set.

  This option is required by the `compute consistent initial condition"` and `"exclude algebraic variables from error test"` options.

`"enforce inequality constraints"`
: Set to one of the following values to enforce the inequality constraints specified by the `"inequality constraint types"` option (default is 0).

  1. To have constraint checking only in the initial condition calculation.
  2. To enforce constraint checking during the integration.
  3. To enforce both options 1 and 2.

`"inequality constraint types"`
: A vector of the same length as the state specifying the type of inequality constraint. Each element of the vector corresponds to an element of the state and should be assigned one of the following codes

  -2 Less than zero.

  -1 Less than or equal to zero.

  0 Not constrained.

  1 Greater than or equal to zero.

  2 Greater than zero.

  This option only has an effect if the `"enforce inequality constraints"` option is nonzero.

`"initial step size"`
: Differential-algebraic problems may occaisionally suffer from severe scaling difficulties on the first step. If you know a great deal about the scaling of your problem, you can help to alleviate this problem by specifying an initial stepsize (default is computed automatically).

`"maximum order"`
: Restrict the maximum order of the solution method. This option must be between 1 and 5, inclusive (default is 5).

`"maximum step size"`
: Setting the maximum stepsize will avoid passing over very large regions (default is not specified).

Octave also includes DASSL, an earlier version of *Daspk*, and *dasrt*, which can be used to solve DAEs with constraints (stopping conditions).

`[x, xdot, t_out, istat, msg] = dasrt (fcn [, g], x_0, xdot_0, t [, t_crit])` [Loadable Function]

Solve the set of differential-algebraic equations

$$0 = f(x, \dot{x}, t)$$

with

$$x(t_0) = x_0, \dot{x}(t_0) = \dot{x}_0$$

with functional stopping criteria (root solving).

The solution is returned in the matrices *x* and *xdot*, with each row in the result matrices corresponding to one of the elements in the vector *t_out*. The first element of *t* should be t_0 and correspond to the initial state of the system *x_0* and its derivative *xdot_0*, so that the first row of the output *x* is *x_0* and the first row of the output *xdot* is *xdot_0*.

The vector *t* provides an upper limit on the length of the integration. If the stopping condition is met, the vector *t_out* will be shorter than *t*, and the final element of *t_out* will be the point at which the stopping condition was met, and may not correspond to any element of the vector *t*.

The first argument, *fcn*, is a string that names the function to call to compute the vector of residuals for the set of equations. It must have the form

```
res = f (x, xdot, t)
```

in which *x*, *xdot*, and *res* are vectors, and *t* is a scalar.

If *fcn* is a two-element string array, the first element names the function *f* described above, and the second element names a function to compute the modified Jacobian

$$J = \frac{\partial f}{\partial x} + c\frac{\partial f}{\partial \dot{x}}$$

The modified Jacobian function must have the form

```
jac = j (x, xdot, t, c)
```

The optional second argument names a function that defines the constraint functions whose roots are desired during the integration. This function must have the form

```
g_out = g (x, t)
```

and return a vector of the constraint function values. If the value of any of the constraint functions changes sign, DASRT will attempt to stop the integration at the point of the sign change.

If the name of the constraint function is omitted, `dasrt` solves the same problem as `daspk` or `dassl`.

Note that because of numerical errors in the constraint functions due to roundoff and integration error, DASRT may return false roots, or return the same root at two or more nearly equal values of *T*. If such false roots are suspected, the user should consider smaller error tolerances or higher precision in the evaluation of the constraint functions.

If a root of some constraint function defines the end of the problem, the input to DASRT should nevertheless allow integration to a point slightly past that root, so that DASRT can locate the root by interpolation.

The third and fourth arguments to `dasrt` specify the initial condition of the states and their derivatives, and the fourth argument specifies a vector of output times at which the solution is desired, including the time corresponding to the initial condition.

The set of initial states and derivatives are not strictly required to be consistent. In practice, however, DASSL is not very good at determining a consistent set for you, so it is best if you ensure that the initial values result in the function evaluating to zero.

The sixth argument is optional, and may be used to specify a set of times that the DAE solver should not integrate past. It is useful for avoiding difficulties with singularities and points where there is a discontinuity in the derivative.

After a successful computation, the value of *istate* will be greater than zero (consistent with the Fortran version of DASSL).

If the computation is not successful, the value of *istate* will be less than zero and *msg* will contain additional information.

You can use the function `dasrt_options` to set optional parameters for `dasrt`.

`dasrt_options` (*opt*, *val*) [Loadable Function]

When called with two arguments, this function allows you set options parameters for the function `dasrt`. Given one argument, `dasrt_options` returns the value of the corresponding option. If no arguments are supplied, the names of all the available options and their current values are displayed.

Options include

`"absolute tolerance"`
: Absolute tolerance. May be either vector or scalar. If a vector, it must match the dimension of the state vector, and the relative tolerance must also be a vector of the same length.

`"relative tolerance"`
: Relative tolerance. May be either vector or scalar. If a vector, it must match the dimension of the state vector, and the absolute tolerance must also be a vector of the same length.

  The local error test applied at each integration step is

```
abs (local error in x(i)) <= rtol(i) * abs (Y(i)) + atol(i)
```

`"initial step size"`
: Differential-algebraic problems may occaisionally suffer from severe scaling difficulties on the first step. If you know a great deal about the scaling of your problem, you can help to alleviate this problem by specifying an initial stepsize.

`"maximum order"`
: Restrict the maximum order of the solution method. This option must be between 1 and 5, inclusive.

`"maximum step size"`
: Setting the maximum stepsize will avoid passing over very large regions.

`"step limit"`
: Maximum number of integration steps to attempt on a single call to the underlying Fortran code.

See K. E. Brenan, et al., *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, North-Holland (1989) for more information about the implementation of DASSL.

24 最適化

24.1 Linear Programming

24.2 Quadratic Programming

24.3 Nonlinear Programming

24.4 最小二乗法

`[beta, v, r] = gls (y, x, o)` [Function File]

多変量モデル $y = xb + e$ で，$\bar{e} = 0$ および $\mathrm{cov}(\mathrm{vec}(e)) = (s^2)o$ に対する一般化最小二乗推定を行う。ここで，y は $t \times p$ の行列，x は $t \times k$ の行列，b は $k \times p$ の行列，e は $t \times p$ の行列であり，o は $tp \times tp$ の行列である。

y と x の各行は観察値であり，各列は変数である。戻り値 *beta*，v および r は以下のように定義される。

beta b についての GLS 推定値である。

v s^2 についての GLS 推定値である。

r GLS 残差 $r = y - xbeta$ についての行列である。

`[beta, sigma, r] = ols (y, x)` [Function File]

以下の多変量モデルに対して通常最小二乗推定を行う: $y = xb + e$ について $\bar{e} = 0$，および $\mathrm{cov}(\mathrm{vec}(e)) = \mathrm{kron}\ (s, I)$ であり，ここで y は $t \times p$ の行列，x は $t \times k$ の行列，b は $k \times p$ の行列であり，e は $t \times p$ の行列である。

y と x の各行は観測地であり，各列は変数である。

y と x の各行は観察値であり，各列は変数である。戻り値 *beta*，v および r は以下のように定義される。

beta b に対する OLS 推定量 `beta = pinv (x) * y` である。ここで `pinv (x)`は，x の疑似（一般化）逆行列を表す。

sigma 行列 s の OLS 推定量である:

```
sigma = (y-x*beta)'
  * (y-x*beta)
  / (t-rank(x))
```

r OLS 残差 $r = y - x * beta$ についての行列である。

25 統計

いつの日か，Octaveにもっと統計関数を含めることができればと考えています。もしあなたが，Octaveをこの領域で充実させる手助けをしたいと思うのならば，bug@octave.orgへコンタクトをとってください。

25.1 基本的な統計関数

`mean (x, dim, opt)` [Function File]

x がベクトルならば，x の平均

$$\mathrm{mean}(x) = \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

を計算する。もし x が行列ならば，各列について平均を計算し，列ベクトルとして返す。

オプション引数 *opt* をつけると，計算すべき平均の種類を選択できる。以下のオプションを認識する。

`"a"` （通常の）算術平均を計算する。これが初期値である。

`"g"` 幾何平均を計算する。

`"h"` 調和平均を計算する。

もしオプション引数 *dim* が与えられるならば，次元 *dim* に沿って計算する。

dim と *opt* は両方ともオプションである。もし両方を与えるならば，どちらを最初にしてもよい。

`median (x)` [Function File]

もし x がベクトルならば，以下の要素の中央値を計算する。x.

$$\mathrm{median}(x) = \begin{cases} x(\lceil N/2 \rceil), & N \text{ odd;} \\ (x(N/2) + x(N/2+1))/2, & N \text{ even.} \end{cases}$$

もし x が行列ならば，各列に対する中央値を計算し，行ベクトルとして返す。

`std (x)` [Function File]
`std (x, opt)` [Function File]
`std (x, opt, dim)` [Function File]

もし x がベクトルならば，x の要素の標準偏差を計算する。

$$\mathrm{std}(x) = \sigma(x) = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})}{N-1}}$$

もし x が行列ならば，各列について標準偏差を計算し，それらを行ベクトルとして返す。

引数 *opt* は，使用すべき標準化の種類を決定する。個々に指定できる値は，以下のようなものである。

0: N-1 で標準化する。これは，分散の最良不偏推定量の平方根を提供する（初期設定）。

1: N で標準化する。これは，平均まわりの第 2 モーメントの平方根を提供する。

第 3 の引数 *dim* は，標準偏差を計算するために沿う次元を決定する。

`cov (x, y)` [Function File]

もし x t p y の各行が観測値であり，各列が変数ならば，`cov (x, y)`の (i, j) 番めのエントリは，x の i 番めの変数と y の j 番めの変数との間の共分散である。1 個の引数を付けて呼び出すと，`corrcoef (x, x)` を計算する。

`corrcoef (x, y)` [Function File]

もし x t p y の各行が観測値であり，各列が変数ならば，`corrcoef (x, y)`の (i, j) 番めのエントリは，x の i 番めの変数と y の j 番めの変数との間の相関である。1 個の引数を付けて呼び出すと，`corrcoef (x, x)` を計算する。

`kurtosis (x, dim)` [Function File]

もし x が長さ N のベクトルならば，x の尖度

$$\text{kurtosis}(x) = \frac{1}{N\sigma(x)^4}\sum_{i=1}^{N}(x_i - \bar{x})^4 - 3$$

を返す。もし x が行列ならば，first non-singleton dimension にわたって尖度を返す。オプション引数 *dim* は，その次元にわたって尖度を得るようにする。

`mahalanobis (x, y)` [Function File]

多変量サンプル x と y 間について，マハラノビスの D-二乗距離を返す。これらは，同じ成分数（列数）でなければならないが，異なる観察値数（行数）であってもよい。

`skewness (x, dim)` [Function File]

x が長さ n のベクトルならば，x の歪度

$$\text{skewness}(x) = \frac{1}{N\sigma(x)^3}\sum_{i=1}^{N}(x_i - \bar{x})^3$$

を返す。もし x が行列ならば，その first non-singleton dimension に沿って歪度を返す。もしオプション引数 *dim* が与えられれば，この次元に沿って計算する。

`values (x)` [Function File]

列ベクトルに存在する互いに異なる値を，昇順で返す。

`var (x)` [Function File]

x がベクトルならば，(真の）分散を返す。x が行列ならば，各列についての分散を含む行ベクトルを返す。

引数 *opt* は，使用すべき標準化の種類を決定する。個々に指定できる値は，以下のようなものである。

0: N-1 で標準化する。これは，分散を提供する（初期設定)。

1: N で標準化する。これは，平均まわりの第 2 モーメントを提供する。

第 3 の引数 *dim* は，分散を計算するために沿う次元を決定する。

`[t, l_x] = table (x)` [Function File]
`[t, l_x, l_y] = table (x, y)` [Function File]
データベクトルから分割表 t を作成する。ベクトル l は対応する水準である。

現在のところ，1 次または 2 次元の表だけをサポートしている。

`studentize (x, dim)` [Function File]
もし x がベクトルならば，その平均を引き，その標準偏差で除す。

もし x が行列ならば，first non-singleton dimension について計算し，列ベクトルとして返す。もしオプション引数 dim が与えられるならば，次元 dim に沿って計算する。

`statistics (x)` [Function File]
もし x が行列ならば，x の各列について最小値，第 3 分位数，中央値，第 3 分位数，最大値，平均，標準偏差，歪度，尖度を返す。

もし x がベクトルならば，それは列ベクトルとして扱われる。

`spearman (x, y)` [Function File]
入力引数によって指定された変数の各々について，スピアマンの順位相関係数 rho を計算する

行列について，各行が観測値であり，各列が変数である。すなわち，ベクトルは常に観測値であり，行ベクトルでも列ベクトルでも良い。

`spearman (x)`は，`spearman (x,x)`と等価である。

2 つのデータベクトル x と y について，スピアマンの rho は x と y の順位相関となる。

もし x と y が同じ分布から導かれたものであるならば，rho は平均がゼロで分散が `1 / (n - 1)`となり，漸近的に正規分布をなす。

`run_count (x, n)` [Function File]
Count the upward runs along the first non-singleton dimension of x of length 1, 2, ..., n-1 and greater than or equal to n. If the optional argument dim is given operate along this dimension

`ranks (x, dim)` [Function File]
もし x がベクトルならば，ties について補正した x の階数（ランク）の（列）ベクトルを返す。

もし x が行列ならば，first non-singleton dimension に沿って上記の計算を実行する。もしオプション引数 dim が与えられるならば，次元 dim に沿って計算する。

`range (x)` [Function File]
`range (x, dim)` [Function File]
もし x がベクトルならば，その範囲，すなわち入力データの最大値と最小値の差を返す。

もし x が行列ならば，x の各列について計算する。

もしオプション引数 dim が与えられるならば，次元 dim に沿って計算する。

`[q, s] = qqplot (x, dist, params)` [Function File]

QQ-plot（分位点プロット）を実行する。

パラメータが *params* である分布 *dist* の CDF を F，その逆関数を G とし，長さが n のサンプルベクトルを x とすると，QQ-plot は，縦座標が $s(i) = (x \text{ の } i \text{ 番めの最大要素})$ 対，横座標が $q(\text{if}) = \text{G}((i - 0.5)/n)$ のグラフを描く。

そのサンプルが，位置とスケールの変換を除いて F から由来しているなら，そのペアは近似的に直線となる。

dist の初期値は標準正規分布である。オプション引数 *params* は，*dist* のパラメータのリストを含む。たとえば，範囲が [2,4] の一様分布と x の分位点プロットについて，以下の式を使用する。

```
qqplot (x, "uniform", 2, 4)
```

もし出力引数が何も与えられなければ，そのデータが直接プロットされる。

`probit (p)` [Function File]

p の各成分について，p のプロビット（標準正規分布の分位数）を返す。

`[p, y] = ppplot (x, dist, params)` [Function File]

PP-plot（確率プロット）を実行する。

パラメータが *params* である分布 *dist* の CDF を F とし，長さが n のサンプルベクトルを x とすると，PP-plot は，縦座標が $y(i) = \text{F}$ （x の i 番めに大きな要素）に対して，横座標が $p(i) = (i - 0.5)/n$ としてグラフを描く。もしサンプルが F から由来するならば，そのペアは近似的に直線に従う。

dist の初期値は標準正規分布である。オプション引数 *params* は，*dist* のパラメータのリストを含む。たとえば，範囲が [2,4] の一様分布と x の確率プロットについて，以下の式を使用する。

```
ppplot (x, "uniform", 2, 4)
```

もし出力引数が何も与えられなければ，そのデータが直接プロットされる。

`moment (x, p, opt, dim)` [Function File]

もし x がベクトルならば，x の p 番目のモーメントを計算する。

もし x が行列ならば，各列の p 番目のモーメントを含む行ベクトルを返す。

オプション引数 *opt* をつけると，計算すべきモーメントの種類を指定することができる。*opt* が`"c"`または`"a"`を含むならば，中心 and/or 絶対モーメントを返す。たとえば，

```
moment (x, 3, "ac")
```

この式は，x の 3 番目の中心絶対モーメントを計算する。

もしオプション引数 *dim* が与えられるならば，次元 *dim* に沿って計算する。

`meansq (x)` [Function File]
`meansq (x, dim)` [Function File]

ベクトルの引数については，値の平均平方を返す。引数が行列ならば，各列の平均平方を含む行ベクトルを返す。オプション引数 *dim* をつけると，この次元に沿って，値の平均平方を返す。

`logit (p)` [Function File]

p の各成分について，p のロジット `log (p / (1-p))`を返す。

`kendall (x, y)` [Function File]

入力した引数によって指定される変数の各々に対してケンドールのタウ（*tau*）を計算する。

行列については，各行は観測値であり，各列が変数である；ベクトルは常に観測値であり，列または行ベクトルのどちらでもよい。

`kendall (x)`は`kendall (x,x)`に等しい。

同じ長さの 2 つのデータベクトル x と y について，ケンドールの *tau* は，x と y の全ての順位差の符号の相関である；すなわち，もし x と y が distinct entries であるならば，

$$\tau = \frac{1}{n(n-1)} \sum_{i,j} \text{sign}(q_i - q_j)\text{sign}(r_i - r_j)$$

であって，ここで in which the q_i および r_i は，それぞれ x と y の順位である。

もし x と y が独立な分布から引き出されたならば，ケンドールの *tau* は漸近的に，平均が 0 で分散が `(2 * (2n+5)) / (9 * n * (n-1))` の正規分布に従う。

`iqr (x, dim)` [Function File]

もし x がベクトルならば，四分位範囲（interquartile range），すなわち，入力データの第 1 四分位数と第 3 四分位数の差を返す。

x が行列ならば，x の first non singleton dimension に対して上記の演算を行う。もしオプション引数 *dim* が与えられれば，その次元に沿って演算を行う。

`cut (x, breaks)` [Function File]

数値あるいは連続的データから，ある間隔に分けることによって，カテゴリ分けされたデータを生成する。

もし *breaks* がスカラならば，そのデータは多くの等間隔の幅へと分けられる。もし *breaks* が分割点のベクトルならば，そのカテゴリは`length (breaks) - 1`個のグループを持つ。

返される値は，x と同じサイズで，x の各点が入るグループを伝えるベクトルである。グループは，1 からグループ数までのラベルを付けされる。*breaks* を外れる点は，`NaN`というラベルが付けられる。

`cor (x, y)` [Function File]

`cor (x, y)`の (i, j) 番めのエントリは，x の i 番めの変数と y の j 番めの変数との間の相関である。

行列に対しては，各行は観測値であり，各列は変数である。ベクトルは，常に観測値であって，行または列ベクトルのどちらでもよい。

`cor (x)`は，`cor (x, x)`と等価である。

`cloglog (x)` [Function File]

x の complementary log-log 関数（以下に定義する）を返す。

```
- log (- log (x))
```

`center (x)` [Function File]

`center (x, dim)` [Function File]

もし x がベクトルならば，それぞれの要素からその平均を減じる。x が行列ならば，各列について前述の動作を行う。もしオプション引数 *dim* を与えるならば，この次元に沿って前述の操作を実行する。

25.2 検定

`[pval, f, df_b, df_w] = anova (y, g)` [Function File]

一元配置の分散分析（ANOVA）を実行する。この目的は，k 個の異なるグループからとられたデータの集団平均が全て等しいかどうかを検定することである。

データは，対応するグループラベル（たとえば，1 から k までの数字）のベクトル g によって指定されたグループごとに 1 個のベクトル y で与えることになる。これは，各グループあるいはグループラベルについて，データの数には何の制限もない一般の形式である。

y が行列であり，g が省略されるならば，y の各列は同じグループとして扱われる。この形式は，各グループからのサンプル数が全て等しい釣り合い型 ANOVA についてのみ適切である。

平均が等しいという帰無仮説の下では，統計量 f は自由度が df_b と df_w である F 分布に従う。

その確率（この分布の点 f における，1 マイナス CDF）は，*pval* に返る。

もし出力引数が与えられなければ，標準的な一元配置の分散分析表を表示する。

`[pval, chisq, df] = bartlett_test (x1, ...)` [Function File]

データベクトル *x1*，*x2*，...，*xk*（$k > 1$）における分散の均一性について Bartlett 検定を実行する。

分散が等しいという帰無仮説の下では，検定統計量 *chisq* は近似的に自由度 *df* のカイ二乗分布に従う。

その p-値（この分布の点 *chisq* における，1 マイナス CDF）は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, chisq, df] = chisquare_test_homogeneity (x, y, c)` [Function File]

2 つのサンプル x と y を与えるとき，x と y が同じ分布から由来したという帰無仮説の均一性のカイ二乗検定を実行する。これは c の（厳密には増加する）エントリによって指定した分割に基づく。

大標本について，検定統計量 *chisq* は近似的に自由度 df = `length (c)`のカイ二乗分布に従う。

その確率（この分布の点 *chisq* における，1 マイナス CDF）は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, chisq, df] = chisquare_test_independence (x)` [Function File]

分割表 x に基づいて独立性のカイ二乗検定を実行する。独立という帰無仮説の下では，*chisq* は自由度 *df* のカイ二乗分布に近似的に従う。

その確率（この分布の点 *chisq* における，1 マイナス CDF）は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`cor_test (x, y, alt, method)` [Function File]

2 つのサンプル x と y が，相関する母集団からのものであるかどうかを検定する。

オプション引数の文字列 *alt* は，対立仮説を記述する。これには，`"!="`または`"<>"`（ゼロではない），`">"`（0 より大きい），`"<"`（0 より小さい）をとることができる。標準設定は，両側検定である。

オプション引数の文字列 *method* は，検定が基礎とすべき相関係数を指定する。もし *method* が"pearson"（標準設定）ならば,（通常の）ピアソンの積率相関係数を使用する。この場合，そのデータは 2 変量正規分布から由来するべきである。一方で，その他の 2 つの方法は，ノンパラメトリック対立仮説を提供する。もし *method* が"kendall"ならば，ケンドールの順位相関 *tau* が使用される。*method* が"spearman"ならば，スピアマンの順位相関 *rho* が使用される。最初の文字だけが必要である。

その出力は，以下の要素をもつ構造体である。

pval 検定の p-値である。

stat 検定統計量の値である。

dist 検定統計量の分布である。

params 検定統計量の帰無分布のパラメータである。

alternative
対立仮説である。

method 検定に使用した方法である。

もし出力引数が与えられなければ，p-値を表示する。

`[`*pval*`,` *f*`,` *df_num*`,` *df_den*`] = f_test_regression (`*y*`,` *x*`,` *rr*`,` *r*`)` [Function File]

古典的な正規回帰モデル y = X * b + e において，帰無仮説 rr * b = r に対する F 検定を実行する。

帰無仮説の下では，検定統計量 *f* は自由度 *df_num* と *df_den* の F 分布に従う。

その確率（この分布の点 *f* における，1 マイナス CDF）は，*pval* に返る。

もし明示的に与えられなければ，$r = 0$ とする。

もし出力引数が与えられなければ，p-値を表示する。

`[`*pval*`,` *tsq*`] = hotelling_test (`*x*`,` *m*`)` [Function File]

未知の平均と共分散行列である多変量正規分布からのサンプル *x* について，`mean(`*x*`) == `*m* という帰無仮説を検定する。

tsq には Hotelling の T^2 が返る。帰無仮説の下では，$(n-p)T^2/(p(n-1))$ は自由度が p と $n-p$ である F 分布に従う。ここで n と p は，それぞれサンプルと変数の数である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[`*pval*`,` *tsq*`] = hotelling_test_2 (`*x*`,` *y*`)` [Function File]

未知の平均と共分散行列である多変量正規分布からのサンプルで，変数（列）の数が同じ *x* について，`mean(`*x*`) == `*m* という帰無仮説を検定する。

tsq には Hotelling の T^2 が返る。帰無仮説の下では，

```
(n_x+n_y-p-1) T^2 / (p(n_x+n_y-2))
```

は自由度が p と $n_x + n_y - p - 1$ である F 分布に従う。ここで n_x と n_y は，それぞれサンプルサイズおよび変数の数である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, ks] = kolmogorov_smirnov_test (x, dist, params, alt)` [Function File]

サンプル *x* が連続分布 *dist* からのものであるという帰無仮説の Kolmogorov-Smirnov 検定を実行する。すなわち，もし F と G が，それぞれサンプルおよび *dist* に対応する CDF ならば，帰無仮説は F == G である。

オプション引数 *params* は，*dist* のパラメータのリストを含む。たとえば，サンプル *x* が範囲 [2,4] における一様分布からのものであるかどうか検定するには，以下の式を使用する。

```
kolmogorov_smirnov_test(x, "uniform", 2, 4)
```

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が `"!="`または`"<>"`ならば，帰無仮説は，両側対立仮説 F != G に対して検定される。この場合，検定統計量 *ks* は両側 Kolmogorov-Smirnov 分布に従う。*alt* が`">"` ならば，片側対立仮説 F > G を考慮する。同様に，`"<"`について，片側対立仮説 F < G を考慮する。この場合，検定統計量 *ks* は片側 Kolmogorov-Smirnov 分布になる。標準設定は両側検定である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, ks, d] = kolmogorov_smirnov_test_2 (x, y, alt)` [Function File]

サンプル *x* と *y* が同じ（連続）分布からのものであるという帰無仮説の 2 サンプル Kolmogorov-Smirnov 検定を実行する。すなわち，もし F と G は，それぞれサンプルおよび *dist* に対応する CDF ならば，帰無仮説は F == G である。

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が `"!="`または`"<>"`ならば，帰無仮説は，両側対立仮説 F != G に対して検定される。この場合，検定統計量 *ks* は両側 Kolmogorov-Smirnov 分布に従う。*alt* が`">"` ならば，片側対立仮説 F > G を考慮する。同様に，`"<"`について，片側対立仮説 F < G を考慮する。この場合，検定統計量 *ks* は片側 Kolmogorov-Smirnov 分布になる。標準設定は両側検定である。

その p-値は，*pval* に返る。

3 番めの戻り値 *d* は，検定統計量であり，2 つの累積分布関数間の最大の垂直距離である。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, k, df] = kruskal_wallis_test (x1, ...)` [Function File]

Kruskal-Wallis1 要因「分散分析」を実行する。

ある変数は，*k* > 1 の異なるグループについて観測されたと仮定し，*x1*，...，*xk* が対応するデータベクトルとする。

プールしたサンプルにおける順位がグループのメンバによって影響されるないという帰無仮説の下では，検定統計量 *k* は自由度が *df* = *k* - 1 であるカイ二乗分布に近似的に従う。

その確率（この分布の点 *k* における，1 マイナス CDF）は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`manova (y, g)` [Function File]

一元配置の多変量分散分析を実行する。この目的は，*k* 個の異なるグループからとられたデータの集団平均が全て等しいかどうかを検定することである。全てのデータは，同じ共分散行列を持つ p-次元正規分布から独立に得られたと仮定する。

データ行列は *y* によって与えられる。通常，行は観測値であり，列は変数である。ベクトル *g* は，対応するグループラベル（たとえば，1 から *k* までの数字）を指定する。

LR 検定統計量（Wilks' Lambda）と近似的な p-値は計算され，表示される。

`[pval, chisq, df] = mcnemar_test (x)` [Function File]

行および列の変数における交差分類データの平方分割表 x について，McNemar の検定は，クラス分けされた確率の対象性を帰無仮説として検定することができる。

帰無仮説の下では，*chisq* は近似的に自由度 *df* のカイ二乗分布をする。

その確率（この分布の点 k における，1 マイナス CDF）は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, z] = prop_test_2 (x1, n1, x2, n2, alt)` [Function File]

x1 と *n1* が 1 つのサンプルにおける成功と試行の回数であり，*x2* と *n2* が 2 つめのサンプルについてのものであるとき，成功確率 *p1* と *p2* が同じという帰無仮説を検定する。帰無仮説のもとでは，検定統計量 z は近似的に標準正規分布に従う。

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が `"!="`または`"<>"`ならば，帰無仮説は，両側対立仮説 *p1*!=*p2* に対して検定される。*alt* が`">"`ならば，片側対立仮説 *p1*>*p2* を使用する。同様に，`"<"`について，片側対立仮説 *p1*<*p2* を使用する。標準設定は両側検定である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, chisq] = run_test (x)` [Function File]

x の列に置いて，upward runs に基づく自由度 6 のカイ二乗検定を実行する。x が独立なデータを含むかどうかを検定することができる。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, b, n] = sign_test (x, y, alt)` [Function File]

2 つの対サンプル x と y について，帰無仮説 PROB $(x > y)$ == PROB $(x < y)$ == 1/2 の符号検定を実行する。帰無仮説の下では，検定統計量 b は，パラメータが `n = sum (x != y)`および $p = 1/2$ の二項分布に，おおざっぱに従う。

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が `"!="`または`"<>"`ならば，帰無仮説は，両側対立仮説 PROB($x < y$)!= 1/2 に対して検定される。*alt* が`">"`ならば，片側対立仮説 PROB $(x > y) > 1/2$（「x は y よりも統計的に大きい」）を考慮する。同様に，`"<"`について，片側対立仮説 PROB $(x > y) < 1/2$（「x は y よりも統計的に小さい」）を考慮する。標準設定は両側検定である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, t, df] = t_test (x, m, alt)` [Function File]

未知の平均と分散を持つ正規分布からのサンプル x について，帰無仮説 `mean(x) == m`の t-検定を実行する。帰無仮説の下では，検定統計量 t は自由度が `df = length (x) - 1`であるスチューデント分布に従う。

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が`"!="`または`"<>"`ならば，帰無仮説は，両側対立仮説 `mean (x) != m`に対して検定される。*alt* が`">"`ならば，片側対立仮説 `mean (x) > m`を考慮する。同様に，`"<"`について，片側対立仮説 `mean (x) < m`を考慮する。標準設定は両側検定である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, t, df] = t_test_2 (x, y, alt)` [Function File]

未知の平均と分散を持つ正規分布からのサンプル x と y について，平均が等しいという帰無仮説の 2 サンプル t-検定を実行する。帰無仮説の下では，検定統計量 t は自由度が `df` であるスチューデント分布に従う。

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が `"!="` または `"<>"` ならば，帰無仮説は，両側対立仮説 `mean (x) != m` に対して検定される。*alt* が `">"` ならば，片側対立仮説 `mean (x) > m` を考慮する。同様に，`"<"` について，片側対立仮説 `mean (x) < m` を考慮する。標準設定は両側検定である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, t, df] = t_test_regression (y, x, rr, r, alt)` [Function File]

古典的な正規回帰モデル `y = x * b + e` において，帰無仮説 `rr * b = r` に対する t 検定を実行する。帰無仮説の下では，検定統計量 t は自由度が `df` であるスチューデント分布に従う。

もし r が省略されるならば，0 を仮定する。

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が `"!="` または `"<>"` ならば，帰無仮説は，両側対立仮説 `rr * b != r` に対して検定される。*alt* が `">"` ならば，片側対立仮説 `rr * b > r` を考慮する。同様に，`"<"` について，片側対立仮説 `rr * b < r` を考慮する。標準設定は両側検定である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, z] = u_test (x, y, alt)` [Function File]

2 つのサンプル x と y について，帰無仮説が PROB $(x > y) == 1/2 ==$ PROB $(x < y)$ である Mann-Whitney の U 検定を実行する。帰無仮説の下では，検定統計量 z は近似的に標準正規分布に従う。この検定は Wilcoxon の順位和検定と等価である。

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が `"!="` または `"<>"` ならば，帰無仮説は，両側対立仮説 PROB $(x > y)$!= $1/2$ に対して検定される。*alt* が `">"` ならば，片側対立仮説 PROB $(x > y) > 1/2$ を考慮する。同様に，`"<"` について，片側対立仮説 PROB $(x > y) < 1/2$ を考慮する。標準設定は両側検定である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, f, df_num, df_den] = var_test (x, y, alt)` [Function File]

未知の平均と分散を持つ正規分布からの 2 つのサンプル x と y について，帰無仮説が等分散である F 検定を行う。帰無仮説の下では，検定統計量は，自由度が *df_num* と *df_den* の F 分布に従う。

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が `"!="` または `"<>"` ならば，帰無仮説は，両側対立仮説 `var (x) != var (y)` に対して検定される。*alt* が `">"` ならば，片側対立仮説 `var (x) > var (y)` を考慮する。同様に，`"<"` について，片側対立仮説 `var (x) < var (y)` を考慮する。標準設定は両側検定である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, t, df] = welch_test (x, y, alt)` [Function File]

未知の平均と未知の分散（等しい必要はない）である正規分布からの 2 つのサンプル x と y について，平均が等しいという帰無仮説の Welch の検定を実行する。帰無仮説の下では，検定統計量 t は，自由度が df の t 分布に近似的に従う。

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が`"!="`または`"<>"`ならば，帰無仮説は，両側対立仮説 `mean (x) != m` に対して検定される。*alt* が`">"`ならば，片側対立仮説 `mean (x) > m` を考慮する。同様に，`"<"`について，片側対立仮説 `mean (x) < m` を考慮する。標準設定は両側検定である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, z] = wilcoxon_test (x, y, alt)` [Function File]

2 つのサンプル x と y について，帰無仮説が PROB $(x > y) == 1/2$ である Wilcoxon の符号和検定を実行する。帰無仮説の下では，検定統計量 z は近似的に標準正規分布に従う。この検定は Wilcoxon の順位和検定と等価である。

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が`"!="`または`"<>"`ならば，帰無仮説は，両側対立仮説 PROB $(x > y)$!= $1/2$ に対して検定される。*alt* が`">"`ならば，片側対立仮説 PROB $(x > y) > 1/2$ を考慮する。同様に，`"<"`について，片側対立仮説 PROB $(x > y) < 1/2$ を考慮する。標準設定は両側検定である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, z] = z_test (x, m, v, alt)` [Function File]

未知の平均と既知の分散 v をもつ正規分布からのサンプル x について，帰無仮説が `mean (x) == m` である Z-検定を実行する。帰無仮説の下では，検定統計量 z は標準正規分布に従う。

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が`"!="`または`"<>"`ならば，帰無仮説は，両側対立仮説 `mean (x) != m` に対して検定される。*alt* が`">"`ならば，片側対立仮説 `mean (x) > m` を考慮する。同様に，`"<"`について，片側対立仮説 `mean (x) < m` を考慮する。標準設定は両側検定である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

`[pval, z] = z_test_2 (x, y, v_x, v_y, alt)` [Function File]

未知の平均と既知の分散 v_x と v_y をもつ正規分布からの 2 つのサンプル x と y について，平均が等しいという帰無仮説に対して Z-検定を行う。帰無仮説の下では，検定統計量 z は標準正規分布に従う。

オプションの引数文字列 *alt* について，興味ある対立仮説を選択することができる。もし *alt* が`"!="`または`"<>"`ならば，帰無仮説は，両側対立仮説 `mean (x) != m` に対して検定される。*alt* が`">"`ならば，片側対立仮説 `mean (x) > m` を考慮する。同様に，`"<"`について，片側対立仮説 `mean (x) < m` を考慮する。標準設定は両側検定である。

その p-値は，*pval* に返る。

もし出力引数が与えられなければ，p-値を表示する。

25.3 モデル

[*theta*, *beta*, *dev*, *dl*, *d2l*, *p*] = logistic_regression (*y*, *x*, *print*, *theta*, *beta*) [Functio File]

通常のロジスティック回帰を実行する。

k 個の順序付きカテゴリの値をとる y を仮定し，gamma_i (*x*) は，共変量 x を与えたときに y が最初の i カテゴリのひとつに入る累積確率であるとする。このとき，

```
[theta, beta] = logistic_regression (y, x)
```

この関数は，以下のモデルを当てはめる。

```
logit (gamma_i (x)) = theta_i - beta' * x,   i = 1, ..., k-1
```

順序カテゴリの数 k は，round (y)の異なる値の数をとることになる。もし k が 2 と等しいならば，y は 2 値であり，そのモデルは通常のロジスティック回帰である。行列 x は全行階数であると仮定する。

y のみを与えると，theta = logistic_regression (y) は，baseline logit odds のみをもつモデルを当てはめる。

完全な形式は，以下のようである。

```
[theta, beta, dev, dl, d2l, gamma]
   = logistic_regression (y, x, print, theta, beta)
```

ここで，全ての出力引数，および y を除く全ての入力引数はオプションである。

print に 1 をセットすると，当てはめたモデルについての要約した情報を表示するようにする。*print* に 2 をセットすると，各反復における収束についての情報を要望する。他の値は，何も情報を表示しないようにする。入力引数 *theta* および *beta* は，*theta* と *beta* の初期推定値を与える。

戻り値 *dev* は，対数尤度をマイナス 2 倍した値を保持する。

戻り値 *dl* と *d2l* は，*theta* と *beta* についての対数尤度の 1 次導関数のベクトルと，2 次導関数の行列である。

p は x を与えるときの y の条件付き分布に対する推定値を保持する。

25.4 分布

beta_cdf (*x*, a, *b*) [Function File]

x の各要素について，パラメータが a と b であるベータ分布の，x における CDF（累積分布関数）を返す。すなわち，PROB (beta (a, b) <= x) である。

beta_inv (*x*, a, *b*) [Function File]

x の各要素について，パラメータが a と b であるベータ分布の，x における分位点（CDF の逆関数）を返す。

beta_pdf (*x*, a, *b*) [Function File]

x の各要素について，パラメータが a と b であるベータ分布の，x における PDF（確率密度関数）を返す。

`beta_rnd (a, b, r, c)` [Function File]
`beta_rnd (a, b, sz)` [Function File]
パラメータが a と b であるベータ分布からのランダムサンプルを含む r 行 c 列あるいは `size (sz)`の行列を返す。a と b はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。

もし r と c を省略すると，返される行列のサイズは，a および b と同様のサイズになる。

`binomial_cdf (x, n, p)` [Function File]
x の各要素について，パラメータが n と p である二項分布の，x における CDF（累積分布関数）を返す。

`binomial_inv (x, n, p)` [Function File]
x の各要素について，パラメータが n と p である二項分布の，x における分位点（CDF の逆関数）を返す。

`binomial_pdf (x, n, p)` [Function File]
x の各要素について，パラメータが n と p である二項分布の，x における PDF（確率密度関数）を返す。

`binomial_rnd (n, p, r, c)` [Function File]
`binomial_rnd (n, p, sz)` [Function File]
パラメータが n と p である二項分布からのランダムサンプルを含む r 行 c 列あるいは `size (sz)`の行列を返す。a と b はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。

もし r と c を省略すると，返される行列のサイズは，n および p と同様のサイズになる。

`cauchy_cdf (x, lambda, sigma)` [Function File]
x の各要素について，位置パラメータが *lambda* と尺度パラメータが *sigma* であるコーシー分布の x における CDF（累積分布関数）を返す。初期値は，$lambda = 0$ および $sigma = 1$ である。

`cauchy_inv (x, lambda, sigma)` [Function File]
x の各要素について，位置パラメータが *lambda* と尺度パラメータが *sigma* であるコーシー分布の x における分位点（CDF の逆関数）を返す。初期値は，$lambda = 0$ および $sigma = 1$ である。

`cauchy_pdf (x, lambda, sigma)` [Function File]
x の各要素について，位置パラメータが *lambda* と尺度パラメータが *sigma* > 0 であるコーシー分布の x における PDF（確率密度関数）を返す。初期値は，$lambda = 0$ および $sigma = 1$ である。

`cauchy_rnd (lambda, sigma, r, c)` [Function File]
`cauchy_rnd (lambda, sigma, sz)` [Function File]
位置パラメータが *lambda* と尺度パラメータが *sigma* > 0 であるコーシー分布からのランダムサンプルを含む r 行 c 列あるいは `size (sz)`の行列を返す。*lambda* と *sigma* はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。

もし r と c を省略すると，返される行列のサイズは，*lambda* および *sigma* と同様のサイズになる。

`chisquare_cdf (x, n)` [Function File]
x の各要素について，自由度が n であるカイ 2 乗分布の，x における CDF（累積分布関数）を返す。

`chisquare_inv (x, n)` [Function File]
x の各要素について，自由度が n であるカイ 2 乗分布の，x における分位点（CDF の逆関数）を返す。

`chisquare_pdf (x, n)` [Function File]
x の各要素について，自由度が n であるカイ 2 乗分布の，x における PDF（確率密度関数）を返す。

`chisquare_rnd (n, r, c)` [Function File]
`chisquare_rnd (n, sz)` [Function File]
自由度が n であるカイ 2 乗分布からのランダムサンプルを含む r 行 c 列あるいは `size (sz)` の行列を返す。n はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。

もし r と c を省略すると，返される行列のサイズは，n と同様のサイズになる。

`discrete_cdf (x, v, p)` [Function File]
x の各要素について，確率 p で v の値を仮定する単変量離散分布の，x における CDF（累積分布関数）を返す。

`discrete_inv (x, v, p)` [Function File]
x の各要素について，確率 p で v の値を仮定する単変量離散分布の，x における分位点（CDF の逆関数）を返す。

`discrete_pdf (x, v, p)` [Function File]
x の各要素について，確率 p で v の値を仮定する単変量離散分布の，x における PDF（確率密度関数）を返す。

`discrete_rnd (n, v, p)` [Function File]
`discrete_rnd (v, p, r, c)` [Function File]
`discrete_rnd (v, p, sz)` [Function File]
確率 p で v の値を仮定する単変量離散分布からのランダムサンプルを含む行列を生成する。

r と c が与えられると，r 行 c 列の行列を作る。あるいは sz がベクトルならば，サイズが sz の行列を作成する。

`empirical_cdf (x, data)` [Function File]
x の各要素について，単変量サンプル $data$ から得られる経験分布の，x における CDF（累積分布関数）を返す。

`empirical_inv (x, data)` [Function File]
x の各要素について，単変量サンプル $data$ から得られる経験分布の，x における分位点（CDF の逆関数）を返す。

`empirical_pdf (x, data)` [Function File]
x の各要素について，単変量サンプル *data* から得られる経験分布の，*x* における PDF（確率密度関数）を返す。

`empirical_rnd (n, data)` [Function File]
`empirical_rnd (data, r, c)` [Function File]
`empirical_rnd (data, sz)` [Function File]
単変量サンプル *data* から得られる経験分布からのランダムサンプルを含む行列を生成する。

r と *c* が与えられると，*r* 行 *c* 列の行列を作る。あるいは *sz* がベクトルならば，サイズが *sz* の行列を作成する。

`exponential_cdf (x, lambda)` [Function File]
x の各要素について，パラメータが *lambda* である指数分布の，*x* における CDF（累積分布関数）を返す。すなわち，

その引数は，共通のサイズまたはスカラをとることができる。

`exponential_inv (x, lambda)` [Function File]
x の各要素について，パラメータが *lambda* である指数分布の，*x* における分位点（CDF の逆関数）を返す。すなわち，

`exponential_pdf (x, lambda)` [Function File]
x の各要素について，パラメータが *lambda* である指数分布の，*x* における PDF（確率密度関数）を返す。すなわち，

`exponential_rnd (lambda, r, c)` [Function File]
`exponential_rnd (lambda, sz)` [Function File]
パラメータが *lambda* である指数分布からのランダムサンプルを含む *r* 行 *c* 列あるいは `size (sz)`の行列を返す。*a* と *b* はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。

もし *r* と *c* を省略すると，返される行列のサイズは，*lambda* と同様のサイズになる。

`f_cdf (x, m, n)` [Function File]
x の各要素について，自由度が *m* と *n* である F 分布の，*x* における CDF（累積分布関数）を返す。すなわち，PROB (F (*m*, *n*) <= *x*) である。

`f_inv (x, m, n)` [Function File]
x の各要素について，自由度が *m* と *n* である F 分布の，*x* における分位点（CDF の逆関数）を返す。すなわち，

`f_pdf (x, m, n)` [Function File]
x の各要素について，自由度が *m* と *n* である F 分布の，*x* における PDF（確率密度関数）を返す。

`f_rnd (m, n, r, c)` [Function File]
`f_rnd (m, n, sz)` [Function File]
自由度が m と n である F 分布からのランダムサンプルを含む r 行 c 列あるいは `size (sz)`の行列を返す。m と n はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。

もし r と c を省略すると，返される行列のサイズは，m および n と同様のサイズになる。

`gamma_cdf (x, a, b)` [Function File]
x の各要素について，パラメータが a と b であるガンマ分布の，x における CDF（累積分布関数）を返す。

`gamma_inv (x, a, b)` [Function File]
x の各要素について，パラメータが a と b であるガンマ分布の，x における分位点（CDF の逆関数）を返す。

`gamma_pdf (x, a, b)` [Function File]
x の各要素について，パラメータが a と b であるガンマ分布の，x における PDF（確率密度関数）を返す。

`gamma_rnd (a, b, r, c)` [Function File]
`gamma_rnd (a, b, sz)` [Function File]
パラメータが a と b であるガンマ分布からのランダムサンプルを含む r 行 c 列あるいは `size (sz)`の行列を返す。a と b はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。

もし r と c を省略すると，返される行列のサイズは，a および b と同様のサイズになる。

`geometric_cdf (x, p)` [Function File]
x の各要素について，パラメータが p である幾何分布の，x における CDF（累積分布関数）を返す。

`geometric_inv (x, p)` [Function File]
x の各要素について，パラメータが p である幾何分布の，x における分位点（CDF の逆関数）を返す。

`geometric_pdf (x, p)` [Function File]
x の各要素について，パラメータが p である幾何分布の，x における PDF（確率密度関数）を返す。

`geometric_rnd (p, r, c)` [Function File]
`geometric_rnd (p, sz)` [Function File]
パラメータが p である幾何分布からのランダムサンプルを含む r 行 c 列の行列を返す。a と b はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。

r と c が与えられると，r 行 c 列の行列を作る。あるいは sz がベクトルならば，サイズが sz の行列を作成する。

`hypergeometric_cdf (x, m, t, n)` [Function File]

x の各要素について，パラメータが m，t，n である超幾何分布の，x における CDF（累積分布関数）を計算する。これは，m 個のマーク付きアイテムを含む，総数 t 個の母集団から，置き換えなく n 個のサンプルを無作為に抽出するとき，x 個以上のマーク付きアイテムが得られない確率である。

パラメータ m，t，n は正の整数であり，m と n は，t より大きくてはいけない。

`hypergeometric_inv (x, m, t, n)` [Function File]

x の各要素について，パラメータが m，t，n である超幾何分布の，x における分位点（CDF の逆関数）を計算する。これは，m 個のマーク付きアイテムを含む，総数 t 個の母集団から，置き換えなく n 個のサンプルを無作為に抽出するとき，x 個以上のマーク付きアイテムが得られない確率である。

パラメータ m，t，n は正の整数であり，m と n は，t より大きくてはいけない。

`hypergeometric_pdf (x, m, t, n)` [Function File]

x の各要素について，パラメータが m，t，n である超幾何分布の，x における PDF（確率密度関数）を計算する。これは，m 個のマーク付きアイテムを含む，総数 t 個の母集団から，置き換えなく n 個のサンプルを無作為に抽出するとき，x 個以上のマーク付きアイテムが得られない確率である。

引数は同じサイズを持っているか，スカラでなければならない。

`hypergeometric_rnd (n_size, m, t, n)` [Function File]
`hypergeometric_rnd (m, t, n, r, c)` [Function File]
`hypergeometric_rnd (m, t, n, sz)` [Function File]

パラメータが m，t，n である超幾何分布から，サイズ n_size のランダムサンプルを含む列ベクトルを生成する。

r と c が与えられると，r 行 c 列の行列を作る。あるいは sz がベクトルならば，サイズが sz の行列を作成する。

パラメータ m，t，n は正の整数であり，m と n は，t より大きくてはいけない。

`kolmogorov_smirnov_cdf (x, tol)` [Function File]

Kolmogorov-Smirnov 分布の，x における CDF（累積分布関数）を計算する。この分布は，$x > 0$ について以下のようなものである。

$$Q(x) = \sum_{k=-\infty}^{\infty} (-1)^k \exp(-2k^2x^2)$$

オプション引数 tol は，この級数が評価されるべきまでの精度を指定する。初期値は $tol =$ `eps` である。

`laplace_cdf (x)` [Function File]

x の各要素について，ラプラス分布の，x における CDF（累積密度関数）を返す。

`laplace_inv (x)` [Function File]

x の各要素について，ラプラス分布の，x における分位点（CDF の逆関数）を返す。

`laplace_pdf (x)` [Function File]
x の各要素について，ラプラス分布の，x における PDF（確率密度関数）を返す。

`laplace_rnd (r, c)` [Function File]
`laplace_rnd (sz);` [Function File]
ラプラス分布からのランダムサンプルを含む r 行 c 列の行列を返す。あるいは sz がベクトルならば，サイズが sz の行列を作る。

`logistic_cdf (x)` [Function File]
x の各要素について，ロジスティック分布の，x における CDF（累積密度関数）を返す。

`logistic_inv (x)` [Function File]
x の各要素について，ロジスティック分布の，x における分位点（CDF の逆関数）を返す。

`logistic_pdf (x)` [Function File]
x の各要素について，ロジスティック分布の，x における PDF（確率密度関数）を返す。

`logistic_rnd (r, c)` [Function File]
`logistic_rnd (sz)` [Function File]
ロジスティック分布からのランダムサンプルを含む r 行 c 列の行列を返す。あるいは sz がベクトルならば，サイズが sz の行列を作る。

`lognormal_cdf (x, a, v)` [Function File]
x の各要素について，パラメータが a と v である対数正規分布の，x における CDF（累積分布関数）を返す。確率変数がこの分布に従うとき，その対数は平均が `log (a)`で分散が v の正規分布に従う。

初期値は $a = 1$ および $v = 1$ である。

`lognormal_inv (x, a, v)` [Function File]
x の各要素について，パラメータが a と v である対数正規分布の，x における分位点（CDF の逆関数）を返す。確率変数がこの分布に従うとき，その対数は平均が `log (a)`で分散が v の正規分布に従う。

初期値は $a = 1$ および $v = 1$ である。

`lognormal_pdf (x, a, v)` [Function File]
x の各要素について，パラメータが a と v である対数正規分布の，x における PDF（確率密度関数）を返す。確率変数がこの分布に従うとき，その対数は平均が `log (a)`で分散が v の正規分布に従う。

初期値は $a = 1$ および $v = 1$ である。

`lognormal_rnd (a, v, r, c)` [Function File]
`lognormal_rnd (a, v, sz)` [Function File]
パラメータが a と n である対数正規分布分布からのランダムサンプルを含む r 行 c 列の行列を返す。a と v はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。あるいは sz がベクトルならば，サイズが sz の行列を作る。

もし r と c を省略すると，返される行列のサイズは，a および v と同様のサイズになる。

`normal_cdf (x, m, v)` [Function File]

x の各要素について，平均が m で分散が v である正規分布の，x における CDF（累積分布関数）を返す。

初期値は $m = 1$ および $v = 1$ である。

`normal_inv (x, m, v)` [Function File]

x の各要素について，平均が m で分散が v である正規分布の，x における分位点（CDF の逆関数）を返す。

初期値は $m = 1$ および $v = 1$ である。

`normal_pdf (x, m, v)` [Function File]

x の各要素について，平均が m で分散が v である正規分布の，x における PDF（確率密度関数）を返す。

初期値は $m = 1$ および $v = 1$ である。

`normal_rnd (m, v, r, c)` [Function File]

`normal_rnd (m, v, sz)` [Function File]

パラメータが m と v である正規分布からのランダムサンプルを含む r 行 c 列あるいは `size (sz)`の行列を返す。m と v はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。

もし r と c を省略すると，返される行列のサイズは，m および v と同様のサイズになる。

`pascal_cdf (x, n, p)` [Function File]

x の各要素について，パラメータが n と p であるパスカル分布（負の二項分布）の，x における CDF（累積分布関数）を返す。

1 回の試行で成功する確率が p のベルヌーイ試行において，n 回目の成功までに失敗する回数は，この分布に従う。

`pascal_inv (x, n, p)` [Function File]

x の各要素について，パラメータが n と p であるパスカル分布（負の二項分布）の，x における分位点（CDF の逆関数）を返す。

1 回の試行で成功する確率が p のベルヌーイ試行において，n 回目の成功までに失敗する回数は，この分布に従う。

`pascal_pdf (x, n, p)` [Function File]

x の各要素について，パラメータが n と p であるパスカル分布（負の二項分布）の，x における PDF（確率密度関数）を返す。

1 回の試行で成功する確率が p のベルヌーイ試行において，n 回目の成功までに失敗する回数は，この分布に従う。

`pascal_rnd (n, p, r, c)` [Function File]

`pascal_rnd (n, p, sz)` [Function File]

パラメータが n と p であるパスカル分布（負の二項分布）からのランダムサンプルを含む r 行 c 列の行列を返す。n と p はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。

r と c が与えられると，r 行 c 列の行列を作る。あるいは sz がベクトルならば，サイズが sz の行列を作成する。

`poisson_cdf (x, lambda)` [Function File]
x の各要素について，パラメータが *lambda* であるポアソン分布の，x における CDF（累積分布関数）を返す。

`poisson_inv (x, lambda)` [Function File]
x の各要素について，パラメータが *lambda* であるポアソン分布の，x における分位点（CDF の逆関数）を返す。

`poisson_pdf (x, lambda)` [Function File]
x の各要素について，パラメータが *lambda* であるポアソン分布の，x における PDF（確率密度関数）を返す。

`poisson_rnd (lambda, r, c)` [Function File]
パラメータが *lambda* であるポアソン分布からのランダムサンプルを含む r 行 c 列の行列を返す。このパラメータはスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。
もし r と c を省略すると，返される行列のサイズは，*lambda* と同様のサイズになる。

`stdnormal_cdf (x)` [Function File]
x の各要素について，標準正規分布の，x における CDF（累積密度関数）を返す。

`stdnormal_inv (x)` [Function File]
x の各要素について，標準正規分布の，x における分位点（CDF の逆関数）を返す。

`stdnormal_pdf (x)` [Function File]
x の各要素について，標準正規分布の，x における PDF（確率密度関数）を返す。

`stdnormal_rnd (r, c)` [Function File]
`stdnormal_rnd (sz)` [Function File]
標準正規分布からのランダムサンプルを含む r 行 c 列あるいは `size (sz)` の行列を返す。

`t_cdf (x, n)` [Function File]
x の各要素について，自由度が n である t 分布の，x における CDF（累積分布関数）を返す。すなわち，PROB (t (n) <= x) である。

`t_inv (x, n)` [Function File]
x の各要素について，自由度が n である t（スチューデント）分布の，x における分位点（CDF の逆関数）を返す。

`t_pdf (x, n)` [Function File]
x の各要素について，自由度が n である t（スチューデント）分布の，x における PDF（確率密度関数）を返す。

`t_rnd (n, r, c)` [Function File]
`t_rnd (n, sz)` [Function File]
自由度が n である t（スチューデント）分布からのランダムサンプルを含む r 行 c 列の行列を返す。n はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。
もし r と c を省略すると，返される行列のサイズは，n と同様のサイズになる。

`uniform_cdf (x, a, b)` [Function File]
範囲が [a, b]] である一様分布の，x における CDF（累積密度関数）を返す。すなわち，PROB (uniform (a, b) <= x) である。

初期値は，$a = 0$ および $b = 1$ である。

`uniform_inv (x, a, b)` [Function File]
範囲が [a, b]] である一様分布の，x における分位点（CDF の逆関数）を返す。

初期値は，$a = 0$ および $b = 1$ である。

`uniform_pdf (x, a, b)` [Function File]
範囲が [a, b]] である一様分布の，x における PDF（確率密度関数）を返す。

初期値は，$a = 0$ および $b = 1$ である。

`uniform_rnd (a, b, r, c)` [Function File]
`uniform_rnd (a, b, sz)` [Function File]
範囲が [a, b]] である一様分布からのランダムサンプルを含む r 行 c 列あるいは `size (sz)`の行列を返す。a と b はスカラでなければならず，サイズ $r \times c$ もスカラとしなければならない。

もし r と c を省略すると，返される行列のサイズは，a および b と同様のサイズになる。

`weibull_cdf (x, alpha, sigma)` [Function File]
形状パラメータが *alpha* で尺度パラメータが *sigma* であるワイブル分布の，x における CDF（累積分布関数）を返す。この分布は，$x >= 0$ に対して，

```
1 - exp(-(x/sigma)^alpha)
```

である。

`weibull_inv (x, lambda, alpha)` [Function File]
形状パラメータが *alpha* で尺度パラメータが *sigma* であるワイブル分布の，x における分位点（CDF の逆関数）を返す。

`weibull_pdf (x, alpha, sigma)` [Function File]
形状パラメータが *alpha* で尺度パラメータが *sigma* であるワイブル分布の，x における PDF（確率密度関数）を返す。この分布は，$x >= 0$ に対して，

```
1 - exp(-(x/sigma)^alpha)
```

である。

`weibull_rnd (alpha, sigma, r, c)` [Function File]
`weibull_rnd (alpha, sigma, sz)` [Function File]
形状パラメータが *alpha* で尺度パラメータが *sigma* であるワイブル分布からのランダムサンプルを含む r 行 c 列の行列を返す。あるいは *sz* がベクトルならば，サイズが *sz* の行列を作る。

もし r と c を省略すると，返される行列のサイズは，*alpha* および *sigma* と同様のサイズになる。

`wiener_rnd (t, d, n)` [Function File]

範囲 $[0, t]$ における d 次元の Wiener Process のシミュレート実現値を返す。d を省略すると，$d = 1$ を使用する。返される行列の 1 列めには時間を含み，残りの列には Wiener process を含む。

オプション引数 n は，長さ 1 のインターバルにわたる過程をシミュレートするために使用する summand の数を与える。n を省略すると，$n = 1000$ が使用される。

26 Financial Functions

`fv (r, n, p, l, method)` [Function File]

Return the future value at the end of period n of an investment which consists of n payments of p in each period, assuming an interest rate r.

The optional argument l may be used to specify an additional lump-sum payment.

The optional argument *method* may be used ot specify whether the payments are made at the end (`"e"`, default) or at the beginning (`"b"`) of each period.

Note that the rate r is specified as a fraction (i.e., 0.05, not 5 percent).

`fvl (r, n, l)` [Function File]

Return the future value at the end of n periods of an initial lump sum investment l, given a per-period interest rate r.

Note that the rate r is specified as a fraction (i.e., 0.05, not 5 percent).

`irr (p, i)` [Function File]

Return the internal rate of return of a series of payments p from an initial investment i (i.e., the solution of `npv (r, p) = i`. If the second argument is omitted, a value of 0 is used.

`nper (r, p, a, l, method)` [Function File]

Return the number of regular payments of p necessary to amortize a loan of amount a and interest r.

The optional argument l may be used to specify an additional lump-sum payment of l made at the end of the amortization time.

The optional argument *method* may be used to specify whether payments are made at the end (`"e"`, default) or at the beginning (`"b"`) of each period.

Note that the rate r is specified as a fraction (i.e., 0.05, not 5 percent).

`npv (r, p, i)` [Function File]

Returns the net present value of a series of irregular (i.e., not necessarily identical) payments p which occur at the ends of n consecutive periods. r specifies the one-period interest rates and can either be a scalar (constant rates) or a vector of the same length as p.

The optional argument i may be used to specify an initial investment.

Note that the rate r is specified as a fraction (i.e., 0.05, not 5 percent).

`pmt (r, n, a, l, method)` [Function File]

Return the amount of periodic payment necessary to amortize a loan of amount a with interest rate r in n periods.

The optional argument l may be used to specify a terminal lump-sum payment.

The optional argument *method* may be used to specify whether payments are made at the end (`"e"`, default) or at the beginning (`"b"`) of each period.

`pv (r, n, p, l, method)` [Function File]

Returns the present value of an investment that will pay off p for n consecutive periods, assuming an interest r.

The optional argument l may be used to specify an additional lump-sum payment made at the end of n periods.

The optional argument *method* may be used to specify whether payments are made at the end (`"e"`, default) or at the beginning (`"b"`) of each period.

Note that the rate r is specified as a fraction (i.e., 0.05, not 5 percent).

`pvl (r, n, p)` [Function File]

Return the present value of an investment that will pay off p in one lump sum at the end of n periods, given the interest rate r.

Note that the rate r is specified as a fraction (i.e., 0.05, not 5 percent).

`rate (n, p, v, l, method)` [Function File]

Return the rate of return on an investment of present value v which pays p in n consecutive periods.

The optional argument l may be used to specify an additional lump-sum payment made at the end of n periods.

The optional string argument *method* may be used to specify whether payments are made at the end (`"e"`, default) or at the beginning (`"b"`) of each period.

`vol (x, m, n)` [Function File]

Return the volatility of each column of the input matrix x. The number of data sets per period is given by m (e.g. the number of data per year if you want to compute the volatility per year). The optional parameter n gives the number of past periods used for computation, if it is omitted, a value of 1 is used. If t is the number of rows of x, `vol` returns the volatility from `n*m` to t.

27 Sets

Octave has a limited set of functions for managing sets of data, where a set is defined as a collection unique elements.

`create_set (x)` [Function File]

Return a row vector containing the unique values in *x*, sorted in ascending order. For example,

```
create_set ([ 1, 2; 3, 4; 4, 2 ])
⇒ [ 1, 2, 3, 4 ]
```

`union (x, y)` [Function File]

Return the set of elements that are in either of the sets *x* and *y*. For example,

```
union ([ 1, 2, 4 ], [ 2, 3, 5 ])
⇒ [ 1, 2, 3, 4, 5 ]
```

`intersection (x, y)` [Function File]

Return the set of elements that are in both sets *x* and *y*. For example,

```
intersection ([ 1, 2, 3 ], [ 2, 3, 5 ])
⇒ [ 2, 3 ]
```

`complement (x, y)` [Function File]

Return the elements of set *y* that are not in set *x*. For example,

```
complement ([ 1, 2, 3 ], [ 2, 3, 5 ])
⇒ 5
```

28 Polynomial Manipulations

In Octave, a polynomial is represented by its coefficients (arranged in descending order). For example, a vector of length $N+1$ corresponds to the following polynomial of order N

$$p(x) = c_1 x^N + ... + c_N x + c_{N+1}.$$

`compan (c)` [Function File]

Compute the companion matrix corresponding to polynomial coefficient vector c.

The companion matrix is

$$A = \begin{bmatrix} -c_2/c_1 & -c_3/c_1 & \cdots & -c_N/c_1 & -c_{N+1}/c_1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}.$$

The eigenvalues of the companion matrix are equal to the roots of the polynomial.

`conv (a, b)` [Function File]

Convolve two vectors.

`y = conv (a, b)` returns a vector of length equal to `length (a) + length (b) - 1`. If a and b are polynomial coefficient vectors, `conv` returns the coefficients of the product polynomial.

`deconv (y, a)` [Function File]

Deconvolve two vectors.

`[b, r] = deconv (y, a)` solves for b and r such that `y = conv (a, b) + r`.

If y and a are polynomial coefficient vectors, b will contain the coefficients of the polynomial quotient and r will be a remander polynomial of lowest order.

`poly (a)` [Function File]

If a is a square N-by-N matrix, `poly (a)` is the row vector of the coefficients of `det (z * eye (N) - a)`, the characteristic polynomial of a. If x is a vector, `poly (x)` is a vector of coefficients of the polynomial whose roots are the elements of x.

`polyderiv (c)` [Function File]

Return the coefficients of the derivative of the polynomial whose coefficients are given by vector c.

`[p, s] = polyfit (x, y, n)` [Function File]

Return the coefficients of a polynomial $p(x)$ of degree n that minimizes

$$\sum_{i=1}^{N} (p(x_i) - y_i)^2$$

to best fit the data in the least squares sense.

The polynomial coefficients are returned in a row vector.

If two output arguments are requested, the second is a structure containing the following fields:

`R` The Cholesky factor of the Vandermonde matrix used to compute the polynomial coefficients.

`X` The Vandermonde matrix used to compute the polynomial coefficients.

`df` The degrees of freedom.

`normr` The norm of the residuals.

`yf` The values of the polynomial for each value of *x*.

`polyinteg (c)` [Function File]

Return the coefficients of the integral of the polynomial whose coefficients are represented by the vector *c*.

The constant of integration is set to zero.

`polyreduce (c)` [Function File]

Reduces a polynomial coefficient vector to a minimum number of terms by stripping off any leading zeros.

`polyval (c, x)` [Function File]

Evaluate a polynomial.

`polyval (c, x)` will evaluate the polynomial at the specified value of *x*.

If *x* is a vector or matrix, the polynomial is evaluated at each of the elements of *x*.

`polyvalm (c, x)` [Function File]

Evaluate a polynomial in the matrix sense.

`polyvalm (c, x)` will evaluate the polynomial in the matrix sense, i.e. matrix multiplication is used instead of element by element multiplication as is used in polyval.

The argument *x* must be a square matrix.

`residue (b, a, tol)` [Function File]

If *b* and *a* are vectors of polynomial coefficients, then residue calculates the partial fraction expansion corresponding to the ratio of the two polynomials.

The function `residue` returns *r*, *p*, *k*, and *e*, where the vector *r* contains the residue terms, *p* contains the pole values, *k* contains the coefficients of a direct polynomial term (if it exists) and *e* is a vector containing the powers of the denominators in the partial fraction terms.

Assuming *b* and *a* represent polynomials $P(s)$ and $Q(s)$ we have:

$$\frac{P(s)}{Q(s)} = \sum_{m=1}^{M} \frac{r_m}{(s - p_m)_m^e} + \sum_{i=1}^{N} k_i s^{N-i}.$$

where M is the number of poles (the length of the r, p, and e vectors) and N is the length of the k vector.

The argument *tol* is optional, and if not specified, a default value of 0.001 is assumed. The tolerance value is used to determine whether poles with small imaginary components are declared real. It is also used to determine if two poles are distinct. If the ratio of the imaginary part of a pole to the real part is less than *tol*, the imaginary part is discarded. If two poles are farther apart than *tol* they are distinct. For example,

```
b = [1, 1, 1];
a = [1, -5, 8, -4];
[r, p, k, e] = residue (b, a);
⇒ r = [-2, 7, 3]
⇒ p = [2, 2, 1]
⇒ k = [](0x0)
⇒ e = [1, 2, 1]
```

which implies the following partial fraction expansion

$$\frac{s^2 + s + 1}{s^3 - 5s^2 + 8s - 4} = \frac{-2}{s-2} + \frac{7}{(s-2)^2} + \frac{3}{s-1}$$

`roots (v)` [Function File]

For a vector v with N components, return the roots of the polynomial

$$v_1 z^{N-1} + \cdots + v_{N-1} z + v_N.$$

`polyout (c, x)` [Function File]

Write formatted polynomial

$$c(x) = c_1 x^n + \ldots + c_n x + c_{n+1}$$

and return it as a string or write it to the screen (if *nargout* is zero). x defaults to the string `"s"`.

29 Control Theory

The Octave Control Systems Toolbox (OCST) was initially developed by Dr. A. Scottedward Hodel `a.s.hodel@eng.auburn.edu` with the assistance of his students

- R. Bruce Tenison `btenison@dibbs.net`,
- David C. Clem,
- John E. Ingram `John.Ingram@sea.siemans.com`, and
- Kristi McGowan.

This development was supported in part by NASA's Marshall Space Flight Center as part of an in-house CACSD environment. Additional important contributions were made by Dr. Kai Mueller `mueller@ifr.ing.tu-bs.de` and Jose Daniel Munoz Frias (`place.m`).

An on-line menu-driven tutorial is available via `DEMOcontrol`; beginning OCST users should start with this program.

`DEMOcontrol` [Function File]

Octave Control Systems Toolbox demo/tutorial program. The demo allows the user to select among several categories of OCST function:

```
octave:1> DEMOcontrol
O C T A V E   C O N T R O L  S Y S T E M S  T O O L B O X
Octave Controls System Toolbox Demo

  [ 1] System representation
  [ 2] Block diagram manipulations
  [ 3] Frequency response functions
  [ 4] State space analysis functions
  [ 5] Root locus functions
  [ 6] LQG/H2/Hinfinity functions
  [ 7] End
```

Command examples are interactively run for users to observe the use of OCST functions.

29.1 System Data Structure

The OCST stores all dynamic systems in a single data structure format that can represent continuous systems, discrete-systems, and mixed (hybrid) systems in state-space form, and can also represent purely continuous/discrete systems in either transfer function or pole-zero form. In order to provide more flexibility in treatment of discrete/hybrid systems, the OCST also keeps a record of which system outputs are sampled.

Octave structures are accessed with a syntax much like that used by the C programming language. For consistency in use of the data structure used in the OCST, it is recommended that the system structure access m-files be used (see Section 29.2 [sysinterface], 頁 221). Some elements of the data structure are absent depending on the internal system representation(s) used. More than one system representation can be used for SISO systems; the OCST m-files ensure that all representations used are consistent with one another.

`sysrepdemo` [Function File]

Tutorial for the use of the system data structure functions.

29.1.1 Variables common to all OCST system formats

The data structure elements (and variable types) common to all system representations are listed below; examples of the initialization and use of the system data structures are given in subsequent sections and in the online demo `DEMOcontrol`.

n
nz The respective number of continuous and discrete states in the system (scalar)

inname
outname list of name(s) of the system input, output signal(s). (list of strings)

sys System status vector. (vector)

This vector indicates both what representation was used to initialize the system data structure (called the primary system type) and which other representations are currently up-to-date with the primary system type (see Section 29.2.5 [structaccess], 頁 227).

The value of the first element of the vector indicates the primary system type.

0 for tf form (initialized with `tf2sys` or `fir2sys`)

1 for zp form (initialized with `zp2sys`)

2 for ss form (initialized with `ss2sys`)

The next three elements are boolean flags that indicate whether tf, zp, or ss, respectively, are "up to date" (whether it is safe to use the variables associated with these representations). These flags are changed when calls are made to the `sysupdate` command.

tsam Discrete time sampling period (nonnegative scalar). *tsam* is set to 0 for continuous time systems.

yd Discrete-time output list (vector)

indicates which outputs are discrete time (i.e., produced by D/A converters) and which are continuous time. yd(ii) = 0 if output ii is continuous, = 1 if discrete.

The remaining variables of the system data structure are only present if the corresponding entry of the `sys` vector is true (=1).

29.1.2 `tf` format variables

num numerator coefficients (vector)

den denominator coefficients (vector)

29.1.3 `zp` format variables

zer system zeros (vector)

pol system poles (vector)

k leading coefficient (scalar)

29.1.4 `ss` format variables

a
b
c
d The usual state-space matrices. If a system has both continuous and discrete states, they are sorted so that continuous states come first, then discrete states

Note some functions (e.g., `bode`, `hinfsyn`) will not accept systems with both discrete and continuous states/outputs

stname names of system states (list of strings)

29.2 System Construction and Interface Functions

Construction and manipulations of the OCST system data structure (see Section 29.1 [sysstruct], 頁 219) requires attention to many details in order to ensure that data structure contents remain consistent. Users are strongly encouraged to use the system interface functions in this section. Functions for the formatted display in of system data structures are given in Section 29.3 [sysdisp], 頁 231.

29.2.1 Finite impulse response system interface functions

`fir2sys` (*num*, *tsam*, *inname*, *outname*) [Function File]

construct a system data structure from FIR description

Inputs

num vector of coefficients $[c_0, c_1, \ldots, c_n]$ of the SISO FIR transfer function

$$C(z) = c_0 + c_1 z^{-1} + c_2 z^{-2} + \ldots + c_n z^{-n}$$

tsam sampling time (default: 1)

inname name of input signal; may be a string or a list with a single entry.

outname name of output signal; may be a string or a list with a single entry.

Output

sys system data structure

Example

```
octave:1> sys = fir2sys([1 -1 2 4],0.342,\
> "A/D input","filter output");
octave:2> sysout(sys)
Input(s)
        1: A/D input

Output(s):
        1: filter output (discrete)

Sampling interval: 0.342
```

```
transfer function form:
1*z^3 - 1*z^2 + 2*z^1 + 4
-------------------------
1*z^3 + 0*z^2 + 0*z^1 + 0
```

`[c, tsam, input, output] = sys2fir (sys)` [Function File]
Extract FIR data from system data structure; see `fir2sys` for parameter descriptions.

29.2.2 State space system interface functions

`ss (a, b, c, d, tsam, n, nz, stname, inname, outname, outlist)` [Function File]
Create system structure from state-space data. May be continous, discrete, or mixed (sampled data)

Inputs

a
b
c
d usual state space matrices.
default: d = zero matrix

tsam sampling rate. Default: $tsam = 0$ (continuous system)

n
nz number of continuous, discrete states in the system
If *tsam* is 0, $n =$ `rows`(a), $nz = 0$.
If *tsam* is greater than zero, $n = 0$, $nz =$ `rows`(a)
see below for system partitioning

stname cell array of strings of state signal names
default (*stname*=[] on input): `x_n` for continuous states, `xd_n` for discrete states

inname cell array of strings of input signal names
default (*inname* = [] on input): `u_n`

outname cell array of strings of input signal names
default (*outname* = [] on input): `y_n`

outlist
list of indices of outputs y that are sampled
If *tsam* is 0, *outlist* = [].
If *tsam* is greater than 0, *outlist* $= 1 :$ `rows`(c).

Unlike states, discrete/continous outputs may appear in any order.

`sys2ss` returns a vector *yd* where *yd*(*outlist*) = 1; all other entries of *yd* are 0.

Outputs *outsys* = system data structure

System partitioning

Suppose for simplicity that outlist specified that the first several outputs were continuous and the remaining outputs were discrete. Then the system is partitioned as

```
x = [ xc ]  (n x 1)
    [ xd ]  (nz x 1 discrete states)
a = [ acc acd ]  b = [ bc ]
    [ adc add ]      [ bd ]
c = [ ccc ccd ]  d = [ dc ]
    [ cdc cdd ]      [ dd ]

    (cdc = c(outlist,1:n), etc.)
```

with dynamic equations:

$$\begin{aligned}
\frac{d}{dt}x_c(t) &= a_{cc}x_c(t) + a_{cd}x_d(k * t_{sam}) + bc * u(t) \\
x_d((k+1) * t_{sam}) &= a_{dc}x_c(kt_{sam}) + a_{dd}x_d(kt_{sam}) + b_d u(kt_{sam}) \\
y_c(t) &= c_{cc}x_c(t) + c_{cd}x_d(kt_{sam}) + d_c u(t) \\
y_d(kt_{sam}) &= c_{dc}x_c(kt_{sam}) + c_{dd}x_d(kt_{sam}) + d_d u(kt_{sam})
\end{aligned}$$

Signal partitions

	continuous	discrete
states	stname(1:n,:)	stname((n+1):(n+nz),:)
outputs	outname(cout,:)	outname(outlist,:)

where *cout* is the list of in 1:rows(p) that are not contained in outlist. (Discrete/continuous outputs may be entered in any order desired by the user.)

Example

```
octave:1> a = [1 2 3; 4 5 6; 7 8 10];
octave:2> b = [0 0 ; 0 1 ; 1 0];
octave:3> c = eye (3);
octave:4> sys = ss (a, b, c, [], 0, 3, 0, {"volts", "amps", "joules"});
octave:5> sysout(sys);
Input(s)
        1: u_1
        2: u_2

Output(s):
        1: y_1
        2: y_2
        3: y_3

state-space form:
3 continuous states, 0 discrete states
```

```
State(s):
        1: volts
        2: amps
        3: joules

A matrix: 3 x 3
   1   2   3
   4   5   6
   7   8  10
B matrix: 3 x 2
  0  0
  0  1
  1  0
C matrix: 3 x 3
  1  0  0
  0  1  0
  0  0  1
D matrix: 3 x 3
  0  0
  0  0
  0  0
```

Notice that the D matrix is constructed by default to the correct dimensions. Default input and output signals names were assigned since none were given.

`[a, b, c, d, tsam, n, nz, stname, inname, outname, yd] = sys2ss (sys)` [Function File]

Extract state space representation from system data structure.

Input

sys System data structure.

Outputs

a
b
c
d State space matrices for *sys*.

tsam Sampling time of *sys* (0 if continuous).

n
nz Number of continuous, discrete states (discrete states come last in state vector x).

stname
inname
outname Signal names (lists of strings); names of states, inputs, and outputs, respectively.

yd Binary vector; $yd(ii)$ is 1 if output $y(ii)$ is discrete (sampled); otherwise $yd(ii)$ is 0.

A warning massage is printed if the system is a mixed continuous and discrete system.

Example

```
octave:1> sys=tf2sys([1 2],[3 4 5]);
octave:2> [a,b,c,d] = sys2ss(sys)
a =
   0.00000   1.00000
  -1.66667  -1.33333
b =
  0
  1
c = 0.66667  0.33333
d = 0
```

29.2.3 Transfer function system interface functions

`tf2sys` (*num*, *den*, *tsam*, *inname*, *outname*) [Function File]

Build system data structure from transfer function format data.

Inputs

num
den Coefficients of numerator/denominator polynomials.

tsam Sampling interval; default: 0 (continuous time).

inname
outname Input/output signal names; may be a string or cell array with a single string entry.

Output

sys System data structure.

Example

```
octave:1> sys=tf2sys([2 1],[1 2 1],0.1);
octave:2> sysout(sys)
Input(s)
        1: u_1
Output(s):
        1: y_1 (discrete)
Sampling interval: 0.1
transfer function form:
2*z^1 + 1
-----------------
1*z^2 + 2*z^1 + 1
```

[*num*, *den*, *tsam*, *inname*, *outname*] = `sys2tf` (*sys*) [Function File]

Extract transfer function data from a system data structure.

See `tf` for parameter descriptions.

Example

```
octave:1> sys=ss([1 -2; -1.1,-2.1],[0;1],[1 1]);
octave:2> [num,den] = sys2tf(sys)
num = 1.0000  -3.0000
den = 1.0000   1.1000  -4.3000
```

29.2.4 Zero-pole system interface functions

zp2sys (*zer*, *pol*, *k*, *tsam*, *inname*, *outname*) [Function File]

Create system data structure from zero-pole data.

Inputs

zer	Vector of system zeros.
pol	Vector of system poles.
k	Scalar leading coefficient.
tsam	Sampling period; default: 0 (continuous system).
inname *outname*	Input/output signal names (lists of strings).

Output

sys	System data structure.

Example

```
octave:1> sys=zp2sys([1 -1],[-2 -2 0],1);
octave:2> sysout(sys)
Input(s)
        1: u_1
Output(s):
        1: y_1
zero-pole form:
1 (s - 1) (s + 1)
-----------------
s (s + 2) (s + 2)
```

[*zer*, *pol*, *k*, *tsam*, *inname*, *outname*] = sys2zp (*sys*) [Function File]

Extract zero/pole/leading coefficient information from a system data structure.

See zp for parameter descriptions.

Example

```
octave:1> sys=ss([1 -2; -1.1,-2.1],[0;1],[1 1]);
octave:2> [zer,pol,k] = sys2zp(sys)
zer = 3.0000
pol =
  -2.6953
   1.5953
k = 1
```

29.2.5 Data structure access functions

`syschnames (sys, opt, list, names)` [Function File]
Superseded by `syssetsignals`.

`syschtsam (sys, tsam)` [Function File]
This function changes the sampling time (tsam) of the system. Exits with an error if sys is purely continuous time.

`[n, nz, m, p, yd] = sysdimensions (sys, opt)` [Function File]
return the number of states, inputs, and/or outputs in the system *sys*.

Inputs

sys system data structure

opt String indicating which dimensions are desired. Values:

`"all"` (default) return all parameters as specified under Outputs below.

`"cst"` return *n*= number of continuous states

`"dst"` return *n*= number of discrete states

`"in"` return *n*= number of inputs

`"out"` return *n*= number of outputs

Outputs

n number of continuous states (or individual requested dimension as specified by *opt*).

nz number of discrete states

m number of system inputs

p number of system outputs

yd binary vector; $yd(ii)$ is nonzero if output *ii* is discrete. $yd(ii) = 0$ if output *ii* is continous

`[stname, inname, outname, yd] = sysgetsignals (sys)` [Function File]
`siglist = sysgetsignals (sys, sigid)` [Function File]
`signame = sysgetsignals (sys, sigid, signum, strflg)` [Function File]
Get signal names from a system

Inputs

sys system data structure for the state space system

sigid signal id. String. Must be one of

`"in"` input signals

`"out"` output signals

	`"st"`	stage signals
	`"yd"`	value of logical vector *yd*

signum index(indices) or name(s) or signals; see `sysidx`

strflg flag to return a string instead of a cell array; Values:

`0`	(default) return a cell array (even if signum specifies an individual signal)
`1`	return a string. Exits with an error if signum does not specify an individual signal.

Outputs

•If *sigid* is not specified:

stname
inname
outname signal names (cell array of strings); names of states, inputs, and outputs, respectively.

yd binary vector; *yd*(*ii*) is nonzero if output *ii* is discrete.

•If *sigid* is specified but *signum* is not specified:

`sigid="in"`
siglist is set to the cell array of input names.

`sigid="out"`
siglist is set to the cell array of output names.

`sigid="st"`
siglist is set to the cell array of state names.

stage signals

`sigid="yd"`
siglist is set to logical vector indicating discrete outputs; *siglist*(*ii*) = 0 indicates that output *ii* is continuous (unsampled), otherwise it is discrete.

•If the first three input arguments are specified:
signame is a cell array of the specified signal names (*sigid* is `"in"`, `"out"`, or `"st"`), or else the logical flag indicating whether output(s) *signum* is(are) discrete (*sigval*=1) or continuous (*sigval*=0).

Examples (From `sysrepdemo`)

```
octave> sys=ss(rand(4),rand(4,2),rand(3,4));
octave># get all signal names
octave> [Ast,Ain,Aout,Ayd] = sysgetsignals(sys)
Ast =
(
  [1] = x_1
  [2] = x_2
  [3] = x_3
```

```
  [4] = x_4
)
Ain =
(
  [1] = u_1
  [2] = u_2
)
Aout =
(
  [1] = y_1
  [2] = y_2
  [3] = y_3
)
Ayd =

  0  0  0
octave> # get only input signal names:
octave> Ain = sysgetsignals(sys,"in")
Ain =
(
  [1] = u_1
  [2] = u_2
)
octave> # get name of output 2 (in cell array):
octave> Aout = sysgetsignals(sys,"out",2)
Aout =
(
  [1] = y_2
)
octave> # get name of output 2 (as string):
octave> Aout = sysgetsignals(sys,"out",2,1)
Aout = y_2
```

`sysgettype` (*sys*) [Function File]

return the initial system type of the system

Input

sys System data structure.

Output

systype String indicating how the structure was initially constructed. Values: `"ss"`, `"zp"`, or `"tf"`.

FIR initialized systems return `systype="tf"`.

`syssetsignals` (*sys*, *opt*, *names*, *sig_idx*) [Function File]

change the names of selected inputs, outputs and states.

Inputs

sys System data structure.

opt Change default name (output).

`"out"` Change selected output names.

`"in"` Change selected input names.

`"st"` Change selected state names.

`"yd"` Change selected outputs from discrete to continuous or from continuous to discrete.

names

`opt = "out", "in", "st"`
string or string array containing desired signal names or values.

`opt = "yd"`
To desired output continuous/discrete flag. Set name to 0 for continuous, or 1 for discrete.

sig_idx indices or names of outputs, yd, inputs, or states whose respective names/values should be changed.
Default: replace entire cell array of names/entire yd vector.

Outputs

retsys *sys* with appropriate signal names changed (or *yd* values, where appropriate).

Example

```
octave:1> sys=ss([1 2; 3 4],[5;6],[7 8]);
octave:2> sys = syssetsignals(sys,"st",str2mat("Posx","Velx"));
octave:3> sysout(sys)
Input(s)
        1: u_1
Output(s):
        1: y_1
state-space form:
2 continuous states, 0 discrete states
State(s):
        1: Posx
        2: Velx
A matrix: 2 x 2
  1  2
  3  4
B matrix: 2 x 1
  5
  6
C matrix: 1 x 2
  7  8
D matrix: 1 x 1
0
```

`sysupdate (sys, opt)` [Function File]

Update the internal representation of a system.

Inputs

sys: system data structure

opt string:

`"tf"`	update transfer function form
`"zp"`	update zero-pole form
`"ss"`	update state space form
`"all"`	all of the above

Outputs

retsys Contains union of data in sys and requested data. If requested data in *sys* is already up to date then *retsys=sys*.

Conversion to `tf` or `zp` exits with an error if the system is mixed continuous/digital.

`[systype, nout, nin, ncstates, ndstates] = minfo (inmat)` [Function File]

Determines the type of system matrix. *inmat* can be a varying, a system, a constant, and an empty matrix.

Outputs

systype	Can be one of: varying, system, constant, and empty.
nout	The number of outputs of the system.
nin	The number of inputs of the system.
ncstates	The number of continuous states of the system.
ndstates	The number of discrete states of the system.

`sysgettsam (sys)` [Function File]

Return the sampling time of the system *sys*.

29.2.6 Data structure internal functions

29.3 System display functions

`sysout (sys, opt)` [Function File]

print out a system data structure in desired format

sys system data structure

opt Display option

`[]`	primary system form (default)
`"ss"`	state space form

`"tf"`	transfer function form
`"zp"`	zero-pole form
`"all"`	all of the above

`tfout (num, denom, x)` [Function File]
Print formatted transfer function $n(s)/d(s)$ to the screen. x defaults to the string `"s"`

`zpout (zer, pol, k, x)` [Function File]
print formatted zero-pole form to the screen. x defaults to the string `"s"`

29.4 Block Diagram Manipulations

See Section 29.7 [systime], 頁 249.

Unless otherwise noted, all parameters (input,output) are system data structures.

`bddemo (inputs)` [Function File]
Octave Controls toolbox demo: Block Diagram Manipulations demo.

`buildssic (clst, ulst, olst, ilst, s1, s2, s3, s4, s5, s6, s7, s8)` [Function File]
Form an arbitrary complex (open or closed loop) system in state-space form from several systems. `buildssic` can easily (despite its cryptic syntax) integrate transfer functions from a complex block diagram into a single system with one call. This function is especially useful for building open loop interconnections for $\mathcal{H}_\infty$ and $\mathcal{H}_2$ designs or for closing loops with these controllers.

Although this function is general purpose, the use of `sysgroup` `sysmult`, `sysconnect` and the like is recommended for standard operations since they can handle mixed discrete and continuous systems and also the names of inputs, outputs, and states.

The parameters consist of 4 lists that describe the connections outputs and inputs and up to 8 systems *s1–s8*. Format of the lists:

clst connection list, describes the input signal of each system. The maximum number of rows of Clst is equal to the sum of all inputs of s1-s8.

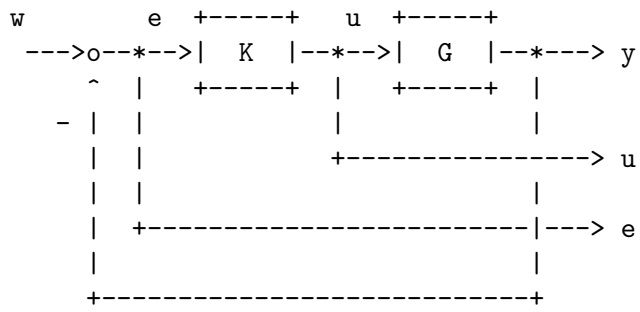
Example: `[1 2 -1; 2 1 0]` means that: new input 1 is old input 1 + output 2 - output 1, and new input 2 is old input 2 + output 1. The order of rows is arbitrary.

ulst if not empty the old inputs in vector *ulst* will be appended to the outputs. You need this if you want to "pull out" the input of a system. Elements are input numbers of *s1–s8*.

olst output list, specifiy the outputs of the resulting systems. Elements are output numbers of *s1–s8*. The numbers are allowed to be negative and may appear in any order. An empty matrix means all outputs.

ilst input list, specifiy the inputs of the resulting systems. Elements are input numbers of *s1–s8*. The numbers are allowed to be negative and may appear in any order. An empty matrix means all inputs.

Example: Very simple closed loop system.



The closed loop system GW can be optained by

```
GW = buildssic([1 2; 2 -1], 2, [1 2 3], 2, G, K);
```

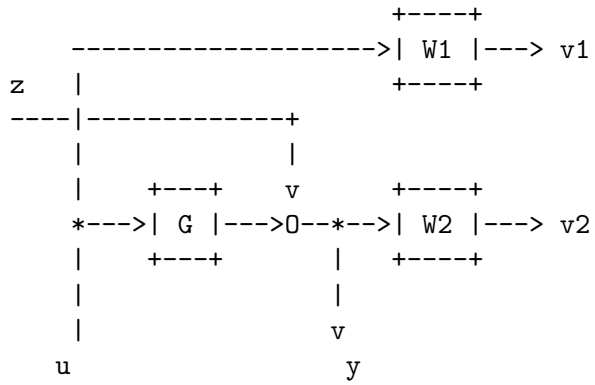
clst 1st row: connect input 1 (G) with output 2 (K).
2nd row: connect input 2 (K) with negative output 1 (G).

ulst Append input of 2 (K) to the number of outputs.

olst Outputs are output of 1 (G), 2 (K) and appended output 3 (from *ulst*).

ilst The only input is 2 (K).

Here is a real example:



$$\min \|GW_{vz}\|_\infty$$

The closed loop system GW from $[z, u]^T$ to $[v_1, v_2, y]^T$ can be obtained by (all SISO systems):

```
GW = buildssic([1, 4; 2, 4; 3, 1], 3, [2, 3, 5],
               [3, 4], G, W1, W2, One);
```

where "One" is a unity gain (auxillary) function with order 0. (e.g. `One = ugain(1);`)

sys = `jet707 ()` [Function File]

Creates a linearized state-space model of a Boeing 707-321 aircraft at v=80 m/s ($M = 0.26$, $G_{a0} = -3°$, $\alpha_0 = 4°$, $\kappa = 50°$).

System inputs: (1) thrust and (2) elevator angle.

System outputs: (1) airspeed and (2) pitch angle.

Reference: R. Brockhaus: *Flugregelung* (Flight Control), Springer, 1994.

`ord2` (*`nfreq`*, *`damp`*, *`gain`*) [Function File]

Creates a continuous 2nd order system with parameters:

Inputs

nfreq natural frequency [Hz]. (not in rad/s)

damp damping coefficient

gain dc-gain This is steady state value only for damp > 0. gain is assumed to be 1.0 if ommitted.

Output

outsys system data structure has representation with $w = 2\pi f$:

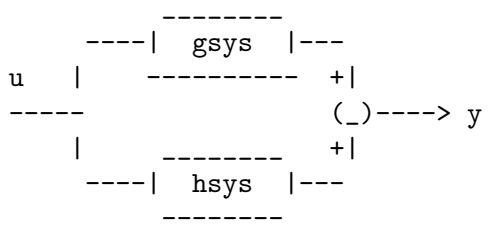
```
                /                                        \
                | / -2w*damp -w \  / w \                 |
            G = | |             |, |   |, [ 0  gain ], 0 |
                | \   w       0 /  \ 0 /                 |
                \                                        /
```

See also `jet707` (MIMO example, Boeing 707-321 aircraft model)

`sysadd` (*`gsys`*, *`hsys`*) [Function File]

returns *sys* = *gsys* + *hsys*.

- Exits with an error if *gsys* and *hsys* are not compatibly dimensioned.
- Prints a warning message is system states have identical names; duplicate names are given a suffix to make them unique.
- *sys* input/output names are taken from *gsys*.



`sys` = `sysappend` (*`syst`*, *`b`*, *`c`*, *`d`*, *`outname`*, *`inname`*, *`yd`*) [Function File]

appends new inputs and/or outputs to a system

Inputs

syst system data structure

b matrix to be appended to sys "B" matrix (empty if none)

c matrix to be appended to sys "C" matrix (empty if none)

d revised sys d matrix (can be passed as [] if the revised d is all zeros)

outname list of names for new outputs

inname list of names for new inputs

yd binary vector; $yd(ii) = 0$ indicates a continuous output; $yd(ii) = 1$ indicates a discrete output.

Outputs

sys

```
sys.b := [syst.b , b]
sys.c := [syst.c  ]
         [ c     ]
sys.d := [syst.d | D12 ]
         [ D21   | D22 ]
```

where $D12$, $D21$, and $D22$ are the appropriate dimensioned blocks of the input parameter d.

- The leading block $D11$ of d is ignored.
- If *inname* and *outname* are not given as arguments, the new inputs and outputs are be assigned default names.
- *yd* is a binary vector of length rows(c) that indicates continuous/sampled outputs. Default value for *yd* is:
  - *sys* is continuous or mixed *yd* = `zeros(1,rows(c))`
  - *sys* is discrete *yd* = `ones(1,rows(c))`

clsys = sysconnect (*sys*, *out_idx*, *in_idx*, *order*, *tol*) [Function File]

Close the loop from specified outputs to respective specified inputs

Inputs

sys System data structure.

out_idx
in_idx Names or indices of signals to connect (see `sysidx`). The output specified by $out_idx(ii)$ is connected to the input specified by $in_idx(ii)$.

order logical flag (default = 0)

`0`	Leave inputs and outputs in their original order.
`1`	Permute inputs and outputs to the order shown in the diagram below.

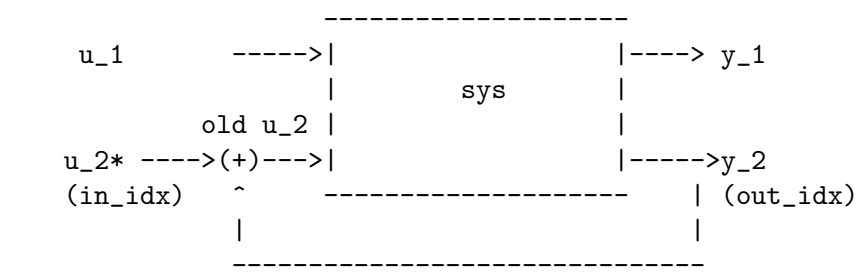
tol Tolerance for singularities in algebraic loops, default: 200`eps`.

Outputs

clsys Resulting closed loop system.

Method

`sysconnect` internally permutes selected inputs, outputs as shown below, closes the loop, and then permutes inputs and outputs back to their original order



The input that has the summing junction added to it has an * added to the end of the input name.

`[csys, acd, ccd] = syscont (sys)` [Function File]

Extract the purely continuous subsystem of an input system.

Input

sys system data structure.

Outputs

csys is the purely continuous input/output connections of *sys*

acd
ccd connections from discrete states to continuous states, discrete states to continuous outputs, respectively.

returns *csys* empty if no continuous/continous path exists

`[dsys, adc, cdc] = sysdisc (sys)` [Function File]

Input

sys System data structure.

Outputs

dsys Purely discrete portion of sys (returned empty if there is no purely discrete path from inputs to outputs).

adc
cdc Connections from continuous states to discrete states and discrete. outputs, respectively.

`retsys = sysdup (asys, out_idx, in_idx)` [Function File]

Duplicate specified input/output connections of a system

Inputs

asys system data structure

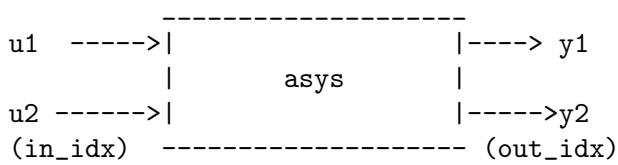
out_idx
in_idx indices or names of desired signals (see `sigidx`). duplicates are made of `y(out_idx(ii))` and `u(in_idx(ii))`.

Output

retsys Resulting closed loop system: duplicated i/o names are appended with a "+" suffix.

Method

`sysdup` creates copies of selected inputs and outputs as shown below. *u1*, *y1* is the set of original inputs/outputs, and *u2*, *y2* is the set of duplicated inputs/outputs in the order specified in *in_idx*, *out_idx*, respectively



sys = sysgroup (*asys*, *bsys*) [Function File]

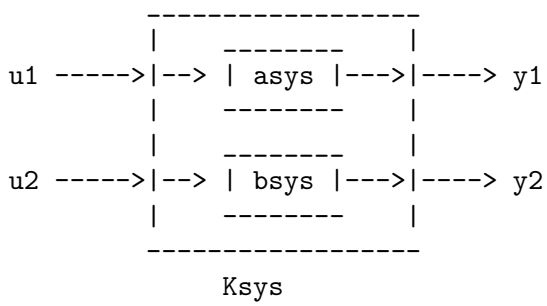
Combines two systems into a single system.

Inputs

asys
bsys System data structures.

Output

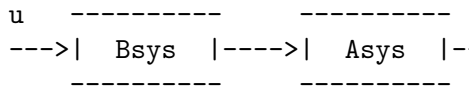
sys $sys = \text{blockdiag}(asys, bsys)$



The function also rearranges the internal state-space realization of *sys* so that the continuous states come first and the discrete states come last. If there are duplicate names, the second name has a unique suffix appended on to the end of the name.

sys = sysmult (*Asys*, *Bsys*) [Function File]

Compute $sys = Asys * Bsys$ (series connection):



A warning occurs if there is direct feed-through from an input or a continuous state of *Bsys*, through a discrete output of *Bsys*, to a continuous state or output in *Asys* (system data structure does not recognize discrete inputs).

retsys = sysprune (*asys*, *out_idx*, *in_idx*) [Function File]

Extract specified inputs/outputs from a system

Inputs

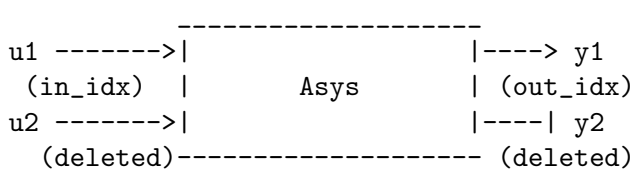
asys system data structure

out_idx
in_idx Indices or signal names of the outputs and inputs to be kept in the returned system; remaining connections are "pruned" off. May select as [] (empty matrix) to specify all outputs/inputs.

```
retsys = sysprune (Asys, [1:3,4], "u_1");
retsys = sysprune (Asys, {"tx", "ty", "tz"}, 4);
```

Output

retsys Resulting system.



pv = sysreorder (*vlen*, *list*) [Function File]

Inputs

vlen Vector length.

list A subset of `[1:vlen]`.

Output

pv A permutation vector to order elements of `[1:vlen]` in `list` to the end of a vector.

Used internally by `sysconnect` to permute vector elements to their desired locations.

retsys = sysscale (*sys*, *outscale*, *inscale*, *outname*, *inname*) [Function File]

scale inputs/outputs of a system.

Inputs

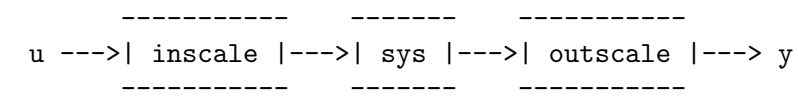
sys Structured system.

outscale
inscale Constant matrices of appropriate dimension.

outname
inname Lists of strings with the names of respectively outputs and inputs.

Output

retsys resulting open loop system:



If the input names and output names (each a list of strings) are not given and the scaling matrices are not square, then default names will be given to the inputs and/or outputs.

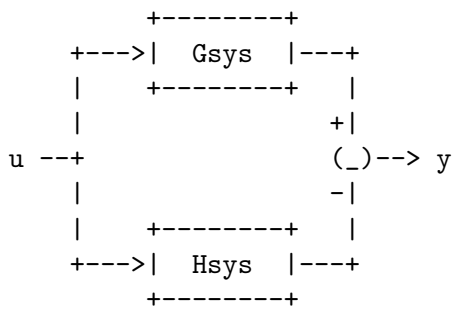
A warning message is printed if outscale attempts to add continuous system outputs to discrete system outputs; otherwise *yd* is set appropriately in the returned value of *sys*.

sys = syssub (*Gsys*, *Hsys*) [Function File]

Return $sys = Gsys - Hsys$.

Method

Gsys and *Hsys* are connected in parallel. The input vector is connected to both systems; the outputs are subtracted. Returned system names are those of *Gsys*.



ugain (*n*) [Function File]

Creates a system with unity gain, no states. This trivial system is sometimes needed to create arbitrary complex systems from simple systems with `buildssic`. Watch out if you are forming sampled systems since `ugain` does not contain a sampling period.

W = wgt1o (*vl*, *vh*, *fc*) [Function File]

State space description of a first order weighting function.

Weighting function are needed by the $\mathcal{H}_2/\mathcal{H}_\infty$ design procedure. These function are part of the augmented plant P (see `hinfdemo` for an application example).

Inputs

vl Gain at low frequencies.

vh Gain at high frequencies.

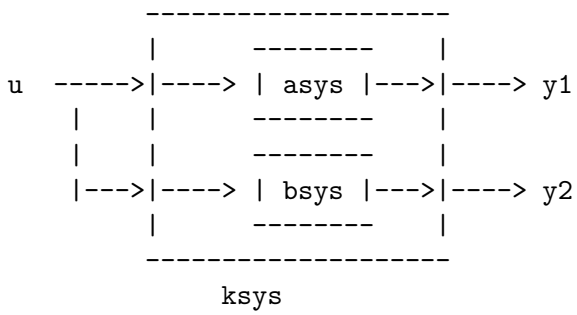
fc Corner frequency (in Hz, **not** in rad/sec)

Output

W Weighting function, given in form of a system data structure.

ksys = parallel (*asys*, *bsys*) [Function File]

Forms the parallel connection of two systems.



`[retsys, nc, no] = sysmin (sys, flg)` [Function File]

Returns a minimal (or reduced order) system

Inputs

sys System data structure

flg When equal to 0 (default value), returns minimal system, in which state names are lost; when equal to 1, returns system with physical states removed that are either uncontrollable or unobservable (cannot reduce further without discarding physical meaning of states).

Outputs

retsys Returned system.

nc Number of controllable states in the returned system.

no Number of observable states in the returned system.

cflg `is_controllable(retsys)`.

oflg `is_observable(retsys)`.

29.5 Numerical Functions

`x = are (a, b, c, opt)` [Function File]

Solve the Algebraic Riccati Equation

$$A^T X + XA - XBX + C = 0$$

Inputs for identically dimensioned square matrices

a n by n matrix;

b n by n matrix or n by m matrix; in the latter case b is replaced by $b := b * b'$;

c n by n matrix or p by m matrix; in the latter case c is replaced by $c := c' * c$;

opt (optional argument; default = `"B"`): String option passed to `balance` prior to ordered Schur decomposition.

Output

x	solution of the ARE.

Method Laub's Schur method (IEEE Transactions on Automatic Control, 1979) is applied to the appropriate Hamiltonian matrix.

`x = dare (a, b, q, r, opt)` [Function File]

Return the solution, x of the discrete-time algebraic Riccati equation

$$A^T X A - X + A^T X B (R + B^T X B)^{-1} B^T X A + Q = 0$$

Inputs

a	n by n matrix;
b	n by m matrix;
q	n by n matrix, symmetric positive semidefinite, or a p by n matrix, In the latter case $q := q' * q$ is used;
r	m by m, symmetric positive definite (invertible);
opt	(optional argument; default = `"B"`): String option passed to `balance` prior to ordered QZ decomposition.

Output

x	solution of DARE.

Method Generalized eigenvalue approach (Van Dooren; SIAM J. Sci. Stat. Comput., Vol 2) applied to the appropriate symplectic pencil.

See also: Ran and Rodman, *Stable Hermitian Solutions of Discrete Algebraic Riccati Equations*, Mathematics of Control, Signals and Systems, Vol 5, no 2 (1992), pp 165–194.

`[tvals, plist] = dre (sys, q, r, qf, t0, tf, ptol, maxits)` [Function File]

Solve the differential Riccati equation

$$-\frac{dP}{dt} = A^T P + PA - PBR^{-1}B^T P + Q$$

$$P(t_f) = Q_f$$

for the LTI system sys. Solution of standard LTI state feedback optimization

$$\min \int_{t_0}^{t_f} x^T Q x + u^T R u dt + x(t_f)^T Q_f x(t_f)$$

optimal input is

$$u = -R^{-1} B^T P(t) x$$

Inputs

sys	continuous time system data structure
q	state integral penalty

r	input integral penalty
qf	state terminal penalty
t0 *tf*	limits on the integral
ptol	tolerance (used to select time samples; see below); default = 0.1
maxits	number of refinement iterations (default=10)

Outputs

tvals	time values at which $p(t)$ is computed
plist	list values of $p(t)$; *plist* `{ i }` is $p(tvals(i))$

tvals is selected so that:

$$\|plist_i - plist_{i-1}\| < ptol$$

for every i between 2 and length($tvals$).

`dgram (a, b)` [Function File]

Return controllability gramian of discrete time system

$$x_{k+1} = ax_k + bu_k$$

Inputs

a	n by n matrix
b	n by m matrix

Output

m	n by n matrix, satisfies

$$ama^T - m + bb^T = 0$$

`dlyap (a, b)` [Function File]

Solve the discrete-time Lyapunov equation

Inputs

a	n by n matrix;
b	Matrix: n by n, n by m, or p by n.

Output

x	matrix satisfying appropriate discrete time Lyapunov equation.

Options:

- b is square: solve
$$axa^T - x + b = 0$$
- b is not square: x satisfies either
$$axa^T - x + bb^T = 0$$
or
$$a^T xa - x + b^T b = 0,$$
whichever is appropriate.

Method Uses Schur decomposition method as in Kitagawa, *An Algorithm for Solving the Matrix Equation $X = FXF' + S$*, International Journal of Control, Volume 25, Number 5, pages 745–753 (1977).

Column-by-column solution method as suggested in Hammarling, *Numerical Solution of the Stable, Non-Negative Definite Lyapunov Equation*, IMA Journal of Numerical Analysis, Volume 2, pages 303–323 (1982).

`gram (a, b)` [Function File]

Return controllability gramian m of the continuous time system $dx/dt = ax + bu$.

m satisfies $am + ma' + bb' = 0$.

`lyap (a, b, c)` [Function File]
`lyap (a, b)` [Function File]

Solve the Lyapunov (or Sylvester) equation via the Bartels-Stewart algorithm (Communications of the ACM, 1972).

If a, b, and c are specified, then `lyap` returns the solution of the Sylvester equation
$$AX + XB + C = 0$$

If only `(a, b)` are specified, then `lyap` returns the solution of the Lyapunov equation
$$A^T X + XA + B = 0$$

If b is not square, then `lyap` returns the solution of either
$$A^T X + XA + B^T B = 0$$
or
$$AX + XA^T + BB^T = 0$$
whichever is appropriate.

Solves by using the Bartels-Stewart algorithm (1972).

`qzval (a, b)` [Function File]

Compute generalized eigenvalues of the matrix pencil $(A - \lambda B)$.

a and b must be real matrices.

`qzval` is obsolete; use `qz` instead.

`y = zgfmul (a, b, c, d, x)` [Function File]
Compute product of *zgep* incidence matrix F with vector x. Used by `zgepbal` (in `zgscal`) as part of generalized conjugate gradient iteration.

`zgfslv (n, m, p, b)` [Function File]
Solve system of equations for dense zgep problem.

`zz = zginit (a, b, c, d)` [Function File]
Construct right hand side vector *zz* for the zero-computation generalized eigenvalue problem balancing procedure. Called by `zgepbal`.

`zgreduce (sys, meps)` [Function File]
Implementation of procedure REDUCE in (Emami-Naeini and Van Dooren, Automatica, # 1982).

`[nonz, zer] = zgrownorm (mat, meps)` [Function File]
Return *nonz* = number of rows of *mat* whose two norm exceeds *meps*, and *zer* = number of rows of mat whose two norm is less than *meps*.

`x = zgscal (f, z, n, m, p)` [Function File]
Generalized conjugate gradient iteration to solve zero-computation generalized eigenvalue problem balancing equation $fx = z$; called by `zgepbal`.

`[a, b] = zgsgiv (c, s, a, b)` [Function File]
Apply givens rotation c,s to row vectors a, b. No longer used in zero-balancing (__zgpbal__); kept for backward compatibility.

`x = zgshsr (y)` [Function File]
Apply householder vector based on e^m to column vector y. Called by `zgfslv`.

References

ZGEP Hodel, *Computation of Zeros with Balancing*, 1992, Linear Algebra and its Applications

Generalized CG
Golub and Van Loan, *Matrix Computations, 2nd ed* 1989.

29.6 System Analysis-Properties

`analdemo ()` [Function File]
Octave Controls toolbox demo: State Space analysis demo

`[n, m, p] = abcddim (a, b, c, d)` [Function File]
Check for compatibility of the dimensions of the matrices defining the linear system $[A, B, C, D]$ corresponding to

$$\frac{dx}{dt} = Ax + Bu$$
$$y = Cx + Du$$

or a similar discrete-time system.

If the matrices are compatibly dimensioned, then `abcddim` returns

n	The number of system states.
m	The number of system inputs.
p	The number of system outputs.

Otherwise `abcddim` returns $n = m = p = -1$.

Note: n = 0 (pure gain block) is returned without warning.

`ctrb (sys, b)` [Function File]
`ctrb (a, b)` [Function File]

Build controllability matrix:

$$Q_s = [BABA^2B \dots A^{n-1}B]$$

of a system data structure or the pair (a, b).

`ctrb` forms the controllability matrix. The numerical properties of `is_controllable` are much better for controllability tests.

`h2norm (sys)` [Function File]

Computes the $\mathcal{H}_2$ norm of a system data structure (continuous time only).

Reference: Doyle, Glover, Khargonekar, Francis, *State-Space Solutions to Standard $\mathcal{H}_2$ and $\mathcal{H}_\infty$ Control Problems*, IEEE TAC August 1989.

`[g, gmin, gmax] = hinfnorm (sys, tol, gmin, gmax, ptol)` [Function File]

Computes the $\mathcal{H}_\infty$ norm of a system data structure.

Inputs

sys	system data structure
tol	$\mathcal{H}_\infty$ norm search tolerance (default: 0.001)
gmin	minimum value for norm search (default: 1e-9)
gmax	maximum value for norm search (default: 1e+9)
ptol	pole tolerance: • if sys is continuous, poles with $\lvert\text{real}(pole)\rvert < ptol\|H\|$ (H is appropriate Hamiltonian) are considered to be on the imaginary axis. • if sys is discrete, poles with $\lvert\text{pole} - 1\rvert < ptol\|[s_1 s_2]\|$ (appropriate symplectic pencil) are considered to be on the unit circle. • Default value: 1e-9

Outputs

g	Computed gain, within *tol* of actual gain. *g* is returned as Inf if the system is unstable.

gmin
gmax Actual system gain lies in the interval [*gmin*, *gmax*].

References: Doyle, Glover, Khargonekar, Francis, *State-space solutions to standard $\mathcal{H}_2$ and $\mathcal{H}_\infty$ control problems*, IEEE TAC August 1989; Iglesias and Glover, *State-Space approach to discrete-time $\mathcal{H}_\infty$ control*, Int. J. Control, vol 54, no. 5, 1991; Zhou, Doyle, Glover, *Robust and Optimal Control*, Prentice-Hall, 1996.

`obsv (`*`sys`*`, c)` [Function File]
`obsv (a, c)` [Function File]
Build observability matrix:

$$Q_b = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

of a system data structure or the pair (a, c).

The numerical properties of `is_observable` are much better for observability tests.

`[`*`zer, pol`*`] = pzmap (`*`sys`*`)` [Function File]
Plots the zeros and poles of a system in the complex plane.

Input

sys System data structure.

Outputs

pol

zer if omitted, the poles and zeros are plotted on the screen. otherwise, *pol* and *zer* are returned as the system poles and zeros (see `sys2zp` for a preferable function call).

`retval` `= is_abcd (a, b, c, d)` [Function File]
Returns *retval* = 1 if the dimensions of a, b, c, d are compatible, otherwise *retval* = 0 with an appropriate diagnostic message printed to the screen. The matrices b, c, or d may be omitted.

`[`*`retval, u`*`] = is_controllable (`*`sys, tol`*`)` [Function File]
`[`*`retval, u`*`] = is_controllable (a, b, `*`tol`*`)` [Function File]
Logical check for system controllability.

Inputs

sys system data structure

a
b n by n, n by m matrices, respectively

tol optional roundoff paramter. default value: `10*eps`

Outputs

retval Logical flag; returns true (1) if the system *sys* or the pair (*a*, *b*) is controllable, whichever was passed as input arguments.

u *u* is an orthogonal basis of the controllable subspace.

Method Controllability is determined by applying Arnoldi iteration with complete re-orthogonalization to obtain an orthogonal basis of the Krylov subspace

```
span ([b,a*b,...,a^{n-1}*b]).
```

The Arnoldi iteration is executed with `krylov` if the system has a single input; otherwise a block Arnoldi iteration is performed with `krylovb`.

retval = `is_detectable` (*a*, *c*, *tol*, *dflg*) [Function File]
retval = `is_detectable` (*sys*, *tol*) [Function File]

Test for detactability (observability of unstable modes) of (*a*, *c*).

Returns 1 if the system *a* or the pair (*a*, *c*) is detectable, 0 if not, and -1 if the system has unobservable modes at the imaginary axis (unit circle for discrete-time systems).

See `is_stabilizable` for detailed description of arguments and computational method.

[*retval*, *dgkf_struct*] = `is_dgkf` (*asys*, *nu*, *ny*, *tol*) [Function File]

Determine whether a continuous time state space system meets assumptions of DGKF algorithm. Partitions system into:

```
[dx/dt]   [A  | Bw  Bu  ][w]
[ z   ] = [Cz | Dzw Dzu ][u]
[ y   ]   [Cy | Dyw Dyu ]
```

or similar discrete-time system. If necessary, orthogonal transformations *qw*, *qz* and nonsingular transformations *ru*, *ry* are applied to respective vectors *w*, *z*, *u*, *y* in order to satisfy DGKF assumptions. Loop shifting is used if *dyu* block is nonzero.

Inputs

asys system data structure

nu number of controlled inputs

ny number of measured outputs

tol threshold for 0; default: 200*`eps`.

Outputs

retval true(1) if system passes check, false(0) otherwise

dgkf_struct
data structure of `is_dgkf` results. Entries:

nw
nz dimensions of *w*, *z*

a system *A* matrix

bw	(n x nw) qw-transformed disturbance input matrix
bu	(n x nu) ru-transformed controlled input matrix; $B = [BwBu]$
cz	(nz x n) Qz-transformed error output matrix
cy	(ny x n) ry-transformed measured output matrix $C = [Cz; Cy]$
dzu	
dyw	off-diagonal blocks of transformed system D matrix that enter z, y from u, w respectively
ru	controlled input transformation matrix
ry	observed output transformation matrix
dyu_nz	nonzero if the *dyu* block is nonzero.
dyu	untransformed *dyu* block
dflg	nonzero if the system is discrete-time

`is_dgkf` exits with an error if the system is mixed discrete/continuous.

References

[1] Doyle, Glover, Khargonekar, Francis, *State Space Solutions to Standard $\mathcal{H}_2$ and $\mathcal{H}_\infty$ Control Problems*, IEEE TAC August 1989.

[2] Maciejowksi, J.M., *Multivariable Feedback Design*, Addison-Wesley, 1989.

digital = `is_digital (`*sys*, *eflg*`)` [Function File]

Return nonzero if system is digital.

Inputs

sys	System data structure.
eflg	When equal to 0 (default value), exits with an error if the system is mixed (continuous and discrete components); when equal to 1, print a warning if the system is mixed (continuous and discrete); when equal to 2, operate silently.

Output

digital	When equal to 0, the system is purely continuous; when equal to 1, the system is purely discrete; when equal to -1, the system is mixed continuous and discrete.

Exits with an error if *sys* is a mixed (continuous and discrete) system.

`[retval, u] = is_observable (a, c, tol)` [Function File]
`[retval, u] = is_observable (sys, tol)` [Function File]

Logical check for system observability.

Default: tol = `tol = 10*norm(a,'fro')*eps`

Returns 1 if the system *sys* or the pair (*a*, *c*) is observable, 0 if not.

See `is_controllable` for detailed description of arguments and default values.

`is_sample (ts)` [Function File]

Return true if *ts* is a valid sampling time (real, scalar, > 0).

`is_siso (sys)` [Function File]

Returns nonzero if the system data structure *sys* is single-input, single-output.

`retval = is_stabilizable (sys, tol)` [Function File]
`retval = is_stabilizable (a, b, tol, dflg)` [Function File]

Logical check for system stabilizability (i.e., all unstable modes are controllable). Returns 1 if the system is stabilizable, 0 if the the system is not stabilizable, -1 if the system has non stabilizable modes at the imaginary axis (unit circle for discrete-time systems.

Test for stabilizability is performed via Hautus Lemma. If *dflg*≠0 assume that discrete-time matrices (a,b) are supplied.

`is_signal_list (mylist)` [Function File]

Return true if *mylist* is a list of individual strings.

`is_stable (a, tol, dflg)` [Function File]
`is_stable (sys, tol)` [Function File]

Returns 1 if the matrix *a* or the system *sys* is stable, or 0 if not.

Inputs

tol	is a roundoff parameter, set to 200*`eps` if omitted.
dflg	Digital system flag (not required for system data structure):
	`dflg != 0` stable if eig(a) is in the unit circle
	`dflg == 0` stable if eig(a) is in the open LHP (default)

29.7 System Analysis-Time Domain

`c2d (sys, opt, t)` [Function File]
`c2d (sys, t)` [Function File]

Converts the system data structure describing:

$$\dot{x} = A_c x + B_c u$$

into a discrete time equivalent model:

$$x_{n+1} = A_d x_n + B_d u_n$$

via the matrix exponential or bilinear transform.

Inputs

sys system data structure (may have both continuous time and discrete time subsystems)

opt string argument; conversion option (optional argument; may be omitted as shown above)

`"ex"` use the matrix exponential (default)

`"bi"` use the bilinear transformation

$$s = \frac{2(z-1)}{T(z+1)}$$

FIXME: This option exits with an error if *sys* is not purely continuous. (The `ex` option can handle mixed systems.)

`"matched"`
Use the matched pole/zero equivalent transformation (currently only works for purely continuous SISO systems).

t sampling time; required if *sys* is purely continuous.

Note: if the second argument is not a string, `c2d()` assumes that the second argument is *t* and performs appropriate argument checks.

Output

dsys Discrete time equivalent via zero-order hold, sample each *t* sec.

This function adds the suffix `_d` to the names of the new discrete states.

`d2c (`*`sys`*`, `*`tol`*`)` [Function File]
`d2c (`*`sys`*`, `*`opt`*`)` [Function File]

Convert a discrete (sub)system into a purely continuous one. The sampling time used is `sysgettsam(`*`sys`*`)`.

Inputs

sys system data structure with discrete components

tol Scalar value. Tolerance for convergence of default `"log"` option (see below)

opt conversion option. Choose from:

`"log"` (default) Conversion is performed via a matrix logarithm. Due to some problems with this computation, it is followed by a steepest descent algorithm to identify continuous time *a*, *b*, to get a better fit to the original data.

If called as `d2c (`*`sys`*`, `*`tol`*`)`, with *tol* positive scalar, the `"log"` option is used. The default value for *tol* is `1e-8`.

`"bi"` Conversion is performed via bilinear transform $z = (1 + sT/2)/(1 - sT/2)$ where T is the system sampling time (see `sysgettsam`).

FIXME: bilinear option exits with an error if *sys* is not purely discrete

Output

csys continuous time system (same dimensions and signal names as in *sys*).

`[dsys, fidx] = dmr2d (sys, idx, sprefix, ts2, cuflg)` [Function File]

convert a multirate digital system to a single rate digital system states specified by *idx*, *sprefix* are sampled at *ts2*, all others are assumed sampled at *ts1* = `sysgettsam (sys)`.

Inputs

sys discrete time system; `dmr2d` exits with an error if *sys* is not discrete

idx indices or names of states with sampling time `sysgettsam(sys)` (may be empty); see `cellidx`

sprefix list of string prefixes of states with sampling time `sysgettsam(sys)` (may be empty)

ts2 sampling time of states not specified by *idx*, *sprefix* must be an integer multiple of `sysgettsam(sys)`

cuflg `"constant u flag"` if *cuflg* is nonzero then the system inputs are assumed to be constant over the revised sampling interval *ts2*. Otherwise, since the inputs can change during the interval t in $[kts2, (k+1)ts2]$, an additional set of inputs is included in the revised B matrix so that these intersample inputs may be included in the single-rate system. default *cuflg* = 1.

Outputs

dsys equivalent discrete time system with sampling time *ts2*.

The sampling time of sys is updated to *ts2*.

if *cuflg*=0 then a set of additional inputs is added to the system with suffixes _d1, ..., _dn to indicate their delay from the starting time k *ts2*, i.e. u = [u_1; u_1_d1; ..., u_1_dn] where u_1_dk is the input k*ts1 units of time after u_1 is sampled. (*ts1* is the original sampling time of the discrete time system and *ts2* = (n+1)*ts1)

fidx indices of `"formerly fast"` states specified by *idx* and *sprefix*; these states are updated to the new (slower) sampling interval *ts2*.

WARNING Not thoroughly tested yet; especially when *cuflg* == 0.

`damp (p, tsam)` [Function File]

Displays eigenvalues, natural frequencies and damping ratios of the eigenvalues of a matrix p or the A matrix of a system p, respectively. If p is a system, *tsam* must not be specified. If p is a matrix and *tsam* is specified, eigenvalues of p are assumed to be in z-domain.

`dcgain (sys, tol)` [Function File]

Returns dc-gain matrix. If dc-gain is infinite an empty matrix is returned. The argument *tol* is an optional tolerance for the condition number of the A Matrix in *sys* (default *tol* = 1.0e-10)

`[y, t] = impulse (sys, inp, tstop, n)` [Function File]
Impulse response for a linear system. The system can be discrete or multivariable (or both). If no output arguments are specified, `impulse` produces a plot or the impulse response data for system *sys*.

Inputs

sys System data structure.

inp Index of input being excited

tstop The argument *tstop* (scalar value) denotes the time when the simulation should end.

n the number of data values.

Both parameters *tstop* and *n* can be omitted and will be computed from the eigenvalues of the A Matrix.

Outputs

y Values of the impulse response.

t Times of the impulse response.

`[y, t] = step (sys, inp, tstop, n)` [Function File]
Step response for a linear system. The system can be discrete or multivariable (or both). If no output arguments are specified, `step` produces a plot or the step response data for system *sys*.

Inputs

sys System data structure.

inp Index of input being excited

tstop The argument *tstop* (scalar value) denotes the time when the simulation should end.

n the number of data values.

Both parameters *tstop* and *n* can be omitted and will be computed from the eigenvalues of the A Matrix.

Outputs

y Values of the step response.

t Times of the step response.

When invoked with the output parameter *y* the plot is not displayed.

29.8 System Analysis-Frequency Domain

Demonstration/tutorial script

`frdemo ()` [Function File]

Octave Control Toolbox demo: Frequency Response demo.

`[mag, phase, w] = bode (sys, w, out_idx, in_idx)` [Function File]

If no output arguments are given: produce Bode plots of a system; otherwise, compute the frequency response of a system data structure

Inputs

sys a system data structure (must be either purely continuous or discrete; see is_digital)

w frequency values for evaluation.

if *sys* is continuous, then bode evaluates $G(jw)$ where $G(s)$ is the system transfer function.

if *sys* is discrete, then bode evaluates G(`exp`(jwT)), where

- T is the system sampling time
- $G(z)$ is the system transfer function.

Default the default frequency range is selected as follows: (These steps are **not** performed if *w* is specified)

1. via routine __bodquist__, isolate all poles and zeros away from w=0 (jw=0 or `exp`(jwT)=1) and select the frequency range based on the breakpoint locations of the frequencies.
2. if *sys* is discrete time, the frequency range is limited to jwT in $[0, 2\pi/T]$
3. A "smoothing" routine is used to ensure that the plot phase does not change excessively from point to point and that singular points (e.g., crossovers from +/- 180) are accurately shown.

out_idx
in_idx

The names or indices of outputs and inputs to be used in the frequency response. See `sysprune`.

Example

```
bode(sys,[],"y_3", {"u_1","u_4"});
```

Outputs

mag
phase the magnitude and phase of the frequency response $G(jw)$ or $G($`exp`$(jwT))$ at the selected frequency values.

w the vector of frequency values used

1. If no output arguments are given, e.g.,

```
bode(sys);
```

bode plots the results to the screen. Descriptive labels are automatically placed.

Failure to include a concluding semicolon will yield some garbage being printed to the screen (`ans = []`).

2. If the requested plot is for an MIMO system, mag is set to $||G(jw)||$ or $||G(\texttt{exp}(jwT))||$ and phase information is not computed.

`[wmin, wmax] = bode_bounds (zer, pol, dflg, tsam)` [Function File]

Get default range of frequencies based on cutoff frequencies of system poles and zeros. Frequency range is the interval $[10^{w_{min}}, 10^{w_{max}}]$

Used internally in `__freqresp__` (`bode`, `nyquist`)

`freqchkw (w)` [Function File]

Used by `__freqresp__` to check that input frequency vector w is valid. Returns boolean value.

`out = ltifr (a, b, w)` [Function File]
`out = ltifr (sys, w)` [Function File]

Linear time invariant frequency response of single-input systems.

Inputs

a	
b	coefficient matrices of $dx/dt = Ax + Bu$
sys	system data structure
w	vector of frequencies

Output

out	frequency response, that is:

$$G(j\omega) = (j\omega I - A)^{-1}B$$

for complex frequencies $s = jw$.

`[realp, imagp, w] = nyquist (sys, w, out_idx, in_idx, atol)` [Function File]
`nyquist (sys, w, out_idx, in_idx, atol)` [Function File]

Produce Nyquist plots of a system; if no output arguments are given, Nyquist plot is printed to the screen.

Compute the frequency response of a system.

Inputs (pass as empty to get default values)

sys	system data structure (must be either purely continuous or discrete; see `is_digital`)
w	frequency values for evaluation. If sys is continuous, then bode evaluates $G(jw)$; if sys is discrete, then bode evaluates $G(exp(jwT))$, where T is the system sampling time.

default the default frequency range is selected as follows: (These steps are **not** performed if *w* is specified)

1. via routine `__bodquist__`, isolate all poles and zeros away from w=0 (jw=0 or $exp(jwT) = 1$) and select the frequency range based on the breakpoint locations of the frequencies.
2. if *sys* is discrete time, the frequency range is limited to jwT in $[0, 2p\pi]$
3. A "smoothing" routine is used to ensure that the plot phase does not change excessively from point to point and that singular points (e.g., crossovers from +/- 180) are accurately shown.

atol for interactive nyquist plots: atol is a change-in-slope tolerance for the of asymptotes (default = 0; 1e-2 is a good choice). This allows the user to "zoom in" on portions of the Nyquist plot too small to be seen with large asymptotes.

Outputs

realp
imagp the real and imaginary parts of the frequency response $G(jw)$ or $G(exp(jwT))$ at the selected frequency values.

w the vector of frequency values used

If no output arguments are given, nyquist plots the results to the screen. If *atol* != 0 and asymptotes are detected then the user is asked interactively if they wish to zoom in (remove asymptotes) Descriptive labels are automatically placed.

Note: if the requested plot is for an MIMO system, a warning message is presented; the returned information is of the magnitude $\|G(jw)\|$ or $\|G(\exp(jwT)\|$ only; phase information is not computed.

`[zer, gain] = tzero (a, b, c, d, opt)` [Function File]
`[zer, gain] = tzero (sys, opt)` [Function File]

Compute transmission zeros of a continuous system:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

or of a discrete one:

$$x_{k+1} = Ax_k + Bu_k$$

$$y_k = Cx_k + Du_k$$

Outputs

zer transmission zeros of the system

gain leading coefficient (pole-zero form) of SISO transfer function returns gain=0 if system is multivariable

References

1. Emami-Naeini and Van Dooren, Automatica, 1982.
2. Hodel, *Computation of Zeros with Balancing*, 1992 Lin. Alg. Appl.

`zr = tzero2 (a, b, c, d, bal)` [Function File]

Compute the transmission zeros of a, b, c, d.

bal = balancing option (see balance); default is `"B"`.

Needs to incorporate `mvzero` algorithm to isolate finite zeros; use `tzero` instead.

29.9 Controller Design

`dgkfdemo ()` [Function File]

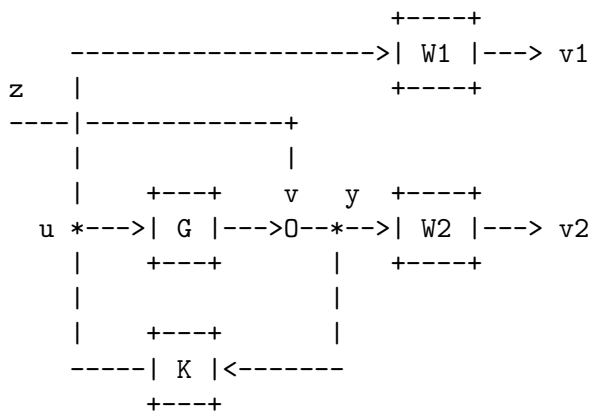
Octave Controls toolbox demo: $\mathcal{H}_2/\mathcal{H}_\infty$ options demos.

`hinfdemo ()` [Function File]

$\mathcal{H}_\infty$ design demos for continuous SISO and MIMO systems and a discrete system. The SISO system is difficult to control because it is non-minimum-phase and unstable. The second design example controls the `jet707` plant, the linearized state space model of a Boeing 707-321 aircraft at v=80 m/s ($M = 0.26$, $G_{a0} = -3°$, $\alpha_0 = 4°$, $\kappa = 50°$). Inputs: (1) thrust and (2) elevator angle Outputs: (1) airspeed and (2) pitch angle. The discrete system is a stable and second order.

SISO plant:

$$G(s) = \frac{s-2}{(s+2)(s-1)}$$

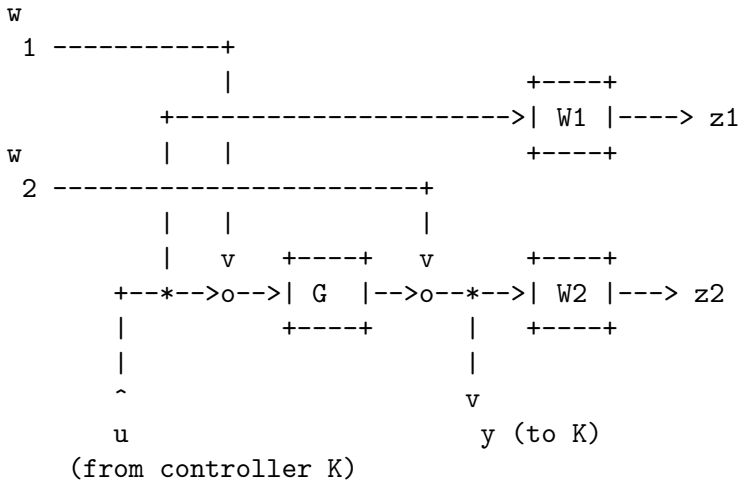


$$\min \|T_{vz}\|_\infty$$

W1 und *W2* are the robustness and performance weighting functions.

MIMO plant:

The optimal controller minimizes the $\mathcal{H}_\infty$ norm of the augmented plant P (mixed-sensitivity problem):



$$\begin{bmatrix} z_1 \\ z_2 \\ y \end{bmatrix} = P \begin{bmatrix} w_1 \\ w_2 \\ u \end{bmatrix}$$

Discrete system:

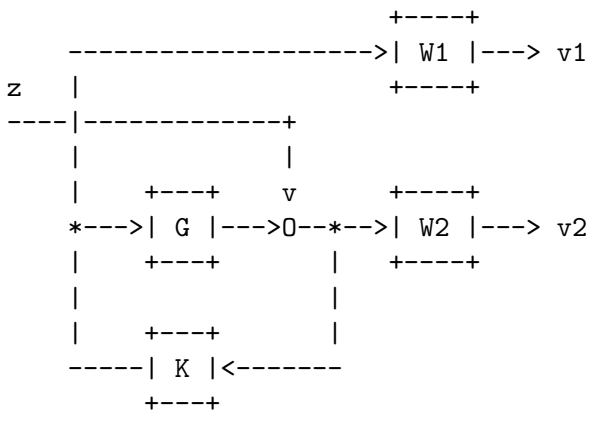
This is not a true discrete design. The design is carried out in continuous time while the effect of sampling is described by a bilinear transformation of the sampled system. This method works quite well if the sampling period is "small" compared to the plant time constants.

The continuous plant:

$$G(s) = \frac{1}{(s+2)(s+1)}$$

is discretised with a ZOH (Sampling period = $Ts = 1$ second):

$$G(z) = \frac{0.199788z + 0.073498}{(z - 0.36788)(z - 0.13534)}$$



$$\min \|T_{vz}\|_\infty$$

W1 and *W2* are the robustness and performance weighting functions.

`[l, m, p, e] = dlqe (a, g, c, sigw, sigv, z)` [Function File]
Construct the linear quadratic estimator (Kalman filter) for the discrete time system

$$x_{k+1} = Ax_k + Bu_k + Gw_k$$

$$y_k = Cx_k + Du_k + v_k$$

where w, v are zero-mean gaussian noise processes with respective intensities *`sigw`* `= cov (w, w)` and *`sigv`* `= cov (v, v)`.

If specified, z is `cov (w, v)`. Otherwise `cov (w, v) = 0`.

The observer structure is

$$z_{k|k} = z_{k|k-1} + l(y_k - Cz_{k|k-1} - Du_k)$$

$$z_{k+1|k} = Az_{k|k} + Bu_k$$

The following values are returned:

l The observer gain, $(A - ALC)$. is stable.

m The Riccati equation solution.

p The estimate error covariance after the measurement update.

e The closed loop poles of $(A - ALC)$.

`[k, p, e] = dlqr (a, b, q, r, z)` [Function File]
Construct the linear quadratic regulator for the discrete time system

$$x_{k+1} = Ax_k + Bu_k$$

to minimize the cost functional

$$J = \sum x^T Qx + u^T Ru$$

z omitted or

$$J = \sum x^T Qx + u^T Ru + 2x^T Zu$$

z included.

The following values are returned:

k The state feedback gain, $(A - BK)$ is stable.

p The solution of algebraic Riccati equation.

e The closed loop poles of $(A - BK)$.

`[Lp, Lf, P, Z] = dkalman (A, G, C, Qw, Rv, S)` [Function File]

Construct the linear quadratic estimator (Kalman predictor) for the discrete time system

$$x_{k+1} = Ax_k + Bu_k + Gw_k$$

$$y_k = Cx_k + Du_k + v_k$$

where w, v are zero-mean gaussian noise processes with respective intensities `Qw = cov (w, w)` and `Rv = cov (v, v)`.

If specified, S is `cov (w, v)`. Otherwise `cov (w, v) = 0`.

The observer structure is $x_{k+1|k} = Ax_{k|k-1} + Bu_k + L_p(y_k - Cx_{k|k-1} - Du_k)$ $x_{k|k} = x_{k|k} + L_f(y_k - Cx_{k|k-1} - Du_k)$

The following values are returned:

Lp The predictor gain, $(A - L_pC)$. is stable.

Lf The filter gain.

P The Riccati solution. $P = E\{(x - x_{n|n-1})(x - x_{n|n-1})'\}$

Z The updated error covariance matrix. $Z = E\{(x - x_{n|n})(x - x_{n|n})'\}$

`[K, gain, kc, kf, pc, pf] = h2syn (asys, nu, ny, tol)` [Function File]

Design $\mathcal{H}_2$ optimal controller per procedure in Doyle, Glover, Khargonekar, Francis, *State-Space Solutions to Standard $\mathcal{H}_2$ and $\mathcal{H}_\infty$ Control Problems*, IEEE TAC August 1989.

Discrete-time control per Zhou, Doyle, and Glover, *Robust and optimal control*, Prentice-Hall, 1996.

Inputs

asys system data structure (see ss, sys2ss)

- controller is implemented for continuous time systems
- controller is **not** implemented for discrete time systems

nu number of controlled inputs

ny number of measured outputs

tol threshold for 0. Default: 200*`eps`

Outputs

k system controller

gain optimal closed loop gain

kc full information control (packed)

kf state estimator (packed)

pc ARE solution matrix for regulator subproblem

pf ARE solution matrix for filter subproblem

`K = hinf_ctr (dgs, f, h, z, g)` [Function File]

Called by `hinfsyn` to compute the $\mathcal{H}_\infty$ optimal controller.

Inputs

dgs	data structure returned by `is_dgkf`
f	
h	feedback and filter gain (not partitioned)
g	final gamma value

Outputs

K	controller (system data structure)

Do not attempt to use this at home; no argument checking performed.

`[k, g, gw, xinf, yinf] = hinfsyn (asys, nu, ny, gmin, gmax, gtol, ptol, tol)` [Function File]

Inputs input system is passed as either

asys	system data structure (see `ss`, `sys2ss`) • controller is implemented for continuous time systems • controller is **not** implemented for discrete time systems (see bilinear transforms in `c2d`, `d2c`)
nu	number of controlled inputs
ny	number of measured outputs
gmin	initial lower bound on $\mathcal{H}_\infty$ optimal gain
gmax	initial upper bound on $\mathcal{H}_\infty$ Optimal gain.
gtol	Gain threshold. Routine quits when *gmax*/*gmin* < 1+tol.
ptol	poles with `abs(real(pole))` $<$ *ptol*$\|\|H\|\|$ (H is appropriate Hamiltonian) are considered to be on the imaginary axis. Default: 1e-9.
tol	threshold for 0. Default: 200*`eps`. *gmax*, *min*, *tol*, and *tol* must all be postive scalars.

Outputs

k	System controller.
g	Designed gain value.
gw	Closed loop system.
xinf	ARE solution matrix for regulator subproblem.
yinf	ARE solution matrix for filter subproblem.

References:

1. Doyle, Glover, Khargonekar, Francis, *State-Space Solutions to Standard $\mathcal{H}_2$ and $\mathcal{H}_\infty$ Control Problems*, IEEE TAC August 1989.

2. Maciejowksi, J.M., *Multivariable feedback design*, Addison-Wesley, 1989, ISBN 0-201-18243-2.
3. Keith Glover and John C. Doyle, *State-space formulae for all stabilizing controllers that satisfy an $\mathcal{H}_\infty$norm bound and relations to risk sensitivity*, Systems & Control Letters 11, Oct. 1988, pp 167–172.

`[retval, pc, pf] = hinfsyn_chk (a, b1, b2, c1, c2, d12, d21, g, ptol)` [Function File]

Called by `hinfsyn` to see if gain g satisfies conditions in Theorem 3 of Doyle, Glover, Khargonekar, Francis, *State Space Solutions to Standard $\mathcal{H}_2$ and $\mathcal{H}_\infty$ Control Problems*, IEEE TAC August 1989.

Warning: do not attempt to use this at home; no argument checking performed.

Inputs

As returned by `is_dgkf`, except for:

g candidate gain level

ptol as in `hinfsyn`

Outputs

retval 1 if g exceeds optimal Hinf closed loop gain, else 0

pc solution of "regulator" $\mathcal{H}_\infty$ ARE

pf solution of "filter" $\mathcal{H}_\infty$ ARE

Do not attempt to use this at home; no argument checking performed.

`[xinf, x_ha_err] = hinfsyn_ric (a, bb, c1, d1dot, r, ptol)` [Function File]

Forms

```
xx = ([bb; -c1'*d1dot]/r) * [d1dot'*c1 bb'];
Ha = [a 0*a; -c1'*c1 - a'] - xx;
```

and solves associated Riccati equation. The error code *x_ha_err* indicates one of the following conditions:

0 successful

1 *xinf* has imaginary eigenvalues

2 *hx* not Hamiltonian

3 *xinf* has infinite eigenvalues (numerical overflow)

4 *xinf* not symmetric

5 *xinf* not positive definite

6 *r* is singular

`[k, p, e] = lqe (a, g, c, sigw, sigv, z)` [Function File]

Construct the linear quadratic estimator (Kalman filter) for the continuous time system

$$\frac{dx}{dt} = Ax + Bu$$
$$y = Cx + Du$$

where w and v are zero-mean gaussian noise processes with respective intensities

```
sigw = cov (w, w)
sigv = cov (v, v)
```

The optional argument z is the cross-covariance `cov (w, v)`. If it is omitted, `cov (w, v) = 0` is assumed.

Observer structure is `dz/dt = A z + B u + k (y - C z - D u)`

The following values are returned:

k	The observer gain, $(A - KC)$ is stable.
p	The solution of algebraic Riccati equation.
e	The vector of closed loop poles of $(A - KC)$.

`[k, q1, p1, ee, er] = lqg (sys, sigw, sigv, q, r, in_idx)` [Function File]

Design a linear-quadratic-gaussian optimal controller for the system

```
dx/dt = A x + B u + G w        [w]=N(0,[Sigw 0    ])
    y = C x + v                [v]  (    0   Sigv ])
```

or

```
x(k+1) = A x(k) + B u(k) + G w(k)        [w]=N(0,[Sigw 0    ])
  y(k) = C x(k) + v(k)                   [v]  (    0   Sigv ])
```

Inputs

sys	system data structure
sigw *sigv*	intensities of independent Gaussian noise processes (as above)
q *r*	state, control weighting respectively. Control ARE is
in_idx	names or indices of controlled inputs (see `sysidx`, `cellidx`) default: last dim(R) inputs are assumed to be controlled inputs, all others are assumed to be noise inputs.

Outputs

k	system data structure format LQG optimal controller (Obtain A, B, C matrices with `sys2ss`, `sys2tf`, or `sys2zp` as appropriate).
p1	Solution of control (state feedback) algebraic Riccati equation.
q1	Solution of estimation algebraic Riccati equation.
ee	Estimator poles.
es	Controller poles.

`[k, p, e] = lqr (a, b, q, r, z)` [Function File]

construct the linear quadratic regulator for the continuous time system

$$\frac{dx}{dt} = Ax + Bu$$

to minimize the cost functional

$$J = \int_0^\infty x^T Q x + u^T R u$$

z omitted or

$$J = \int_0^\infty x^T Q x + u^T R u + 2x^T Z u$$

z included.

The following values are returned:

k The state feedback gain, $(A - BK)$ is stable and minimizes the cost functional

p The stabilizing solution of appropriate algebraic Riccati equation.

e The vector of the closed loop poles of $(A - BK)$.

Reference Anderson and Moore, *Optimal control: linear quadratic methods*, Prentice-Hall, 1990, pp. 56–58.

`[y, x] = lsim (sys, u, t, x0)` [Function File]

Produce output for a linear simulation of a system; produces a plot for the output of the system, *sys*.

u is an array that contains the system's inputs. Each row in u corresponds to a different time step. Each column in u corresponds to a different input. t is an array that contains the time index of the system; t should be regularly spaced. If initial conditions are required on the system, the *x0* vector should be added to the argument list.

When the lsim function is invoked a plot is not displayed; however, the data is returned in y (system output) and x (system states).

`K = place (sys, p)` [Function File]

Computes the matrix K such that if the state is feedback with gain K, then the eigenvalues of the closed loop system (i.e. $A - BK$) are those specified in the vector p.

Version: Beta (May-1997): If you have any comments, please let me know. (see the file place.m for my address)

29.10 Miscellaneous Functions (Not yet properly filed/documented)

`axis2dlim (axdata)` [Function File]

Determine axis limits for 2-D data (column vectors); leaves a 10% margin around the plots. Inserts margins of +/- 0.1 if data is one-dimensional (or a single point).

Input

axdata n by 2 matrix of data [x, y].

Output

axvec Vector of axis limits appropriate for call to `axis` function.

`moddemo (inputs)` [Function File]

Octave Control toolbox demo: Model Manipulations demo.

`prompt (str)` [Function File]

Prompt user to continue

Input

str Input string. Its default value is:

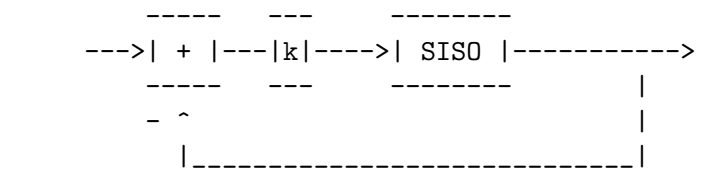
```
\n ---- Press a key to  continue ---
```

`rldemo (inputs)` [Function File]

Octave Control toolbox demo: Root Locus demo.

`[rldata, k] = rlocus (sys[, increment, min_k, max_k])` [Function File]

Displays root locus plot of the specified SISO system.



Inputs

sys system data structure

min_k Minimum value of k

max_k Maximum value of k

increment The increment used in computing gain values

Outputs

Plots the root locus to the screen.

rldata Data points plotted: in column 1 real values, in column 2 the imaginary values.

k Gains for real axis break points.

`[yy, idx] = sortcom (xx[, opt])` [Function File]

Sort a complex vector.

Inputs

xx Complex vector

opt sorting option:

`"re"` Real part (default);

`"mag"` By magnitude;

`"im"` By imaginary part.

if *opt* is not chosen as `"im"`, then complex conjugate pairs are grouped together, $a - jb$ followed by $a + jb$.

Outputs

yy Sorted values

idx Permutation vector: `yy = xx(idx)`

`[num, den] = ss2tf (a, b, c, d)` [Function File]

Conversion from tranfer function to state-space. The state space system:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

is converted to a transfer function:

$$G(s) = \frac{\text{num}(s)}{\text{den}(s)}$$

used internally in system data structure format manipulations.

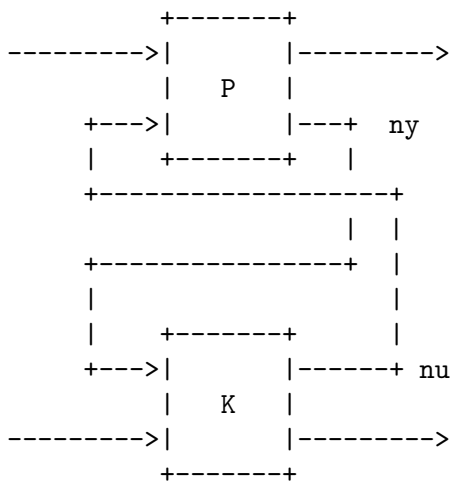
`[pol, zer, k] = ss2zp (a, b, c, d)` [Function File]

Converts a state space representation to a set of poles and zeros; k is a gain associated with the zeros.

Used internally in system data structure format manipulations.

`starp (P, K, ny, nu)` [Function File]

Redheffer star product or upper/lower LFT, respectively.



If *ny* and *nu* "consume" all inputs and outputs of K then the result is a lower fractional transformation. If *ny* and *nu* "consume" all inputs and outputs of P then the result is an upper fractional transformation.

ny and/or *nu* may be negative (i.e. negative feedback).

`[a, b, c, d] = tf2ss (num, den)` [Function File]

Conversion from tranfer function to state-space. The state space system:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

is obtained from a transfer function:

$$G(s) = \frac{\text{num}(s)}{\text{den}(s)}$$

The vector *den* must contain only one row, whereas the vector *num* may contain as many rows as there are outputs y of the system. The state space system matrices obtained from this function will be in controllable canonical form as described in *Modern Control Theory*, (Brogan, 1991).

`[zer, pol, k] = tf2zp (num, den)` [Function File]

Converts transfer functions to poles-and-zero representations.

Returns the zeros and poles of the SISO system defined by *num*/*den*. k is a gain associated with the system zeros.

`[a, b, c, d] = zp2ss (zer, pol, k)` [Function File]

Conversion from zero / pole to state space.

Inputs

zer
pol Vectors of (possibly) complex poles and zeros of a transfer function. Complex values must come in conjugate pairs (i.e., $x + jy$ in *zer* means that $x - jy$ is also in *zer*). The number of zeros must not exceed the number of poles.

k Real scalar (leading coefficient).

Outputs

a
b
c
d The state space system, in the form:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

`[num, den] = zp2tf (zer, pol, k)` [Function File]

Converts zeros / poles to a transfer function.

Inputs

zer
pol Vectors of (possibly complex) poles and zeros of a transfer function. Complex values must appear in conjugate pairs.

k Real scalar (leading coefficient).

30 Signal Processing

I hope that someday Octave will include more signal processing functions. If you would like to help improve Octave in this area, please contact `bug@octave.org`.

`detrend (x, p)` [Function File]

If x is a vector, `detrend (x, p)` removes the best fit of a polynomial of order p from the data x.

If x is a matrix, `detrend (x, p)` does the same for each column in x.

The second argument is optional. If it is not specified, a value of 1 is assumed. This corresponds to removing a linear trend.

`fft (a, n, dim)` [Loadable Function]

Compute the FFT of a using subroutines from FFTPACK. The FFT is calculated along the first non-singleton dimension of the array. Thus if a is a matrix, `fft (a)` computes the FFT for each column of a.

If called with two arguments, n is expected to be an integer specifying the number of elements of a to use, or an empty matrix to specify that its value should be ignored. If n is larger than the dimension along which the FFT is calculated, then a is resized and padded with zeros. Otherwise, if$n$ is smaller than the dimension along which the FFT is calculated, then a is truncated.

If called with three agruments, *dim* is an integer specifying the dimension of the matrix along which the FFT is performed

`ifft (a, n, dim)` [Loadable Function]

Compute the inverse FFT of a using subroutines from FFTPACK. The inverse FFT is calculated along the first non-singleton dimension of the array. Thus if a is a matrix, `fft (a)` computes the inverse FFT for each column of a.

If called with two arguments, n is expected to be an integer specifying the number of elements of a to use, or an empty matrix to specify that its value should be ignored. If n is larger than the dimension along which the inverse FFT is calculated, then a is resized and padded with zeros. Otherwise, if$n$ is smaller than the dimension along which the inverse FFT is calculated, then a is truncated.

If called with three agruments, *dim* is an integer specifying the dimension of the matrix along which the inverse FFT is performed

`fft2 (a, n, m)` [Loadable Function]

Compute the two dimensional FFT of a using subroutines from FFTPACK. The optional arguments n and m may be used specify the number of rows and columns of a to use. If either of these is larger than the size of a, a is resized and padded with zeros.

If a is a multi-dimensional matrix, each two-dimensional sub-matrix of a is treated seperately

`fft2 (a, n, m)` [Loadable Function]

Compute the inverse two dimensional FFT of *a* using subroutines from FFTPACK. The optional arguments *n* and *m* may be used specify the number of rows and columns of *a* to use. If either of these is larger than the size of *a*, *a* is resized and padded with zeros.

If *a* is a multi-dimensional matrix, each two-dimensional sub-matrix of *a* is treated seperately

`fftn (a, size)` [Loadable Function]

Compute the N dimensional FFT of *a* using subroutines from FFTPACK. The optional vector argument *size* may be used specify the dimensions of the array to be used. If an element of *size* is smaller than the corresponding dimension, then the dimension is truncated prior to performing the FFT. Otherwise if an element of *size* is larger than the corresponding dimension *a* is resized and padded with zeros.

`ifftn (a, size)` [Loadable Function]

Compute the invesre N dimensional FFT of *a* using subroutines from FFTPACK. The optional vector argument *size* may be used specify the dimensions of the array to be used. If an element of *size* is smaller than the corresponding dimension, then the dimension is truncated prior to performing the inverse FFT. Otherwise if an element of *size* is larger than the corresponding dimension *a* is resized and padded with zeros.

`fftconv (a, b, n)` [Function File]

Return the convolution of the vectors *a* and *b*, as a vector with length equal to the `length (a) + length (b) - 1`. If *a* and *b* are the coefficient vectors of two polynomials, the returned value is the coefficient vector of the product polynomial.

The computation uses the FFT by calling the function `fftfilt`. If the optional argument *n* is specified, an N-point FFT is used.

`fftfilt (b, x, n)` [Function File]

With two arguments, `fftfilt` filters *x* with the FIR filter *b* using the FFT.

Given the optional third argument, *n*, `fftfilt` uses the overlap-add method to filter *x* with *b* using an N-point FFT.

If *x* is a matrix, filter each column of the matrix.

`y = filter (b, a, x)` [Loadable Function]
`[y, sf] = filter (b, a, x, si)` [Loadable Function]
`[y, sf] = filter (b, a, x, [], dim)` [Loadable Function]
`[y, sf] = filter (b, a, x, si, dim)` [Loadable Function]

Return the solution to the following linear, time-invariant difference equation:

$$\sum_{k=0}^{N} a_{k+1} y_{n-k} = \sum_{k=0}^{M} b_{k+1} x_{n-k}, \qquad 1 \le n \le P$$

where $a \in \Re^{N-1}$, $b \in \Re^{M-1}$, and $x \in \Re^{P}$. over the first non-singleton dimension of x or over *dim* if supplied. An equivalent form of this equation is:

$$y_n = -\sum_{k=1}^{N} c_{k+1} y_{n-k} + \sum_{k=0}^{M} d_{k+1} x_{n-k}, \qquad 1 \le n \le P$$

where $c = a/a_1$ and $d = b/a_1$.

If the fourth argument *si* is provided, it is taken as the initial state of the system and the final state is returned as *sf*. The state vector is a column vector whose length is equal to the length of the longest coefficient vector minus one. If *si* is not supplied, the initial state vector is set to all zeros.

In terms of the z-transform, y is the result of passing the discrete- time signal x through a system characterized by the following rational system function:

$$H(z) = \frac{\sum_{k=0}^{M} d_{k+1} z^{-k}}{1 + \sum_{k+1}^{N} c_{k+1} z^{-k}}$$

`[h, w] = freqz (b, a, n, "whole")` [Function File]

Return the complex frequency response *h* of the rational IIR filter whose numerator and denominator coefficients are *b* and *a*, respectively. The response is evaluated at *n* angular frequencies between 0 and 2π.

The output value *w* is a vector of the frequencies.

If the fourth argument is omitted, the response is evaluated at frequencies between 0 and π.

If *n* is omitted, a value of 512 is assumed.

If *a* is omitted, the denominator is assumed to be 1 (this corresponds to a simple FIR filter).

For fastest computation, *n* should factor into a small number of small primes.

`h = freqz (b, a, w)` [Function File]

Evaluate the response at the specific frequencies in the vector *w*. The values for *w* are measured in radians.

`[...] = freqz (..., Fs)` [Function File]

Return frequencies in Hz instead of radians assuming a sampling rate *Fs*. If you are evaluating the response at specific frequencies *w*, those frequencies should be requested in Hz rather than radians.

`freqz (...)` [Function File]

Plot the pass band, stop band and phase response of *h* rather than returning them.

`freqz_plot (w, h)` [Function File]

Plot the pass band, stop band and phase response of *h*.

`sinc (x)` [Function File]

Return $\sin(\pi x)/(\pi x)$.

`b = unwrap (a, tol, dim)` [Function File]

Unwrap radian phases by adding multiples of 2*pi as appropriate to remove jumps greater than *tol*. *tol* defaults to pi.

Unwrap will unwrap along the first non-singleton dimension of *a*, unless the optional argument *dim* is given, in which case the data will be unwrapped along this dimension

`[a, b] = arch_fit (y, x, p, iter, gamma, a0, b0)` [Function File]

Fit an ARCH regression model to the time series y using the scoring algorithm in Engle's original ARCH paper. The model is

```
y(t) = b(1) * x(t,1) + ... + b(k) * x(t,k) + e(t),
h(t) = a(1) + a(2) * e(t-1)^2 + ... + a(p+1) * e(t-p)^2
```

in which $e(t)$ is $N(0, h(t))$, given a time-series vector y up to time $t-1$ and a matrix of (ordinary) regressors x up to t. The order of the regression of the residual variance is specified by p.

If invoked as `arch_fit (y, k, p)` with a positive integer k, fit an ARCH(k, p) process, i.e., do the above with the t-th row of x given by

```
[1, y(t-1), ..., y(t-k)]
```

Optionally, one can specify the number of iterations *iter*, the updating factor *gamma*, and initial values $a0$ and $b0$ for the scoring algorithm.

`arch_rnd (a, b, t)` [Function File]

Simulate an ARCH sequence of length t with AR coefficients b and CH coefficients a. I.e., the result $y(t)$ follows the model

```
y(t) = b(1) + b(2) * y(t-1) + ... + b(lb) * y(t-lb+1) + e(t),
```

where $e(t)$, given y up to time $t-1$, is $N(0, h(t))$, with

```
h(t) = a(1) + a(2) * e(t-1)^2 + ... + a(la) * e(t-la+1)^2
```

`[pval, lm] = arch_test (y, x, p)` [Function File]

For a linear regression model

```
y = x * b + e
```

perform a Lagrange Multiplier (LM) test of the null hypothesis of no conditional heteroscedascity against the alternative of CH(p).

I.e., the model is

```
y(t) = b(1) * x(t,1) + ... + b(k) * x(t,k) + e(t),
```

given y up to $t-1$ and x up to t, e(t) is $N(0, h(t))$ with

```
h(t) = v + a(1) * e(t-1)^2 + ... + a(p) * e(t-p)^2,
```

and the null is $a(1) == \ldots == a(p) == 0$.

If the second argument is a scalar integer, k, perform the same test in a linear autoregression model of order k, i.e., with

```
[1, y(t-1), ..., y(t-k)]
```

as the t-th row of x.

Under the null, LM approximately has a chisquare distribution with p degrees of freedom and *pval* is the p-value (1 minus the CDF of this distribution at LM) of the test.

If no output argument is given, the p-value is displayed.

`arma_rnd (a, b, v, t, n)` [Function File]

Return a simulation of the ARMA model

```
x(n) = a(1) * x(n-1) + ... + a(k) * x(n-k)
     + e(n) + b(1) * e(n-1) + ... + b(l) * e(n-l)
```

in which k is the length of vector a, l is the length of vector b and e is gaussian white noise with variance v. The function returns a vector of length t.

The optional parameter n gives the number of dummy $x(i)$ used for initialization, i.e., a sequence of length t+n is generated and $x(n$+1:t+$n)$ is returned. If n is omitted, n = 100 is used.

`autocor (x, h)` [Function File]

Return the autocorrelations from lag 0 to h of vector x. If h is omitted, all autocorrelations are computed. If x is a matrix, the autocorrelations of each column are computed.

`autocov (x, h)` [Function File]

Return the autocovariances from lag 0 to h of vector x. If h is omitted, all autocovariances are computed. If x is a matrix, the autocovariances of each column are computed.

`autoreg_matrix (y, k)` [Function File]

Given a time series (vector) y, return a matrix with ones in the first column and the first k lagged values of y in the other columns. I.e., for t > k, `[1, y(t-1), ..., y(t-k)]` is the t-th row of the result. The resulting matrix may be used as a regressor matrix in autoregressions.

`bartlett (m)` [Function File]

Return the filter coefficients of a Bartlett (triangular) window of length m.

For a definition of the Bartlett window, see e.g. A. V. Oppenheim & R. W. Schafer, "Discrete-Time Signal Processing".

`blackman (m)` [Function File]

Return the filter coefficients of a Blackman window of length m.

For a definition of the Blackman window, see e.g. A. V. Oppenheim & R. W. Schafer, "Discrete-Time Signal Processing".

`[d, dd] = diffpara (x, a, b)` [Function File]

Return the estimator d for the differencing parameter of an integrated time series.

The frequencies from $[2 * pi * a/t, 2 * pi * b/T]$ are used for the estimation. If b is omitted, the interval $[2 * pi/T, 2 * pi * a/T]$ is used. If both b and a are omitted then $a = 0.5 * sqrt(T)$ and $b = 1.5 * sqrt(T)$ is used, where T is the sample size. If x is a matrix, the differencing parameter of each column is estimated.

The estimators for all frequencies in the intervals described above is returned in *dd*. The value of d is simply the mean of *dd*.

Reference: Brockwell, Peter J. & Davis, Richard A. Time Series: Theory and Methods Springer 1987.

`durbinlevinson (c, oldphi, oldv)` [Function File]

Perform one step of the Durbin-Levinson algorithm.

The vector c specifies the autocovariances `[gamma_0, ..., gamma_t]` from lag 0 to t, *oldphi* specifies the coefficients based on $c(t\text{-}1)$ and *oldv* specifies the corresponding error.

If *oldphi* and *oldv* are omitted, all steps from 1 to t of the algorithm are performed.

`fftshift (v)` [Function File]
`fftshift (v, dim)` [Function File]

Perform a shift of the vector v, for use with the `fft` and `ifft` functions, in order the move the frequency 0 to the center of the vector or matrix.

If v is a vector of N elements corresponding to N time samples spaced of Dt each, then `fftshift (fft (v))` corresponds to frequencies

```
f = ((1:N) - ceil(N/2)) / N / Dt
```

If v is a matrix, the same holds for rows and columns. If v is an array, then the same holds along each dimension.

The optional *dim* argument can be used to limit the dimension along which the permutation occurs.

`fractdiff (x, d)` [Function File]

Compute the fractional differences $(1 - L)^d x$ where L denotes the lag-operator and d is greater than -1.

`hamming (m)` [Function File]

Return the filter coefficients of a Hamming window of length m.

For a definition of the Hamming window, see e.g. A. V. Oppenheim & R. W. Schafer, "Discrete-Time Signal Processing".

`hanning (m)` [Function File]

Return the filter coefficients of a Hanning window of length m.

For a definition of this window type, see e.g. A. V. Oppenheim & R. W. Schafer, "Discrete-Time Signal Processing".

`hurst (x)` [Function File]
Estimate the Hurst parameter of sample x via the rescaled range statistic. If x is a matrix, the parameter is estimated for every single column.

`periodogram (x)` [Function File]
For a data matrix x from a sample of size n, return the periodogram.

`rectangle_lw (n, b)` [Function File]
Rectangular lag window. Subfunction used for spectral density estimation.

`rectangle_sw (n, b)` [Function File]
Rectangular spectral window. Subfunction used for spectral density estimation.

`sinetone (freq, rate, sec, ampl)` [Function File]
Return a sinetone of frequency *freq* with length of *sec* seconds at sampling rate *rate* and with amplitude *ampl*. The arguments *freq* and *ampl* may be vectors of common size.

Defaults are *rate* = 8000, *sec* = 1 and *ampl* = 64.

`sinewave (m, n, d)` [Function File]
Return an m-element vector with i-th element given by `sin (2 * pi * (i+d-1) / n)`.

The default value for d is 0 and the default value for n is m.

`spectral_adf (c, win, b)` [Function File]
Return the spectral density estimator given a vector of autocovariances c, window name *win*, and bandwidth, b.

The window name, e.g., `"triangle"` or `"rectangle"` is used to search for a function called `win_sw`.

If *win* is omitted, the triangle window is used. If b is omitted, `1 / sqrt (length (x))` is used.

`spectral_xdf (x, win, b)` [Function File]
Return the spectral density estimator given a data vector x, window name *win*, and bandwidth, b.

The window name, e.g., `"triangle"` or `"rectangle"` is used to search for a function called `win_sw`.

If *win* is omitted, the triangle window is used. If b is omitted, `1 / sqrt (length (x))` is used.

`spencer (x)` [Function File]
Return Spencer's 15 point moving average of every single column of x.

`[y, c] =` *stft* (*x*, *win_size*, *inc*, *num_coef*, *w_type*) [Function File]
Compute the short-term Fourier transform of the vector *x* with *num_coef* coefficients by applying a window of *win_size* data points and an increment of *inc* points.

Before computing the Fourier transform, one of the following windows is applied:

hanning w_type = 1

hamming w_type = 2

rectangle w_type = 3

The window names can be passed as strings or by the *w_type* number.

If not all arguments are specified, the following defaults are used: *win_size* = 80, *inc* = 24, *num_coef* = 64, and *w_type* = 1.

`y = stft (x, ...)` returns the absolute values of the Fourier coefficients according to the *num_coef* positive frequencies.

`[y, c] = stft (x, ...)` returns the entire STFT-matrix *y* and a 3-element vector *c* containing the window size, increment, and window type, which is needed by the synthesis function.

`synthesis (y, c)` [Function File]
Compute a signal from its short-time Fourier transform *y* and a 3-element vector *c* specifying window size, increment, and window type.

The values *y* and *c* can be derived by

```
[y, c] = stft (x , ...)
```

`triangle_lw (n, b)` [Function File]
Triangular lag window. Subfunction used for spectral density estimation.

`triangle_sw (n, b)` [Function File]
Triangular spectral window. Subfunction used for spectral density estimation.

`[a, v] = yulewalker (c)` [Function File]
Fit an AR (p)-model with Yule-Walker estimates given a vector *c* of autocovariances `[gamma_0, ..., gamma_p]`.

Returns the AR coefficients, *a*, and the variance of white noise, *v*.

31 Image Processing

Octave can display images with the X Window System using the `xloadimage` program. You do not need to be running X in order to manipulate images, however, so some of these functions may be useful even if you are not able to view the results.

Loading images only works with Octave's image format (a file with a matrix containing the image data, and a matrix containing the colormap). Contributions of robust, well-written functions to read other image formats are welcome. If you can provide them, or would like to improve Octave's image processing capabilities in other ways, please contact `bug@octave.org`.

`colormap (`*`map`*`)` [Function File]
`colormap ("`*default*`")` [Function File]

Set the current colormap.

`colormap (`*`map`*`)` sets the current colormap to *map*. The color map should be an *n* row by 3 column matrix. The columns contain red, green, and blue intensities respectively. All entries should be between 0 and 1 inclusive. The new colormap is returned.

`colormap ("default")` restores the default colormap (a gray scale colormap with 64 entries). The default colormap is returned.

With no arguments, `colormap` returns the current color map.

`gray (`*`n`*`)` [Function File]

Return a gray colormap with *n* entries corresponding to values from 0 to *n*-1. The argument *n* should be a scalar. If it is omitted, 64 is assumed.

`[`*`img`*`, `*`map`*`] = gray2ind ()` [Function File]

Convert a gray scale intensity image to an Octave indexed image.

`image (`*`x`*`, `*`zoom`*`)` [Function File]
`image (`*`x`*`, `*`y`*`, `*`A`*`, `*`zoom`*`)` [Function File]

Display a matrix as a color image. The elements of *x* are indices into the current colormap and should have values between 1 and the length of the colormap. If *zoom* is omitted, the image will be scaled to fit within 600x350 (to a max of 4).

It first tries to use `display` from `ImageMagick` then `xv` and then `xloadimage`.

The axis values corresponding to the matrix elements are specified in *x* and *y*. At present they are ignored.

`imagesc (`*`A`*`)` [Function File]
`imagesc (`*`x`*`, `*`y`*`, `*`A`*`)` [Function File]
`imagesc (...,` *`zoom`*`)` [Function File]
`imagesc (...,` *`limits`*`)` [Function File]
`B` `= imagesc (...)` [Function File]

Display a scaled version of the matrix *A* as a color image. The matrix is scaled so that its entries are indices into the current colormap. The scaled matrix is returned.

If *zoom* is omitted, a comfortable size is chosen. If *limits* = [*lo*, *hi*] are given, then that range maps into the full range of the colormap rather than the minimum and maximum values of *A*.

The axis values corresponding to the matrix elements are specified in *x* and *y*, either as pairs giving the minimum and maximum values for the respective axes, or as values for each row and column of the matrix *A*. At present they are ignored.

`imshow (i)` [Function File]
`imshow (x, map)` [Function File]
`imshow (i, n)` [Function File]
`imshow (r, g, b)` [Function File]

Display an image.

`imshow (x)` displays an image *x*. The numerical class of the image determines its bit-depth: 1 for `logical`, 8 for `uint8` and `logical`, and 16 for `double` or `uint16`. If *x* has dimensions MxNx3, the three matrices represent the red, green and blue components of the image.

`imshow (x, map)` displays an indexed image using the specified colormap.

`imshow (i, n)` displays a gray scale intensity image of N levels.

`imshow (r, g, b)` displays an RGB image.

The character string `"truesize"` can always be used as an optional final argument to prevent automatic zooming of the image.

`ind2gray (x, map)` [Function File]

Convert an Octave indexed image to a gray scale intensity image. If *map* is omitted, the current colormap is used to determine the intensities.

`[r, g, b] = ind2rgb (x, map)` [Function File]

Convert an indexed image to red, green, and blue color components. If *map* is omitted, the current colormap is used for the conversion.

`[x, map] = loadimage (file)` [Function File]

Load an image file and it's associated color map from the specified *file*. The image must be stored in Octave's image format.

`rgb2ntsc (rgb)` [Function File]

Image format conversion.

`ntsc2rgb (yiq)` [Function File]

Image format conversion.

rgb_map = *hsv2rgb* (`hsv_map`) [Function File]

Transform a colormap from the hsv space to the rgb space.

hsv_map = *rgb2hsv* (*rgb_map*) [Function File]

Transform a colormap from the rgb space to the hsv space.

A color n the RGB space consists of the red, green and blue intensities.

In the HSV space each color is represented by their hue, saturation and value (brightness). Value gives the amount of light in the color. Hue describes the dominant wavelegth. Saturation is the amount of Hue mixed into the color.

ocean (*n*) [Function File]

Create color colormap. The argument *n* should be a scalar. If it is omitted, 64 is assumed.

[*x*, *map*] = rgb2ind (*r*, *g*, *b*) [Function File]

Convert and RGB image to an Octave indexed image.

saveimage (*file*, *x*, *fmt*, *map*) [Function File]

Save the matrix *x* to *file* in image format *fmt*. Valid values for *fmt* are

"img" Octave's image format. The current colormap is also saved in the file.

"ppm" Portable pixmap format.

"ps" PostScript format. Note that images saved in PostScript format can not be read back into Octave with loadimage.

If the fourth argument is supplied, the specified colormap will also be saved along with the image.

Note: if the colormap contains only two entries and these entries are black and white, the bitmap ppm and PostScript formats are used. If the image is a gray scale image (the entries within each row of the colormap are equal) the gray scale ppm and PostScript image formats are used, otherwise the full color formats are used.

IMAGEPATH [Built-in Variable]

A colon separated list of directories in which to search for image files.

32 Audio Processing

Octave provides a few functions for dealing with audio data. An audio 'sample' is a single output value from an A/D converter, i.e., a small integer number (usually 8 or 16 bits), and audio data is just a series of such samples. It can be characterized by three parameters: the sampling rate (measured in samples per second or Hz, e.g. 8000 or 44100), the number of bits per sample (e.g. 8 or 16), and the number of channels (1 for mono, 2 for stereo, etc.).

There are many different formats for representing such data. Currently, only the two most popular, *linear encoding* and *mu-law encoding*, are supported by Octave. There is an excellent FAQ on audio formats by Guido van Rossum <guido@cwi.nl> which can be found at any FAQ ftp site, in particular in the directory '`/pub/usenet/news.answers/audio-fmts`' of the archive site `rtfm.mit.edu`.

Octave simply treats audio data as vectors of samples (non-mono data are not supported yet). It is assumed that audio files using linear encoding have one of the extensions '`lin`' or '`raw`', and that files holding data in mu-law encoding end in '`au`', '`mu`', or '`snd`'.

`lin2mu (`*x*, *n*`)` [Function File]

Converts audio data from linear to mu-law. Mu-law values use 8-bit unsigned integers. Linear values use *n*-bit signed integers or floating point values in the range -1<=*x*<=1 if *n* is 0. If *n* is not specified it defaults to 0, 8 or 16 depending on the range values in *x*.

`mu2lin (`*x*, *bps*`)` [Function File]

Converts audio data from linear to mu-law. Mu-law values are 8-bit unsigned integers. Linear values use *n*-bit signed integers or floating point values in the range -1<=y<=1 if *n* is 0. If *n* is not specified it defaults to 8.

`loadaudio (`*name*, *ext*, *bps*`)` [Function File]

Loads audio data from the file '*name*`.`*ext*' into the vector *x*.

The extension *ext* determines how the data in the audio file is interpreted; the extensions '`lin`' (default) and '`raw`' correspond to linear, the extensions '`au`', '`mu`', or '`snd`' to mu-law encoding.

The argument *bps* can be either 8 (default) or 16, and specifies the number of bits per sample used in the audio file.

`saveaudio (`*name*, *x*, *ext*, *bps*`)` [Function File]

Saves a vector *x* of audio data to the file '*name*`.`*ext*'. The optional parameters *ext* and *bps* determine the encoding and the number of bits per sample used in the audio file (see `loadaudio`); defaults are '`lin`' and 8, respectively.

The following functions for audio I/O require special A/D hardware and operating system support. It is assumed that audio data in linear encoding can be played and recorded by reading from and writing to '`/dev/dsp`', and that similarly '`/dev/audio`' is used for mu-law encoding. These file names are system-dependent. Improvements so that these functions will work without modification on a wide variety of hardware are welcome.

`playaudio (`*`name`*, *`ext`*`)` [Function File]
`playaudio (`*`x`*`)` [Function File]
Plays the audio file '*`name.ext`*' or the audio data stored in the vector *x*.

`record (`*`sec`*, *`sampling_rate`*`)` [Function File]
Records *sec* seconds of audio input into the vector *x*. The default value for *sampling_rate* is 8000 samples per second, or 8kHz. The program waits until the user types ⟨RET⟩ and then immediately starts to record.

`setaudio ([`*`w_type`* [, *`value`*]])` [Function File]
Execute the shell command '`mixer [`*`w_type`* `[, `*`value`*`]]`'

33 Quaternions

Quaternions are hypercomplex numbers used to represent spatial rotations in three dimensions. This set of routines provides a useful basis for working with quaternions in Octave. A tutorial is in the Octave source, scripts/quaternion/quaternion.ps.

These functions were written by A. S. Hodel, Associate Professor, Auburn University.

`[a, b, c, d] = quaternion (w)` [Function File]
`[vv, theta] = quaternion (w)` [Function File]
`w = quaternion (a, b, c, d)` [Function File]
`w = quaternion (vv, theta)` [Function File]

Construct or extract a quaternion

```
w = a*i + b*j + c*k + d
```

from given data.

`qconj (q)` [Function File]

Conjugate of a quaternion.

```
q = [w, x, y, z] = w*i + x*j + y*k + z
qconj (q) = -w*i -x*j -y*k + z
```

`qderiv (omega)` [Function File]

Derivative of a quaternion.

Let Q be a quaternion to transform a vector from a fixed frame to a rotating frame. If the rotating frame is rotating about the [x, y, z] axes at angular rates [wx, wy, wz], then the derivative of Q is given by

```
Q' = qderivmat (omega) * Q
```

If the passive convention is used (rotate the frame, not the vector), then

```
Q' = -qderivmat (omega) * Q
```

`qderivmat (omega)` [Function File]

Derivative of a quaternion.

Let Q be a quaternion to transform a vector from a fixed frame to a rotating frame. If the rotating frame is rotating about the [x, y, z] axes at angular rates [wx, wy, wz], then the derivative of Q is given by

```
Q' = qderivmat (omega) * Q
```

If the passive convention is used (rotate the frame, not the vector), then

```
Q' = -qderivmat (omega) * Q.
```

`qinv (q)` [Function File]

Return the inverse of a quaternion.

```
q = [w, x, y, z] = w*i + x*j + y*k + z
qmult (q, qinv (q)) = 1 = [0 0 0 1]
```

`qmult (a, b)` [Function File]

Multiply two quaternions.

```
[w, x, y, z] = w*i + x*j + y*k + z
```

identities:

```
i^2 = j^2 = k^2 = -1
ij = k                  jk = i
ki = j                  kj = -i
ji = -k                 ik = -j
```

`qtrans (v, q)` [Function File]

Transform the unit quaternion v by the unit quaternion q. Returns `v = q*v/q`.

`qtransv (v, q)` [Function File]

Transform the 3-D vector v by the unit quaternion q. Return a column vector.

```
vi = (2*real(q)^2 - 1)*vb + 2*imag(q)*(imag(q)'*vb)
   + 2*real(q)*cross(imag(q),vb)
```

Where imag(q) is a column vector of length 3.

`qtransvmat (qib)` [Function File]

Construct a 3x3 transformation matrix from quaternion *qib* that is equivalent to rotation of th radians about axis *vv*, where `[vv, th] = quaternion (qib)`.

`qcoordinate_plot (qf, qb, qv)` [Function File]

Plot in the current figure a set of coordinate axes as viewed from the orientation specified by quaternion *qv*. Inertial axes are also plotted:

qf	Quaternion from reference (x,y,z) to inertial.
qb	Quaternion from reference to body.
qv	Quaternion from reference to view angle.

34 システムユーティリティ

この章では，Octave の外部で何が起こっているのかについての情報を，プログラムの実行中に得ることができ，それを自分のプログラムでこの情報を使用できるようにする関数を説明しています。たとえば，環境変数や現在時刻のについての情報を得たり，Octave プロンプトから他のプログラムを起動したりすらできます。

34.1 時間ユーティリティ

Octave の時間値を操作するための一連の関数の核は，標準 C ライブラリからの対応する関数に似ています。これら関数もいくつかは，以下の要素を含む，時間に関するデータ構造体を使用します。

`usec` 秒以下のマイクロ秒 (0-999999) である。

`sec` 分以下の秒 (0-61) である。この数値は，うるう秒を説明するために 61 をとることがある。

`min` 時以下の分 (0-59) である。

`hour` 深夜からの時 (0-23) である。

`mday` 1 か月内の日 (1-31) である。

`mon` 1 月からの月 (0-11) である。

`year` 1900 年からの年である。

`wday` 日曜日からの曜日 (0-6) である。

`yday` 1 月 1 日からの日数 (0-365) である。

`isdst` 夏時間のフラグである。

`zone` タイムゾーンである。

以下の関数の解説において，この構造体は *tm_struct* として参照する。

`time ()` [Loadable Function]

現在の時間を紀元からの秒数として返す。その紀元は，1970 年 1 月 1 日 00:00:00 CUT（協定世界時）を示す。たとえば，1997 年 2 月 17 日月曜日の 07:15:06 CUT において，`time`によって返される値は 856163706 である。

`ctime (t)` [Function File]

`time`から戻る値（あるいは任意の非負の整数）を，ローカル時刻に変換し，`asctime`と同じ文字列を返す。関数 `ctime (time)` は，`asctime (localtime (time))`に等しい。以下の例を参照せよ。

```
ctime (time ())
⇒ "Mon Feb 17 01:15:06 1997\n"
```

`gmtime (t)` [Loadable Function]

関数 `time`から返る値（あるいは任意の非負の整数）を与え，CUT に対応する時刻構造体を返す。以下に例を示す。

```
gmtime (time ())
     ⇒ {
           usec = 0
           year = 97
           mon = 1
           mday = 17
           sec = 6
           zone = CST
           min = 15
           wday = 1
           hour = 7
           isdst = 0
           yday = 47
         }
```

`localtime (t)` [Loadable Function]

関数timeから返る値（あるいは任意の非負の整数）を与え，ローカルタイムゾーンに対応する時刻構造体を返す。以下に例を示す。

```
localtime (time ())
     ⇒ {
           usec = 0
           year = 97
           mon = 1
           mday = 17
           sec = 6
           zone = CST
           min = 15
           wday = 1
           hour = 1
           isdst = 0
           yday = 47
         }
```

`mktime (tm_struct)` [Loadable Function]

ローカル時刻に対応する時刻構造体を，紀元からの秒数に変換する。以下に例を示す。

```
mktime (localtime (time ())
     ⇒ 856163706
```

`asctime (tm_struct)` [Function File]

以下の 5 つのフィールドをもつフォーマットを使用して，時刻構造体を文字列に変換する: Thu Mar 28 08:40:14 1996 以下に例を示す。

```
asctime (localtime (time ())
⇒ "Mon Feb 17 01:15:06 1997\n"
```

これは`ctime (time ())`と等価である。

`strftime (tm_struct)` [Loadable Function]

時刻構造体を，printfと同様の '%'代入子を用いて柔軟にフォーマットする。ここで述べたものを除き，代入フィールドは固定サイズである; 数値フィールドは必要に応じて詰められる。標準ではゼロで埋める; 1 個の数を表示するフィールドについて，詰め物は，以下に述べる修飾子をもつ '%'によって変更されたり引き継がれる。未知のフィールド指定子は，通常の文字としてコピーされる。全ての他の文字は，変更なく出力にコピーされる。たとえば，以下の例を参照せよ。

```
strftime ("%r (%Z) %A %e %B %Y", localtime (time ()))
     ⇒ "01:15:06 AM (CST) Monday 17 February 1997"
```

Octave の strftime関数は，ANSI C フィールド指定子の上位互換性をサポートしている。

リテラル文字フィールド:

`%` % という文字

`n` 改行文字

`t` タブ文字

数値修飾子（標準にはない拡張）:

`-`（ダッシュ）
フィールドを詰めものをしない

`_`（アンダースコア）
フィールドをスペースで埋める

時刻フィールド:

`%H` 時 (00-23)

`%I` 時 (01-12)

`%k` 時 (0-23)

`%l` 時 (1-12)

`%M` 分 (00-59)

`%p` ロケールの AM または PM

`%r` 時間，12 時間表示 (hh:mm:ss [AP]M)

`%R` 時間，24 時間表示 (hh:mm)

`%s` 1970 年 1 月 1 日 00:00:00 からの秒（標準にはない拡張）

`%S` 秒 (00-61)

`%T` 時間，24 時間表示 (hh:mm：ss)

`%X` ロケールの時刻表示 (%H:%M:%S)

`%Z` タイムゾーン（EDT）あるいはタイムゾーンが決定できなければ何もなし

日付フィールド:

`%a` ロケールの短縮曜日名 (Sun-Sat)

`%A` ロケールの完全曜日名，文字列の長さは異なる (Sunday-Saturday)

%b	ロケールの短縮月名 (Jan-Dec)
%B	ロケールの完全月名，文字列の長さは異なる (January-December)
%c	ロケールの日付と時刻 (Sat Nov 04 12:02:33 EST 1989)
%C	世紀 (00-99)
%d	日 (01-31)
%e	日 (1-31)
%D	日付 (mm/dd/yy)
%h	%b に同じ
%j	この年の何日目か (001-366)
%m	月 (01-12)
%U	日曜日を週の初めとするとき，何週目か (00-53)
%w	この週の何日目か (0-6)
%W	月曜日を週の初めとするとき，何週目か (00-53)
%x	ロケールの日付表記 (mm/dd/yy)
%y	年の下 2 桁 (00-99)
%Y	年 (1970-)

`[`*tm_struct*`, `*nchars*`] = strptime (`*str*`, `*fmt*`)` [Loadable Function]

文字列 *str* を，フォーマット *fmt* の支配下で，時刻構造体に変換する。

このセクションで解説している残りの関数の大部分は，標準 C ライブラリにちなんだものではない。その中には MATLAB との互換性のために利用可能であり，それ以外の関数は有用だという理由で提供しています。

`clock ()` [Function File]

現在の年，月（1-12），日（1-31），時間（0-23），分（0-59）および秒（0-61）を含むベクトルを返す。以下に例を示す。

```
clock ()
⇒ [ 1993, 8, 20, 4, 56, 1 ]
```

関数 clockは，gettimeofday関数をもつシステムにおいてより正確である。

`date ()` [Function File]

日付を DD-MMM-YY 形式の文字列を返す。以下に例を示す。

```
date ()
⇒ "20-Aug-93"
```

`etime (`*t1*`, `*t2*`)` [Function File]

clockによって返される 2 値の間の差を（秒で）返す。たとえば，

```
t0 = clock ();
 many computations later...
elapsed_time = etime (clock (), t0);
```

この式は，変数 t0をセットしてからの秒数を elapsed_timeにセットする。

`[`*total*`, `*user*`, `*system*`] = cputime ();` [Function File]

実行中の Octave セッションによって使用された CPU 時間を返す。1 番めの出力はそのプロセスが実行するのに費やした総時間であり，2 番めと 3 番めの出力の和に等しい。ここでこれらの出力は，それぞれユーザモードおよびシステムモードでの実行に費やした CPU 秒数である。使用しているシステムが CPU 時間の利用を報告するための方策を持たないならば，その出力値の各々について 0 を返す。Octave は起動にいくぶん CPU 時間を使用するので，使用した総 CPU 時間がゼロでないことを確かめるチェックを行うことにより，`cputime` 関数が動作しているかどうかを確認をすることは合理的である。

`is_leap_year (`*year*`)` [Function File]

与えられた年 *year* がうるう年ならば 1，そうでなければ 0 を返す。もし何も引数を与えなければ，`is_leap_year`は今年を使用する。以下に例を示す。

```
is_leap_year (2000)
⇒ 1
```

`tic ()` [Function File]
`toc ()` [Function File]

これらの関数は，掛け時計タイマーをセットしたりチェックしたりする。たとえば，

```
tic ();
 多くの計算を行った後...
elapsed_time = toc ();
```

この式は，最も最近，関数 `tic`を呼び出したときからの秒数を，変数 `elapsed_time`に返す。

プロセスが使用した CPU 時間により興味があるならば，かわりに `cputime` 関数を使用するべきである。`tic`と `toc`関数は，呼び出しの間に経過した実際の時刻を報告する。この値には，他の作業を処理した時間や，何もしなかった時間も含んでいる。以下の例を参照せよ。

```
tic (); sleep (5); toc ()
⇒ 5
t = cputime (); sleep (5); cputime () - t
⇒ 0
```

（この例は，CPU タイマがかなり荒い精度であることも示している。）

`pause (`*seconds*`)` [Built-in Function]

プログラムの実行を一時中断する。何も引数を付けずに起動すると，Octave は何か文字が入力されるまで待機する。数値を引数とすると，与えられた秒数だけ待機する。たとえば，以下のステートメントはメッセージを表示し，その後，スクリーンをクリアする前に 5 秒待機する。

```
fprintf (stderr, "wait please...
");
pause (5);
clc;
```

`sleep (`*seconds*`)` [Built-in Function]

与えた数値の秒数だけプログラムの実行を一時中断する。

`usleep (`*`microseconds`*`)` [Built-in Function]
与えられた数値のマイクロ秒（1/1000 秒）だけプログラムの実行を一時中断する。1 秒以下の時間のスリープが可能ではないシステムにおいては，`usleep`は `round (`*`microseconds`*` / 1e6)`秒だけ実行を一時中断する。

34.2 ファイルシステムユーティリティ

Octave には，ファイル名を変更や削除，ディレクトリを作成，削除，読み込み，ファイルの状態についての情報を得るための，以下の関数群が含まれています。

`[`*`err`*`, `*`msg`*`] = rename (`*`old`*`, `*`new`*`)` [Built-in Function]
ファイル名を *old* から *new* へ変更する。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

`[`*`err`*`, `*`msg`*`] = link (`*`old`*`, `*`new`*`)` [Built-in Function]
存在するファイルへの新たなリンク（ハードリンクとしても知られている）を作成する。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

`[`*`err`*`, `*`msg`*`] = symlink (`*`old`*`, `*`new`*`)` [Built-in Function]
文字列 *old* を含むシンボリックリンク *new* を作成する。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

`[`*`result`*`, `*`err`*`, `*`msg`*`] = readlink (`*`symlink`*`)` [Built-in Function]
シンボリックリンク *symlink* を読み込む。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

`[`*`err`*`, `*`msg`*`] = unlink (`*`file`*`)` [Built-in Function]
ファイル *file* を削除する。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

`[`*`files`*`, `*`err`*`, `*`msg`*`] = readdir (`*`dir`*`)` [Built-in Function]
ディレクトリ *dir* にあるファイルの名前を，文字列のセル配列 *files* として返す。エラーが発生したならば，空のセル配列を返す。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

`[`*`err`*`, `*`msg`*`] = mkdir (`*`dir`*`)` [Built-in Function]
dir という名前のディレクトリを作成する。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

`[err, msg] = rmdir (dir)` [Built-in Function]

dir という名前のディレクトリを削除する。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

`[err, msg] = mkfifo (name, mode)` [Built-in Function]

ファイル名が *name* である特殊ファイル *fifo* を，ファイルモード *mode* で作成する。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

`umask (mask)` [Built-in Function]

ファイル生成時のパーミッションマスクを指定する。引数 *mask* は整数であり，8 進数として解釈される。成功すると，マスクの以前の値（8 進数として解釈される整数）を返す。そうでなければ，エラーメッセージを表示する

`[info, err, msg] = stat (file)` [Built-in Function]
`[info, err, msg] = lstat (file)` [Built-in Function]

ファイル *file* についての以下の情報を含む構造体 *s* を返す。

`dev`	このファイルに関するディレクトリエントリを含むデバイスの ID
`ino`	そのファイルのファイル番号
`modestr`	ファイルモード（`ls -l`によって返されるものと同じように，10 個の文字またはダッシュとして返す）
`nlink`	リンクの数
`uid`	ファイル所有者のユーザ ID
`gid`	ファイルグループのグループ ID
`rdev`	ブロックまたはキャラクタ特殊ファイルに対するデバイスの ID
`size`	バイト表記のサイズ
`atime`	最終アクセス時刻（`time`から返る時刻値と同じ形式）Section 34.1 [Timing Utilities], 頁 285 を参照せよ。
`mtime`	最終修正時刻（`time`から返る時刻値と同じ形式）Section 34.1 [Timing Utilities], 頁 285 を参照せよ。
`ctime`	最終ファイル状態変更時刻（`time`から返る時刻値と同じ形式）Section 34.1 [Timing Utilities], 頁 285 を参照せよ。
`blksize`	ファイルのブロックサイズ
`blocks`	ファイルが占めるブロック数

この呼び出しが成功すると，*err* は 0 で *msg* は空文字列となる。そのファイルが存在していない，もしくは他のエラーが発生するならば，*s* は空行列，*err* は -1，*msg* には対応するエラーメッセージを含む。

file がシンボリックリンクならば，`stat`関数は，リンクによって参照される実際のファイルについての情報を返す。シンボリックリンクそのものについての情報が欲しいのならば，`lstat`を使用せよ。

以下に例を示す。

```
[s, err, msg] = stat ("/vmlinuz")
      ⇒ s =
        {
          atime = 855399756
          rdev = 0
          ctime = 847219094
          uid = 0
          size = 389218
          blksize = 4096
          mtime = 847219094
          gid = 6
          nlink = 1
          blocks = 768
          modestr = -rw-r--r--
          ino = 9316
          dev = 2049
        }
      ⇒ err = 0
      ⇒ msg =
```

glob (*pattern*) [Built-in Function]

old to *new* に文字列配列を与え，それらのいずれかにマッチするファイル名のセル配列を返す。マッチしなければ，空のセル配列を返す。マッチする前ファイルを探す前に，パターンの各々においてチルダ展開を実行する。以下に例を示す。

```
glob ("/vm*")
     ⇒ "/vmlinuz"
```

fnmatch (*pattern*, *string*) [Built-in Function]

ファイルパターンマッチングの規則を利用し，文字列配列 *pattern* の要素のいずれかにマッチする *string* の各要素について 1 または 0 を返す。以下に例を示す。

```
fnmatch ("a*b", ["ab"; "axyzb"; "xyzab"])
     ⇒ [ 1; 1; 0 ]
```

file_in_path (*path*, *file*) [Built-in Function]

file_in_path (*path*, *file*, "*all*") [Built-in Function]

ファイル *file* が *path* に見つかったならば，その絶対名を返す。*path* の値は，組み込み変数LOADPATHの記述フォーマットでコロンで区切ったディレクトリのリストとすべきである。何もファイルが見つからなければ，空の行列を返す。以下に例を示す。

```
file_in_path (LOADPATH, "nargchk.m")
     ⇒ "/usr/local/share/octave/2.0/m/general/nargchk.m"
```

もし 2 番目の引数が文字列のセル配列ならば，セル配列の要素について，パスの各ディレクトリを検索し，最初にマッチしたものを返す。

3 番目のオプション引数"all"を与えると，そのパスで同じファイル名をもつ全てのファイルのリストを含むセル配列を返す。もし何もファイルが見つからなければ，空のセル配列を返す。

`tilde_expand` (*string*) [Built-in Function]

文字列 *string* においてチルダ展開を行う。もし *string* がチルダ文字（'~'）で始まっていれば，最初のスラッシュまでの全ての文字（もしスラッシュがなければ全ての文字）は可能なユーザ名として扱われ，チルダとスラッシュ以降の文字は，ユーザ名のホームディレクトリに置き換えられる。もしチルダの直後にスラッシュがあれば，チルダは Octave を実行しているユーザのホームディレクトリで置き換える。たとえば，

```
tilde_expand ("~joeuser/bin")
     ⇒ "/home/joeuser/bin"
tilde_expand ("~/bin")
     ⇒ "/home/jwe/bin"
```

34.3 サブプロセスのコントロール

Octave は，サブプロセスの開始について，`system`や `popen`のような高水準のコマンドをを備えています。もし何らかの作業を行うめに別のプログラムを実行してその出力を見たいならば，おそらく，これらの関数を使用したいと思うでしょう。

Octave はいくつかの低水準の UNIX ライクな関数も提供しています。この関数は，サブプロセスを開始するために使用することもできますが，高水準関数では実行する方法が見あたらないときにのみ使用するべきです。

`system` (*string*, *return_output*, *type*) [Built-in Function]

文字列 *string* によって指定されたシェルコマンドを実行する。2 番めの引数はオプションである。もし *type* が`"async"`ならば，そのプロセスはバックグラウンドで実行され，子プロセスのプロセス ID が直ちに返る。そうでなければ，そのプロセスは開始し，これが終了するまで Octave は待機する。引数 *type* を省略すると，値`"sync"`を仮定する。

もし 2 つの入力引数が与えられ（*return_output* の実際の値は関係しない），そのサブプロセスは同期的に開始されるならば，あるいは *system* が 1 つの入力引数と 1 つ以上の出力引数で呼び出されるならば，このコマンドからの出力は返される。そうでなければ，もしサブプロセスを同期的に実行しているならば，その出力は標準出力に送られる。*system* で実行したコマンドの出力をページャに送るには，以下のようなコマンドを使用せよ:

```
disp (system (cmd, 1));
```

or

```
printf ("%s
", system (cmd, 1));
```

`system`関数は，2 つの値を返すことになる。1 番目は，そのコマンドが標準出力ストリームに各込んだ任意の出力であり，2 番目はコマンドの出力ステータスである。たとえば，

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

この式は，変数 `output`に文字列 'foo'をセットし，変数 `status`に整数 '2'をセットする。

fid = `popen` (*command*, *mode*) [Built-in Function]

パイプを作成し，プロセスを開始する。実行すべきコマンドの名前は，*command* によって与える。そのプロセスの入出力に対応するファイル ID は，*fid* に返される。引数 *mode* は，以下のようになる。

`"r"` パイプは，そのプロセスの標準出力に結合し，読み込みのためにオープンされる。

`"w"` パイプは，そのプロセスの標準入力に結合し，書き出しのためにオープンされる。

例を挙げる。

```
fid = popen ("ls -ltr / | tail -3", "r");
while (isstr (s = fgets (fid)))
  fputs (stdout, s);
endwhile
     ⊣ drwxr-xr-x  33 root  root  3072 Feb 15 13:28 etc
     ⊣ drwxr-xr-x   3 root  root  1024 Feb 15 13:28 lib
     ⊣ drwxrwxrwt  15 root  root  2048 Feb 17 14:53 tmp
```

pclose (*fid*) [Built-in Function]

popenによってオープンされたファイル ID をクローズする。この目的には，fcloseを使うこともできる。

[*in*, *out*, *pid*] = popen2 (*command*, *args*) [Function File]

2 方向通信を行うサブプロセスを開始する。プロセスの名前は，*command* によって与えられ，*args* はそのコマンドに対するオプションを含む文字列の配列である。入力および出力ストリームに関するファイル識別子は，*in* と *out* に返る。コマンドの実行が成功すると，*pid* はサブプロセスのプロセス ID を含む。そうでなければ，*pid* は -1 である。

以下に例を示す。

```
[in, out, pid] = popen2 ("sort", "-nr");
fputs (in, "these\nare\nsome\nstrings\n");
fclose (in);
while (isstr (s = fgets (out)))
  fputs (stdout, s);
endwhile
fclose (out);
⊣ are
⊣ some
⊣ strings
⊣ these
```

EXEC_PATH [Built-in Variable]

変数 EXEC_PATHは，外部プログラムを実行するときに検索するディレクトリを，コロンで区切って並べたものである。その初期値は,（存在していれば）環境変数 OCTAVE_EXEC_PATHあるいはPATHからとられる。しかし，その値はコマンドライン引数--exec-path PATH，またはスタートアップスクリプトで EXEC_PATHに値を設定することにより，上書きされる。EXEC_PATHの値の先頭（末端）にコロンがあれば，ディレクトリ

octave-home/libexec/octave/site/exec/*arch*
octave-home/libexec/octave/*version*/exec/*arch*

が，EXEC_PATHの先頭（末尾）に追加される。ここで *octave-home* は Octave のすべてがインストールされたトップレベルディレクトリ（標準設定は‘/usr/local’）である。もし EXEC_PATHの値を明示的に指定しなければ，これらの特殊ディレクトリは，シェルパスの先頭に追加される。

大部分のケースにおいて，以下の関数は単にその引数をデコードし，対応する UNIX のシステムコールを行います。これらをどのように使用することができるかの完全な例については，関数 popen2 の定義を見てください。

`[pid, msg] = fork ()` [Built-in Function]

カレントプロセスをのコピーを生成する。

この関数は，以下の値のひとつを返す:

> 0 親プロセスにいる。`fork`から返った値は，子プロセスのプロセス ID である。おそらく，終了すべき子プロセスを待つために準備すべきであろう。

0 子プロセスにいる。別のプロセスを開始するために `exec`を呼び出すことができる。それが失敗すると，`exit`を呼び出すべきであろう。

< 0 何らかの理由で，`fork`の呼び出しが失敗した。回避する行動をとらなければならない。システムに依存するエラーメッセージは，*msg* に入ることになる。

`[err, msg] = exec (file, args)` [Built-in Function]

現在のプロセスを新しいプロセスで置き換える。最初に `fork`を呼び出さずに `exec`を呼ぶと，Octave の現在の処理を終了し，それを *file* という名前のプログラムで置き換える。たとえば,

```
exec ("ls" "-l")
```

この式は `ls`を実行し，シェルのプロンプトに戻る。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

`[file_ids, err, msg] = pipe ()` [Built-in Function]

パイプを作成し，ベクトル *file_ids* を返す。これは，パイプの読み込みと書き出しの末端に対応する。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

`[fid, msg] = dup2 (old, new)` [Built-in Function]

ファイル記述子を複製する。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

`[pid, msg] = waitpid (pid, options)` [Built-in Function]

プロセス *pid* が終了するのを待つ。引数 *pid* は，以下の値をとることができる:

−1 任意の子プロセスを待つ。

0 グループ ID が Octave のインタプリタプロセスと等しい任意の子プロセスを待つ。

> 0 ID が *pid* である子プロセスの終了を待機する。

引数 *options* には，以下の値を設定できる:

0 シグナルを受け取るか子プロセスが終了するまで待機する（引数 *options* を指定しないときは，この動作が標準設定である）。

1 ステータスが直ちに得られないならば，ハングしない。

2 停止した任意の子プロセスのステータスを報告する。また，そのステータスは，それらが停止するまで報告されない。

3 1 と 2 の両方を指定するのと同じ。

もし *pid* の戻り値が 0 より大きいならば，その値は存在する子プロセスのプロセス ID である。もしエラーが発生すると，*pid* は 0 より小さくなり，*msg* にはシステム依存のエラーメッセージが入る。

`[err, msg] = fcntl (fid, request, arg)` [Built-in Function]

オープンしたファイル *fid* のプロパティを変更する。以下の値を *request* として渡すことができる:

`F_DUPFD` 複製したファイル記述子を返す。

`F_GETFD` *fid* に関するファイル記述子フラグを返す。

`F_SETFD` *fid* に関するファイル記述子フラグをセットする。

`F_GETFL` *fid* に関するファイルステータスフラグをセットする。以下のコードが返ってくる（システムによっては，いくつかのフラグが定義されないことがある）。

`O_RDONLY` 読み込み専用でオープンされた

`O_WRONLY` 書き出し専用でオープンされた

`O_RDWR` 読み書き両用でオープンされた

`O_APPEND` 書き出しを追加

`O_CREAT` ファイルが存在していなければファイルを生成する

`O_NONBLOCK`
Nonblocking モード

`O_SYNC` 書き込みが完了するまで待機

`O_ASYNC` 非同期 I/O

`F_SETFL` *fid* についてのファイルステータスフラグを，*arg* によって指定された値にセットする。変更可能なフラグは，`O_APPEND`と `O_NONBLOCK`である。

成功すると，*err* は 0 になり，*msg* は空文字列となる。そうでないならば，*err* はゼロ以外になり，*msg* にはシステム依存のエラーメッセージが入る。

34.4 プロセス，グループ，ユーザ ID

`pgid = getpgrp ()` [Built-in Function]

カレントプロセスのプロセスグループ ID を返す。

`pid = getpid ()` [Built-in Function]

カレントプロセスのプロセス ID を返す。

`pid = getppid ()` [Built-in Function]

親プロセスのプロセス ID を返す。

`euid = geteuid ()` [Built-in Function]

カレントプロセスの有効なユーザ ID を返す。

`uid = getuid ()` [Built-in Function]
カレントプロセスの実ユーザ ID を返す。

`egid = getegid ()` [Built-in Function]
カレントプロセスの有効なグループ ID を返す。

`gid = getgid ()` [Built-in Function]
カレントプロセスの実グループ ID を返す。

34.5 環境変数

`getenv (var)` [Built-in Function]
環境変数 *var* の値を返す。たとえば,

```
getenv ("PATH")
```

この式はパスの値を含む文字列を返す。

`putenv (var, value)` [Built-in Function]
環境変数 *var* に値 *value* をセットする。

34.6 現在の作業ディレクトリ

`cd` *dir* [Command]
`chdir` *dir* [Command]
カレント作業ディレクトリを *dir* に変更する。*dir* を省略すると，ユーザのホームディレクトリに変更する。たとえば,

```
cd ~/octave
```

このコマンドは，カレント作業ディレクトリを '`~/octave`'に変更する。もしそのディレクトリが存在しなければ，エラーメッセージを表示し，作業ディレクトリは変更されない。

`ls` *options* [Command]
ディレクトリの内容を一覧表示する。以下に例を示す。

```
ls -l
     ⊣ total 12
     ⊣ -rw-r--r--   1 jwe  users  4488 Aug 19 04:02 foo.m
     ⊣ -rw-r--r--   1 jwe  users  1315 Aug 17 23:14 bar.m
```

`dir`および `ls`コマンドは，システムのディレクトリ一覧表示コマンドを呼び出すことで実装されている。ゆえに，利用できるオプションは，システムによって変化する。

`pwd ()` [Built-in Function]
カレント作業ディレクトリを返す。

34.7 パスワードデータベース関数

Octave のパスワードデータベース関数は，以下のフィールドをもつ構造体に情報を返す。

`name` ユーザ名

`passwd` 入手できるなら，暗号化されたパスワード

`uid` ユーザ ID の数値

`gid` グループ ID の数値

`gecos` GECOS フィールド

`dir` ホームディレクトリ

`shell` 初期シェル

以下の関数の解説において，このデータ構造体は *pw_struct* として参照している。

pw_struct = `getpwent` () [Loadable Function]
パスワードデータベースから（必要であればオープンして）エントリを含む構造体を返す。一度データの終端に達したならば，`getpwent`は 0 を返す。

pw_struct = `getpwuid` (*uid*). [Loadable Function]
ユーザ ID*uid* について，パスワードデータベースからの最初のエントリを含む構造体を返す。もしユーザ ID がそのデータベースに存在していなければ，0 を返す。

pw_struct = `getpwnam` (*name*) [Loadable Function]
ユーザ名 *name* について，パスワードデータベースからの最初のエントリを含む構造体を返す。もしユーザ名がそのデータベースに存在していなければ，0 を返す。

`setpwent` () [Loadable Function]
パスワードデータベースの始点に対する内部ポインタを返す。

`endpwent` () [Loadable Function]
パスワードデータベースをクローズする。

34.8 グループデータベース関数

Octave のグループデータベース関数は，以下のフィールドをもつ構造体に情報を返す。

`name` ユーザ名

`passwd` 入手できるなら，暗号化されたパスワード

`gid` ユーザ ID の数値

`mem` グループの数値

以下の関数の解説において，このデータ構造体は *grp_struct* として参照している。

grp_struct = `getgrent` () [Loadable Function]
グループデータベースから（必要であればオープンして）エントリを含む構造体を返す。一度データの終端に達したならば，`getgrent`は 0 を返す。

grp_struct = getgrgid (*gid*). [Loadable Function]
グループ ID*gid* について，グループデータベースからの最初のエントリを含む構造体を返す。もしグループ ID がそのデータベースに存在していなければ，0 を返す。

grp_struct = getgrnam (*name*) [Loadable Function]
グループ名 *name* について，グループデータベースからの最初のエントリを含む構造体を返す。もしグループ名がそのデータベースに存在していなければ，0 を返す。

setgrent () [Loadable Function]
グループデータベースの始点に対する内部ポインタを返す。

endgrent () [Loadable Function]
グループデータベースをクローズする。

34.9 システム情報

computer () [Function File]
cpu-vendor-os の形式の文字列を返す。これは，Octave が動作しているコンピュータの種類を識別する。もし出力引数付きで呼び出すと，その値は表示せずに返す。たとえば，以下のようである。

```
computer ()
⊣ i586-pc-linux-gnu

x = computer ()
⇒ x = "i586-pc-linux-gnu"
```

isieee () [Built-in Function]
もし使用しているコンピュータが，浮動小数点演算についての IEEE 標準に従うと主張するならば 1 を返す。

OCTAVE_VERSION [Built-in Variable]
Octave のバージョン数を文字列で表したものである。

octave_config_info (*option*) [Built-in Function]
Octave に関する設定とインストール情報を含む構造体を返す。

option が文字列ならば，指定したオプションについての設定情報を返す。

getrusage () [Loadable Function]
カレント Octave プロセスについて，種々の統計量を含む構造体を返す。全てのシステムで，全ての情報が手にはいるわけではない。もし CPU 時間統計量が得られないならば，CPU 時間項目はゼロにセットされる。他の未取得項目は NaN で置き換えられる。getrusageによって返される構造体に入る，全ての可能なフィールドのリストは，以下のようなものである:

`idrss` 非共有データサイズ

`inblock` Number of block input operations.

`isrss` 非共有スタックサイズ

`ixrss` 共有メモリサイズ

`majflt` Number of major page faults.

`maxrss` Maximum data size.

`minflt` Number of minor page faults.

`msgrcv` Number of messages received.

`msgsnd` Number of messages sent.

`nivcsw` Number of involuntary context switches.

`nsignals` Number of signals received.

`nswap` Number of swaps.

`nvcsw` Number of voluntary context switches.

`oublock` Number of block output operations.

`stime` この構造体は，使用したシステム CPU 時間を含む。その構造体は，要素 `sec`（秒），`usec`（マイクロ秒）をもつ。

`utime` この構造体は，使用したユーザ CPU 時間を含む。その構造体は，要素 `sec`（秒），`usec`（マイクロ秒）をもつ。

付記 A Tips and Standards

This chapter describes no additional features of Octave. Instead it gives advice on making effective use of the features described in the previous chapters.

A.1 Writing Clean Octave Programs

Here are some tips for avoiding common errors in writing Octave code intended for widespread use:

- Since all global variables share the same name space, and all functions share another name space, you should choose a short word to distinguish your program from other Octave programs. Then take care to begin the names of all global variables, constants, and functions with the chosen prefix. This helps avoid name conflicts.

  If you write a function that you think ought to be added to Octave under a certain name, such as `fiddle_matrix`, don't call it by that name in your program. Call it `mylib_fiddle_matrix` in your program, and send mail to `maintainers@octave.org` suggesting that it be added to Octave. If and when it is, the name can be changed easily enough.

  If one prefix is insufficient, your package may use two or three alternative common prefixes, so long as they make sense.

  Separate the prefix from the rest of the symbol name with an underscore '_'. This will be consistent with Octave itself and with most Octave programs.
- When you encounter an error condition, call the function `error` (or `usage`). The `error` and `usage` functions do not return. See Section 2.5 [Errors], 頁 22.
- Please put a copyright notice on the file if you give copies to anyone. Use the same lines that appear at the top of the function files distributed with Octave. If you have not signed papers to assign the copyright to anyone else, then place your name in the copyright notice.

A.2 Tips for Making Code Run Faster.

Here are some ways of improving the execution speed of Octave programs.

- Avoid looping wherever possible.
- Use iteration rather than recursion whenever possible. Function calls are slow in Octave.
- Avoid resizing matrices unnecessarily. When building a single result matrix from a series of calculations, set the size of the result matrix first, then insert values into it. Write

```
result = zeros (big_n, big_m)
for i = over:and_over
  r1 = ...
  r2 = ...
  result (r1, r2) = new_value ();
endfor
```

  instead of

```
result = [];
for i = ever:and_ever
  result = [ result, new_value() ];
endfor
```

- Avoid calling `eval` or `feval` whenever possible, because they require Octave to parse input or look up the name of a function in the symbol table.

  If you are using `eval` as an exception handling mechanism and not because you need to execute some arbitrary text, use the `try` statement instead. See Section 12.9 [The try Statement], 頁 84.
- If you are calling lots of functions but none of them will need to change during your run, set the variable `ignore_function_time_stamp` to `"all"` so that Octave doesn't waste a lot of time checking to see if you have updated your function files.

A.3 Tips for Documentation Strings

Here are some tips for the writing of documentation strings.

- Every command, function, or variable intended for users to know about should have a documentation string.
- An internal variable or subroutine of an Octave program might as well have a documentation string.
- The first line of the documentation string should consist of one or two complete sentences that stand on their own as a summary.

  The documentation string can have additional lines that expand on the details of how to use the function or variable. The additional lines should also be made up of complete sentences.
- For consistency, phrase the verb in the first sentence of a documentation string as an infinitive with "to" omitted. For instance, use "Return the frob of A and B." in preference to "Returns the frob of A and B." Usually it looks good to do likewise for the rest of the first paragraph. Subsequent paragraphs usually look better if they have proper subjects.
- Write documentation strings in the active voice, not the passive, and in the present tense, not the future. For instance, use "Return a list containing A and B." instead of "A list containing A and B will be returned."
- Avoid using the word "cause" (or its equivalents) unnecessarily. Instead of, "Cause Octave to display text in boldface," write just "Display text in boldface."
- Do not start or end a documentation string with whitespace.
- Format the documentation string so that it fits in an Emacs window on an 80-column screen. It is a good idea for most lines to be no wider than 60 characters.

  However, rather than simply filling the entire documentation string, you can make it much more readable by choosing line breaks with care. Use blank lines between topics if the documentation string is long.
- **Do not** indent subsequent lines of a documentation string so that the text is lined up in the source code with the text of the first line. This looks nice in the source code, but looks bizarre when users view the documentation. Remember that the indentation before the starting double-quote is not part of the string!

- The documentation string for a variable that is a yes-or-no flag should start with words such as "Nonzero means. . .", to make it clear that all nonzero values are equivalent and indicate explicitly what zero and nonzero mean.
- When a function's documentation string mentions the value of an argument of the function, use the argument name in capital letters as if it were a name for that value. Thus, the documentation string of the operator `/` refers to its second argument as '`DIVISOR`', because the actual argument name is `divisor`.

  Also use all caps for meta-syntactic variables, such as when you show the decomposition of a list or vector into subunits, some of which may vary.

A.4 Tips on Writing Comments

Here are the conventions to follow when writing comments.

'`#`' Comments that start with a single sharp-sign, '`#`', should all be aligned to the same column on the right of the source code. Such comments usually explain how the code on the same line does its job. In the Emacs mode for Octave, the *`M-;`* (`indent-for-comment`) command automatically inserts such a '`#`' in the right place, or aligns such a comment if it is already present.

'`##`' Comments that start with two semicolons, '`##`', should be aligned to the same level of indentation as the code. Such comments usually describe the purpose of the following lines or the state of the program at that point.

The indentation commands of the Octave mode in Emacs, such as *`M-;`* (`indent-for-comment`) and *`TAB`* (`octave-indent-line`) automatically indent comments according to these conventions, depending on the number of semicolons. See section "Manipulating Comments" in *The GNU Emacs Manual*.

A.5 Conventional Headers for Octave Functions

Octave has conventions for using special comments in function files to give information such as who wrote them. This section explains these conventions.

The top of the file should contain a copyright notice, followed by a block of comments that can be used as the help text for the function. Here is an example:

```
## Copyright (C) 1996, 1997 John W. Eaton
##
## This file is part of Octave.
##
## Octave is free software; you can redistribute it and/or
## modify it under the terms of the GNU General Public
## License as published by the Free Software Foundation;
## either version 2, or (at your option) any later version.
##
## Octave is distributed in the hope that it will be useful,
## but WITHOUT ANY WARRANTY; without even the implied
## warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
## PURPOSE.  See the GNU General Public License for more
```

```
## details.
##
## You should have received a copy of the GNU General Public
## License along with Octave; see the file COPYING.  If not,
## write to the Free Software Foundation, Inc., 51 Franklin Street,
## Fifth Floor, Boston, MA 02110-1301, USA.

## usage: [IN, OUT, PID] = popen2 (COMMAND, ARGS)
##
## Start a subprocess with two-way communication.  COMMAND
## specifies the name of the command to start.  ARGS is an
## array of strings containing options for COMMAND.  IN and
## OUT are the file ids of the input and streams for the
## subprocess, and PID is the process id of the subprocess,
## or -1 if COMMAND could not be executed.
##
## Example:
##
##  [in, out, pid] = popen2 ("sort", "-nr");
##  fputs (in, "these\nare\nsome\nstrings\n");
##  fclose (in);
##  while (isstr (s = fgets (out)))
##    fputs (stdout, s);
##  endwhile
##  fclose (out);
```

Octave uses the first block of comments in a function file that do not appear to be a copyright notice as the help text for the file. For Octave to recognize the first comment block as a copyright notice, it must match the regular expression

```
^ Copyright (C).*\n\n This file is part of Octave.
```

or

```
^ Copyright (C).*\n\n This program is free softwar
```

(after stripping the leading comment characters). This is a fairly strict requirement, and may be relaxed somewhat in the future.

After the copyright notice and help text come several *header comment* lines, each beginning with '`## header-name :`'. For example,

```
## Author: jwe
## Keywords: subprocesses input-output
## Maintainer: jwe
```

Here is a table of the conventional possibilities for *header-name*:

'`Author`' This line states the name and net address of at least the principal author of the library.

```
## Author: John W. Eaton <jwe@bevo.che.wisc.edu>
```

'`Maintainer`'
This line should contain a single name/address as in the Author line, or an address only, or the string '`jwe`'. If there is no maintainer line, the person(s)

in the Author field are presumed to be the maintainers. The example above is mildly bogus because the maintainer line is redundant.

The idea behind the '`Author`' and '`Maintainer`' lines is to make possible a function to "send mail to the maintainer" without having to mine the name out by hand.

Be sure to surround the network address with '`<...>`' if you include the person's full name as well as the network address.

'`Created`' This optional line gives the original creation date of the file. For historical interest only.

'`Version`' If you wish to record version numbers for the individual Octave program, put them in this line.

'`Adapted-By`'
In this header line, place the name of the person who adapted the library for installation (to make it fit the style conventions, for example).

'`Keywords`'
This line lists keywords. Eventually, it will be used by an apropos command to allow people will find your package when they're looking for things by topic area. To separate the keywords, you can use spaces, commas, or both.

Just about every Octave function ought to have the '`Author`' and '`Keywords`' header comment lines. Use the others if they are appropriate. You can also put in header lines with other header names—they have no standard meanings, so they can't do any harm.

付記 B Known Causes of Trouble

This section describes known problems that affect users of Octave. Most of these are not Octave bugs per se—if they were, we would fix them. But the result for a user may be like the result of a bug.

Some of these problems are due to bugs in other software, some are missing features that are too much work to add, and some are places where people's opinions differ as to what is best.

B.1 Actual Bugs We Haven't Fixed Yet

- Output that comes directly from Fortran functions is not sent through the pager and may appear out of sequence with other output that is sent through the pager. One way to avoid this is to force pending output to be flushed before calling a function that will produce output from within Fortran functions. To do this, use the command

  ```
  fflush (stdout)
  ```

  Another possible workaround is to use the command

  ```
  page_screen_output = "false"
  ```

  to turn the pager off.

- If you get messages like

  ```
  Input line too long
  ```

  when trying to plot many lines on one graph, you have probably generated a plot command that is too large for `gnuplot`'s fixed-length buffer for commands. Splitting up the plot command doesn't help because replot is implemented in gnuplot by simply appending the new plotting commands to the old command line and then evaluating it again.

  You can demonstrate this 'feature' by running gnuplot and doing something like

  ```
  plot sin (x), sin (x), sin (x), ... lots more ..., sin (x)
  ```

  and then

  ```
  replot sin (x), sin (x), sin (x), ... lots more ..., sin (x)
  ```

  after repeating the replot command a few times, gnuplot will give you an error.

  Also, it doesn't help to use backslashes to enter a plot command over several lines, because the limit is on the overall command line length, once the backslashed lines are all pasted together.

  Because of this, Octave tries to use as little of the command-line length as possible by using the shortest possible abbreviations for all the plot commands and options. Unfortunately, the length of the temporary file names is probably what is taking up the most space on the command line.

  You can buy a little bit of command line space by setting the environment variable `TMPDIR` to be `"."` before starting Octave, or you can increase the maximum command line length in gnuplot by changing the following limits in the file plot.h in the gnuplot distribution and recompiling gnuplot.

  ```
  #define MAX_LINE_LEN 32768  /* originally 1024 */
  ```

```
#define MAX_TOKENS 8192     /* originally 400 */
```

Of course, this doesn't really fix the problem, but it does make it much less likely that you will run into trouble unless you are putting a very large number of lines on a given plot.

A list of ideas for future enhancements is distributed with Octave. See the file 'PROJECTS' in the top level directory in the source distribution.

B.2 Reporting Bugs

Your bug reports play an essential role in making Octave reliable.

When you encounter a problem, the first thing to do is to see if it is already known. See 付記 B [Trouble], 頁 307. If it isn't known, then you should report the problem.

Reporting a bug may help you by bringing a solution to your problem, or it may not. In any case, the principal function of a bug report is to help the entire community by making the next version of Octave work better. Bug reports are your contribution to the maintenance of Octave.

In order for a bug report to serve its purpose, you must include the information that makes it possible to fix the bug.

If you have Octave working at all, the easiest way to prepare a complete bug report is to use the Octave function `bug_report`. When you execute this function, Octave will prompt you for a subject and then invoke the editor on a file that already contains all the configuration information. When you exit the editor, Octave will mail the bug report for you.

B.3 Have You Found a Bug?

If you are not sure whether you have found a bug, here are some guidelines:

- If Octave gets a fatal signal, for any input whatever, that is a bug. Reliable interpreters never crash.
- If Octave produces incorrect results, for any input whatever, that is a bug.
- Some output may appear to be incorrect when it is in fact due to a program whose behavior is undefined, which happened by chance to give the desired results on another system. For example, the range operator may produce different results because of differences in the way floating point arithmetic is handled on various systems.
- If Octave produces an error message for valid input, that is a bug.
- If Octave does not produce an error message for invalid input, that is a bug. However, you should note that your idea of "invalid input" might be my idea of "an extension" or "support for traditional practice".
- If you are an experienced user of programs like Octave, your suggestions for improvement are welcome in any case.

B.4 Where to Report Bugs

If you have Octave working at all, the easiest way to prepare a complete bug report is to use the Octave function `bug_report`. When you execute this function, Octave will prompt you

for a subject and then invoke the editor on a file that already contains all the configuration information. When you exit the editor, Octave will mail the bug report for you.

If for some reason you cannot use Octave's `bug_report` function, send bug reports for Octave to `bug@octave.org`.

Do not send bug reports to 'help-octave'. Most users of Octave do not want to receive bug reports. Those that do have asked to be on the mailing list.

As a last resort, send bug reports on paper to:

```
Octave Bugs c/o John W. Eaton
University of Wisconsin-Madison
Department of Chemical Engineering
1415 Engineering Drive
Madison, Wisconsin 53706  USA
```

B.5 How to Report Bugs

Send bug reports for Octave to one of the addresses listed in Section B.4 [Bug Lists], 頁 308.

The fundamental principle of reporting bugs usefully is this: **report all the facts**. If you are not sure whether to state a fact or leave it out, state it!

Often people omit facts because they think they know what causes the problem and they conclude that some details don't matter. Thus, you might assume that the name of the variable you use in an example does not matter. Well, probably it doesn't, but one cannot be sure. Perhaps the bug is a stray memory reference which happens to fetch from the location where that name is stored in memory; perhaps, if the name were different, the contents of that location would fool the interpreter into doing the right thing despite the bug. Play it safe and give a specific, complete example.

Keep in mind that the purpose of a bug report is to enable someone to fix the bug if it is not known. Always write your bug reports on the assumption that the bug is not known.

Sometimes people give a few sketchy facts and ask, "Does this ring a bell?" This cannot help us fix a bug. It is better to send a complete bug report to begin with.

Try to make your bug report self-contained. If we have to ask you for more information, it is best if you include all the previous information in your response, as well as the information that was missing.

To enable someone to investigate the bug, you should include all these things:

- The version of Octave. You can get this by noting the version number that is printed when Octave starts, or running it with the '`-v`' option.
- A complete input file that will reproduce the bug.

  A single statement may not be enough of an example—the bug might depend on other details that are missing from the single statement where the error finally occurs.
- The command arguments you gave Octave to execute that example and observe the bug. To guarantee you won't omit something important, list all the options.

  If we were to try to guess the arguments, we would probably guess wrong and then we would not encounter the bug.
- The type of machine you are using, and the operating system name and version number.

- The command-line arguments you gave to the `configure` command when you installed the interpreter.
- A complete list of any modifications you have made to the interpreter source.

  Be precise about these changes—show a context diff for them.
- Details of any other deviations from the standard procedure for installing Octave.
- A description of what behavior you observe that you believe is incorrect. For example, "The interpreter gets a fatal signal," or, "The output produced at line 208 is incorrect."

  Of course, if the bug is that the interpreter gets a fatal signal, then one can't miss it. But if the bug is incorrect output, we might not notice unless it is glaringly wrong.

  Even if the problem you experience is a fatal signal, you should still say so explicitly. Suppose something strange is going on, such as, your copy of the interpreter is out of synch, or you have encountered a bug in the C library on your system. Your copy might crash and the copy here would not. If you said to expect a crash, then when the interpreter here fails to crash, we would know that the bug was not happening. If you don't say to expect a crash, then we would not know whether the bug was happening. We would not be able to draw any conclusion from our observations.

  Often the observed symptom is incorrect output when your program is run. Unfortunately, this is not enough information unless the program is short and simple. It is very helpful if you can include an explanation of the expected output, and why the actual output is incorrect.
- If you wish to suggest changes to the Octave source, send them as context diffs. If you even discuss something in the Octave source, refer to it by context, not by line number, because the line numbers in the development sources probably won't match those in your sources.

Here are some things that are not necessary:

- A description of the envelope of the bug.

  Often people who encounter a bug spend a lot of time investigating which changes to the input file will make the bug go away and which changes will not affect it. Such information is usually not necessary to enable us to fix bugs in Octave, but if you can find a simpler example to report *instead* of the original one, that is a convenience. Errors in the output will be easier to spot, running under the debugger will take less time, etc. Most Octave bugs involve just one function, so the most straightforward way to simplify an example is to delete all the function definitions except the one in which the bug occurs.

  However, simplification is not vital; if you don't want to do this, report the bug anyway and send the entire test case you used.
- A patch for the bug. Patches can be helpful, but if you find a bug, you should report it, even if you cannot send a fix for the problem.

B.6 Sending Patches for Octave

If you would like to write bug fixes or improvements for Octave, that is very helpful. When you send your changes, please follow these guidelines to avoid causing extra work for us in studying the patches.

If you don't follow these guidelines, your information might still be useful, but using it will take extra work. Maintaining Octave is a lot of work in the best of circumstances, and we can't keep up unless you do your best to help.

- Send an explanation with your changes of what problem they fix or what improvement they bring about. For a bug fix, just include a copy of the bug report, and explain why the change fixes the bug.
- Always include a proper bug report for the problem you think you have fixed. We need to convince ourselves that the change is right before installing it. Even if it is right, we might have trouble judging it if we don't have a way to reproduce the problem.
- Include all the comments that are appropriate to help people reading the source in the future understand why this change was needed.
- Don't mix together changes made for different reasons. Send them *individually*.

  If you make two changes for separate reasons, then we might not want to install them both. We might want to install just one.
- Use '`diff -c`' to make your diffs. Diffs without context are hard for us to install reliably. More than that, they make it hard for us to study the diffs to decide whether we want to install them. Unidiff format is better than contextless diffs, but not as easy to read as '`-c`' format.

  If you have GNU diff, use '`diff -cp`', which shows the name of the function that each change occurs in.
- Write the change log entries for your changes.

  Read the '`ChangeLog`' file to see what sorts of information to put in, and to learn the style that we use. The purpose of the change log is to show people where to find what was changed. So you need to be specific about what functions you changed; in large functions, it's often helpful to indicate where within the function the change was made.

  On the other hand, once you have shown people where to find the change, you need not explain its purpose. Thus, if you add a new function, all you need to say about it is that it is new. If you feel that the purpose needs explaining, it probably does—but the explanation will be much more useful if you put it in comments in the code.

  If you would like your name to appear in the header line for who made the change, send us the header line.

B.7 How To Get Help with Octave

The mailing list `help@octave.org` exists for the discussion of matters related to using and installing Octave. If would like to join the discussion, please send a short note to **`help-request@octave.org`**.

Please do not send requests to be added or removed from the mailing list, or other administrative trivia to the list itself.

If you think you have found a bug in the installation procedure, however, you should send a complete bug report for the problem to `bug@octave.org`. See Section B.5 [Bug Reporting], 頁 309, for information that will help you to submit a useful report.

付記 C Installing Octave

UNIX システムにおいて，スクラッチから Octave をインストールするための手順を紹介します。

- シェルスクリプト ‘`configure`’を実行します。これは，お使いのシステムが持っている（持っていない）機能を判定し，‘`Makefile.in`’という名前のファイルの各々から，‘`Makefile`’という名前のファイルを生成します。

  Octave をビルドするときに，最も頻繁に使用される configure オプションの要約を示します。

  `--prefix=`*prefix*
  : *prefix* 以下のサブディレクトリに Octave をインストールします。*prefix* の初期値は ‘`/usr/local`’です。

  `--srcdir=`*dir*
  : ディレクトリ *dir* から Octave のソースを探します。

  `--with-f2c`
  : Fortran コンパイラが利用可能であっても，`f2c`を使用します。

  `--with-f77`
  : Fortran コードをコンパイルするために `f77`を使用します。使用したいコンパイラ名をオプション引数として指定することもできます。たとえば，`--with-f77=g77` は，Fortran コンパイラ名を `g77`にセットします。

  `--enable-shared`
  : 共有ライブラリを作成します。もし`--enable-lite-kernel`または動的ロード機能を使う予定ならば，おそらくこのオプションを使用したいことでしょう。このオプションにより，‘`.oct`’ファイルはずっと小さくなります。また，あるシステムにおいては，動的リンク関数を使用するために，共有ライブラリをビルドするために必要でしょう。

    お使いのシステムに `libstdc++`の共有版がないときに，それをビルドしたいとも思うかもしれません。HP-PA アーキテクチャで `libstdc++` のバージョン 2.7.2 の共有版をビルドするためには，パッチが必要だということに注意してください。そのパッチは，以下の URL から手に入ります:

    `ftp://ftp.cygnus.com/pub/g++/libg++-2.7.2-hppa-gcc-fix`

  `--enable-dl`
  : `dlopen`を使用し，外部でコンパイルされた関数を動的にリンクする機能を Octave に持たせるようにします。このオプションは，それらの関数を実際にもっているシステムにおいてのみ動作します。もしこの機能を使おうと計画しているならば，おそらく，‘`.oct`’ファイルのサイズを減らすための`--enable-shared`も使用すべきです。

  `--enable-shl`
  : `shl_load`を使用し，外部でコンパイルされた関数を動的にリンクする機能を Octave に持たせるようにします。このオプションは，それらの関数を実際にもっているシステム（HP-UX のみ）においてのみ動作します。もしこの機能を使おうと計画しているならば，おそらく，‘`.oct`’ファイルのサイズを減らすための`--enable-shared`も使用すべきです。

`--enable-lite-kernel`
より小さなカーネルをコンパイルします。現在のところ，このオプションは，動的リンク関数 `dlopen`または `shl_load`を必要とします。また，結果として，コンパイル時ではなくリンク時に関数をロードできるようになります。

`--without-blas`
Octave に含まれている BLAS と LAPACK の汎用バージョンをコンパイルして利用します。初期設定では，configure は，お使いのシステムにある BLAS および LAPACK 行列ライブラリ（ベンダーがチューニングしたライブラリだけでなく，フリーの ATRAS3.0 のような最適化された BLAS 実装を含む）を最初に探します。（最適化 BLAS の利用によって，一般に，行列演算が数倍高速になります。）システムにある BLAS/LAPACK ライブラリが何らかの理由により問題を抱えているときにのみ，このオプションを使用してください。特定の BLAS ライブラリを指定するために`--with-blas=lib`，あるいは configure が自動的にチェックを行わない`-llib`も使用することができます。

`--help` configure スクリプトによって認識されるオプションの要約を表示します。

configure によって使用されるコマンドラインオプションについてのさらなる情報について，ファイル‘`INSTALL`’を参照してください。そのファイルには，ソースが置いてある以外のディレクトリでコンパイルを行うための方法も書かれています。

- make を実行します。

  GNU make の最近のバージョンが必要です。その他の make プログラムで動作するように Octave のメイクファイルを修正することは，おそらく時間の無駄です。かわりに GNU make を入手してコンパイルすることを推奨します。

  プロットするには，お使いのシステムに gnuplot がインストールされていることが必要になります。gnuplot はコマンド駆動型の対話的機能を持つプロットプログラムです。gnuplot は著作権を主張していますが，自由に配布することができます。gnuplot に ‘gnu’ が含まれているのは偶然です—これは GNU プロジェクトあるいは周辺的な意味において FSF とは関連がありません。

  Octave をコンパイルするためには，最近のバージョンの GNU Make が必要になります。`g++` 2.7.2 以降も必要です。バージョン 2.8.0 あるいは `egcs` 1.0.x も動作するでしょう。より新しいバージョンも動作するでしょうが，C++ は未だに発展しているため，何らかのトラブルに巻き込まれることもあり得ます。

  もはや`libg++`が存在する必要はありませんが，`libstdc++`の GNU 実装は存在しなければなりません。`g++` 2.7.2 をお使いならば，`libstdc++`は`libg++`とともに配布されています。しかし，後のバージョンについては，`libstdc++`は別個に配布されています。`egcs` について，`libstdc++` はコンパイラの配布物に含まれています。

  もしパーサを修正する予定があるならば，GNU `bison`および `flex` も必要になります。もしドキュメント類を修正するならば，Octave とともに配布されている `makeinfo`プログラムに対するパッチのほか，Gnu Texinfo が必要です。

  GNU Make，`gcc`，`libstdc++`，`gnuplot`，`bison`，`flex`および Texinfo は，すべて，多数の匿名 ftp アーカイブから入手することができます。主要なサイトは `ftp.gnu.org`ですが，ここはしばしば混んでいます。`ftp.gnu.org`にあるソフトウエアをミラーするサイトの一覧は，`ftp://ftp.gnu.org/pub/gnu/GNUinfo/FTP`からの匿名 ftp により入手できます。

  もし Fortran コンパイラを持っていない，あるいはお使いの Fortran コンパイラが伝統的な UNIX f77 のようには動作しないならば，Fortran を C に翻訳する `f2c`を持っている必要があるでしょ

う。多くの匿名 ftp サイトから `f2c`を得ることができます。`f2c`の最新版は，`netlib.att.com` からいつでも入手できます。

Linux が動作しているアイドル状態の Pentium 133MHz において，何もかもをコンパイルするためには，共有ライブラリをビルドするかどうかによって，1 時間半から 3 時間かかるでしょう。動作には，約 100 メガバイトのディスク容量が必要となります（デバッグシンボル付きでコンパイルしなければ，かなり少なくなります）。デバッグ情報を含めないためには，単に ‘`make`’と入力する代わりに，以下のコマンドを使用してください。

```
make CFLAGS=-O CXXFLAGS=-O LDFLAGS=
```

- もし Octave のコンパイル中にエラーに遭遇したらならば，まず最初に，問題の回避策や解決策があるかどうかを確認するため，既知の問題点のリストをチェックしてください。そうでなければ，バグ報告の仕方についての情報に関して，付記 B [Trouble], 頁 307 を参照してください。
- いちど Octave のコンパイルに成功すれば，‘`make install`’を実行します。

  このコマンドは，Octave のコピー，そのライブラリ，およびそのドキュメントを指定したディレクトリにインストールします。内容を分配するとき，Octave は以下のディレクトリにインストールされます。以下の表において，*prefix* の初期値は ‘`/usr/local`’であり，*version* はインタプリタのカレントバージョン番号を表し，*arch* は Octave をインストールしたコンピュータの種類（たとえば ‘`i586-unknown-gnu`’）です。

  ‘*prefix*`/bin`’
  : 直接実行したいと考えるであろう Octave とその他のバイナリ

  ‘*prefix*`/lib`’
  : libcruft.a や liboctave.a のようなライブラリ

  ‘*prefix*`/share`’
  : アーキテクチャとは独立なデータファイル

  ‘*prefix*`/include/octave`’
  : Octave とともに配布されているインクルードファイル

  ‘*prefix*`/man/man1`’
  : Octave を解説している UNIX 型の man page

  ‘*prefix*`/info`’
  : Octave を解説する info ファイル

  ‘*prefix*`/share/octave/`*version*`/m`’
  : Octave とともに配布されている関数ファイル; これには Octave のバージョンが含まれているので，複数のバージョンの Octave を同時にインストールできます。

  ‘*prefix*`/lib/octave/`*version*`/exec/`*arch*’
  : ユーザではなく Octave によって実行されることになる実行プログラム

  ‘*prefix*`/lib/octave/`*version*`/oct/`*arch*’
  : 動的にロードされるオブジェクトファイル

  ‘*prefix*`/share/octave/`*version*`/imagelib`’
  : Octave とともに配布されている画像ファイル

C.1 インストール上の問題

この節では，Octave をインストールしている間に現れる問題（および，本当は何かが悪いわけではない問題）の一覧を掲載しています。

- SCO システムの一部では，HAVE_TERMIOS_Hが‘config.h’内で定義されているときに，infoがコンパイルに失敗します。‘info/config.h’ から単に定義を取り除くことにより，コンパイルできるようになります。
- configureが dlopen，dlsym，dlcloseおよび dlerrorを発見したが，ヘッダファイル‘dlfcn.h’を見つけられなかったとき，ヘッダファイルのソースを見つけて，それをディレクトリ‘usr/include’にインストールする必要があります。
- ‘.oct’ファイルのビルドが動きません。

  おそらくlibstdc++の共有版を使っているのでしょう。HP-PA アーキテクチャ上でlibstdc++のバージョン 2.7.2 を共有版をビルドするには，パッチが必要です。そのパッチは，以下の URL で見つけることができます。

  ftp://ftp.cygnus.com/pub/g++/libg++-2.7.2-hppa-gcc-fix
- Alpha システムの中には，libdxmlライブラリについて問題をもっているものがあります。結果として，Octave によって呼び出される線形代数ルーチンにおいて浮動小数点エラーまたは segmentation faults が発生します。もしこの問題に遭遇したならば，configure スクリプトを，SPECIAL_MATH_LIBが-ldxmlをセットしないように修正すべきです。
- FreeBSD システムでは，いくつかの内部定数を初期化している間に，Octave がハングするかもしれません。これを修正するには，kernel configuration files（典型的には‘/sys/i386/conf’なるディレクトリにあります）において，

  ```
  options      MATH_EMULATE
  ```

  のかわりに，

  ```
  options      GPL_MATH_EMULATE
  ```

  を使用してください。

  この変更を行った後，カーネルを再構築し，インストール後は再起動する必要があるでしょう。
- ‘sighandlers.cc’をコンパイルしている間に，以下のエラー

  ```
  passing ‘void (*)()’ as argument 2 of
    ‘octave_set_signal_handler(int, void (*)(int))’
  ```

  あるいは

  ```
  warning: ANSI C++ prohibits conversion from ‘(int)’ to ‘(...)’
  ```

  に遭遇したならば，関数に関する適切なプロトタイプを追加するため，gcc のインクルードサブディレクトリにあるいくつかのファイルを編集する必要があります。たとえば，Ultrix 4.2 は，‘signal.h’に signal関数についての適切なプロトタイプと SIG_IGNマクロを追加する必要があります。

  あるシステムでは，SIG_IGNマクロは以下のように定義されます:

  ```
  #define  SIG_IGN  (void (*)())1
  ```

  signal関数に対するプロトタイプ宣言にマッチするために，以下のようなすべきであるときには。

  ```
  #define  SIG_IGN  (void (*)(int))1
  ```

  この変更は，SIG_DFLおよび SIG_ERRシンボルについても行うべきです。さらに，‘sys/signal.h’にある定義を変更する必要があるかもしれません。

gccが一連のヘッダファイルの修正版をインストールするときには，gcc fixincludesおよびfixprotoスクリプトは，おそらく修正されるべきです。しかし，それはまだ行われているとは思えません。

‘/usr/include’にあるファイルを変更すべきではありません。以下のコマンドを実行することにより，gccのインクルードディレクトリツリーを見つけることができます。

```
gcc -print-libgcc-file-name
```

gccインクルードファイルのディレクトリは，通常は，‘libgcc.a’というファイルを含むディレクトリと同じところから始まります。

- Fortran サブルーチンの中には，Sun の Fortran コンパイラの旧版を使用すると，コンパイルに失敗するものがあるかもしれません。サブディレクトリ‘libcruft’にある Fortran サブルーチンをコンパイルしているときに，以下のようなエラーが出るならば，コンパイラをアップグレードし，最適化をオフにしてコンパイルを試みるべきです。

```
zgemm.f:
zgemm:
warning: unexpected parent of complex expression subtree
zgemm.f, line 245: warning: unexpected parent of complex
  expression subtree
warning: unexpected parent of complex expression subtree
zgemm.f, line 304: warning: unexpected parent of complex
  expression subtree
warning: unexpected parent of complex expression subtree
zgemm.f, line 327: warning: unexpected parent of complex
  expression subtree
pcc_binval: missing IR_CONV in complex op
make[2]: *** [zgemm.o] Error 1
```

- NeXT システムにおいて，‘Array.cc’および‘Matrix.cc’をコンパイルしているときに，以下のエラーが出るならば，-gオプションをつけずにこれらファイルを再コンパイルしてください。

```
/usr/tmp/cc007458.s:unknown:Undefined local symbol LBB7656
/usr/tmp/cc007458.s:unknown:Undefined local symbol LBE7656
```

- SunOS システムでは，shell_cmd の呼び出しやページャが動作しないと報告をしてくる人々がいます。これは明らかに，libg++をコンパイルするときに，G_HAVE_SYS_WAITを 1 の代わりに 0 としていることによるものです。

- NeXT システムでは，‘libsys_s.a’をリンクすることが以下の関数を resolve することに失敗するかもしれません。

```
_tcgetattr
_tcsetattr
_tcflow
```

これらは‘libposix.a’の一部です。不幸にも，-posixつきで Octave をリンクすることは，以下の未定義シンボルが生じます。

```
.destructors_used
.constructors_used
_objc_msgSend
_NXGetDefaultValue
_NXRegisterDefaults
.objc_class_name_NXStringTable
.objc_class_name_NXBundle
```

One kluge around this problem は，‘libposix.a’から‘termios.o’ を展開し，それを Octave の‘src’ディレクトリに置き，makefile 中でリンクすべきファイルのリストにそれを加えることです。この問題を解決するための方法について提案があれば歓迎します。

- もし Octave が浮動小数点例外ですぐにクラッシュするなら，無限大と NaN に対する IEEE 浮動小数点値を初期化することに失敗しているようです。

  もしお使いのシステムが，実際には IEEE 演算をサポートしているならば，Octave の内部変数 infinity と NaN を正しく初期化するために，‘lo-ieee.cc’ 内の関数 octave_ieee_initを修正することにより，この問題を修正することができるでしょう。

  もしお使いのシステムが，IEEE 演算をサポートしておらず，Octave の configure スクリプトがそれを正しく決定できていないならば，HAVE_ISINF，HAVE_FINITEおよび HAVE_ISNANを定義しないように，ファイル‘config.h’を編集することにより，この問題を解決することができます。

  いずれのケースにおいても，バグとしてこれを報告してください。なぜならば，行うべき適切なことを自動的に決定するために，Octave の configure スクリプトを修正することが可能かもしれないからです。

- Octave のバイナリ配布物を別のディレクトリにインストールした後では，Emacs の run-octaveコマンドが動作しません。Emacs は inferior-octave-startupの accept-process-output内でハングします。

  これは，comint パッケージを使用したシェルスクリプトを実行することに関する問題であるように思われます。シェルスクリプトについての必要性を消去するために，Octave をインストールする方法を変更することにより，この問題を回避できます。ソース配布物を使用すると Octave のコンパイルとインストールの両方を行うことができます。標準のディレクトリにバイナリ配布物を再インストールしてもよいです。あるいは，Octave のシェルスクリプトラッパにあるコマンドをシェルのスタートアップファイル（および Octave を使用している他の人のシェルスタートアップファイル）にコピーし，ファイル‘octave.bin’を‘octave’にリネームしてもよいです。

付記 D Emacs Octave Support

The development of Octave code can greatly be facilitated using Emacs with Octave mode, a major mode for editing Octave files which can e.g. automatically indent the code, do some of the typing (with Abbrev mode) and show keywords, comments, strings, etc. in different faces (with Font-lock mode on devices that support it).

It is also possible to run Octave from within Emacs, either by directly entering commands at the prompt in a buffer in Inferior Octave mode, or by interacting with Octave from within a file with Octave code. This is useful in particular for debugging Octave code.

Finally, you can convince Octave to use the Emacs info reader for `help -i`.

All functionality is provided by the Emacs Lisp package EOS (for "Emacs Octave Support"). This chapter describes how to set up and use this package.

Please contact <Kurt.Hornik@ci.tuwien.ac.at> if you have any questions or suggestions on using EOS.

D.1 Installing EOS

The Emacs package EOS consists of the three files '`octave-mod.el`', '`octave-inf.el`', and '`octave-hlp.el`'. These files, or better yet their byte-compiled versions, should be somewhere in your Emacs load-path.

If you have GNU Emacs with a version number at least as high as 19.35, you are all set up, because EOS is respectively will be part of GNU Emacs as of version 19.35.

Otherwise, copy the three files from the '`emacs`' subdirectory of the Octave distribution to a place where Emacs can find them (this depends on how your Emacs was installed). Byte-compile them for speed if you want.

D.2 Using Octave Mode

If you are lucky, your sysadmins have already arranged everything so that Emacs automatically goes into Octave mode whenever you visit an Octave code file as characterized by its extension '`.m`'. If not, proceed as follows.

1. To begin using Octave mode for all '`.m`' files you visit, add the following lines to a file loaded by Emacs at startup time, typically your '`~/.emacs`' file:

   ```
   (autoload 'octave-mode "octave-mod" nil t)
   (setq auto-mode-alist
         (cons '("\\.m$" . octave-mode) auto-mode-alist))
   ```

2. Finally, to turn on the abbrevs, auto-fill and font-lock features automatically, also add the following lines to one of the Emacs startup files:

   ```
   (add-hook 'octave-mode-hook
             (lambda ()
               (abbrev-mode 1)
               (auto-fill-mode 1)
               (if (eq window-system 'x)
                   (font-lock-mode 1))))
   ```

   See the Emacs manual for more information about how to customize Font-lock mode.

In Octave mode, the following special Emacs commands can be used in addition to the standard Emacs commands.

C-h m Describe the features of Octave mode.

LFD Reindent the current Octave line, insert a newline and indent the new line (`octave-reindent-then-newline-and-indent`). An abbrev before point is expanded if `abbrev-mode` is non-`nil`.

TAB Indents current Octave line based on its contents and on previous lines (`indent-according-to-mode`).

; Insert an "electric" semicolon (`octave-electric-semi`). If `octave-auto-indent` is non-`nil`, reindent the current line. If `octave-auto-newline` is non-`nil`, automagically insert a newline and indent the new line.

‘ Start entering an abbreviation (`octave-abbrev-start`). If Abbrev mode is turned on, typing *‘C-h* or *‘?* lists all abbrevs. Any other key combination is executed normally. Note that all Octave abbrevs start with a grave accent.

M-LFD Break line at point and insert continuation marker and alignment (`octave-split-line`).

M-TAB Perform completion on Octave symbol preceding point, comparing that symbol against Octave's reserved words and builtin variables (`octave-complete-symbol`).

M-C-a Move backward to the beginning of a function (`octave-beginning-of-defun`). With prefix argument N, do it that many times if N is positive; otherwise, move forward to the N-th following beginning of a function.

M-C-e Move forward to the end of a function (`octave-end-of-defun`). With prefix argument N, do it that many times if N is positive; otherwise, move back to the N-th preceding end of a function.

M-C-h Puts point at beginning and mark at the end of the current Octave function, i.e., the one containing point or following point (`octave-mark-defun`).

M-C-q Properly indents the Octave function which contains point (`octave-indent-defun`).

M-; If there is no comment already on this line, create a code-level comment (started by two comment characters) if the line is empty, or an in-line comment (started by one comment character) otherwise (`octave-indent-for-comment`). Point is left after the start of the comment which is properly aligned.

C-c ; Puts the comment character '`#`' (more precisely, the string value of `octave-comment-start`) at the beginning of every line in the region (`octave-comment-region`). With just *C-u* prefix argument, uncomment each line in the region. A numeric prefix argument N means use N comment characters.

C-c : Uncomments every line in the region (`octave-uncomment-region`).

C-c C-p Move one line of Octave code backward, skipping empty and comment lines (`octave-previous-code-line`). With numeric prefix argument N, move that many code lines backward (forward if N is negative).

`C-c C-n` Move one line of Octave code forward, skipping empty and comment lines (`octave-next-code-line`). With numeric prefix argument *N*, move that many code lines forward (backward if *N* is negative).

`C-c C-a` Move to the 'real' beginning of the current line (`octave-beginning-of-line`). If point is in an empty or comment line, simply go to its beginning; otherwise, move backwards to the beginning of the first code line which is not inside a continuation statement, i.e., which does not follow a code line ending in '`...`' or '\', or is inside an open parenthesis list.

`C-c C-e` Move to the 'real' end of the current line (`octave-end-of-line`). If point is in a code line, move forward to the end of the first Octave code line which does not end in '`...`' or '\' or is inside an open parenthesis list. Otherwise, simply go to the end of the current line.

`C-c M-C-n` Move forward across one balanced begin-end block of Octave code (`octave-forward-block`). With numeric prefix argument *N*, move forward across *n* such blocks (backward if *N* is negative).

`C-c M-C-p` Move back across one balanced begin-end block of Octave code (`octave-backward-block`). With numeric prefix argument *N*, move backward across *N* such blocks (forward if *N* is negative).

`C-c M-C-d` Move forward down one begin-end block level of Octave code (`octave-down-block`). With numeric prefix argument, do it that many times; a negative argument means move backward, but still go down one level.

`C-c M-C-u` Move backward out of one begin-end block level of Octave code (`octave-backward-up-block`). With numeric prefix argument, do it that many times; a negative argument means move forward, but still to a less deep spot.

`C-c M-C-h` Put point at the beginning of this block, mark at the end (`octave-mark-block`). The block marked is the one that contains point or follows point.

`C-c ]` Close the current block on a separate line (`octave-close-block`). An error is signaled if no block to close is found.

`C-c f` Insert a function skeleton, prompting for the function's name, arguments and return values which have to be entered without parens (`octave-insert-defun`).

`C-c C-h` Search the function, operator and variable indices of all info files with documentation for Octave for entries (`octave-help`). If used interactively, the entry is prompted for with completion. If multiple matches are found, one can cycle through them using the standard '`,`' (`Info-index-next`) command of the Info reader.

The variable `octave-help-files` is a list of files to search through and defaults to `'("octave")`. If there is also an Octave Local Guide with corresponding info file, say, '`octave-LG`', you can have `octave-help` search both files by

```
(setq octave-help-files '("octave" "octave-LG"))
```

in one of your Emacs startup files.

A common problem is that the ⟨RET⟩ key does *not* indent the line to where the new text should go after inserting the newline. This is because the standard Emacs convention

is that ⟨RET⟩ (aka *C-m*) just adds a newline, whereas ⟨LFD⟩ (aka *C-j*) adds a newline and indents it. This is particularly inconvenient for users with keyboards which do not have a special ⟨LFD⟩ key at all; in such cases, it is typically more convenient to use ⟨RET⟩ as the ⟨LFD⟩ key (rather than typing *C-j*).

You can make ⟨RET⟩ do this by adding

```
(define-key octave-mode-map "\C-m"
  'octave-reindent-then-newline-and-indent)
```

to one of your Emacs startup files. Another, more generally applicable solution is

```
(defun RET-behaves-as-LFD ()
  (let ((x (key-binding "\C-j")))
    (local-set-key "\C-m" x)))
(add-hook 'octave-mode-hook 'RET-behaves-as-LFD)
```

(this works for all modes by adding to the startup hooks, without having to know the particular binding of ⟨RET⟩ in that mode!). Similar considerations apply for using ⟨M-RET⟩ as ⟨M-LFD⟩. As Barry A. Warsaw <bwarsaw@cnri.reston.va.us> says in the documentation for his `cc-mode`, "This is a very common question. `:-)` If you want this to be the default behavior, don't lobby me, lobby RMS!"

The following variables can be used to customize Octave mode.

`octave-auto-indent`
: Non-`nil` means auto-indent the current line after a semicolon or space. Default is `nil`.

`octave-auto-newline`
: Non-`nil` means auto-insert a newline and indent after semicolons are typed. The default value is `nil`.

`octave-blink-matching-block`
: Non-`nil` means show matching begin of block when inserting a space, newline or '`;`' after an else or end keyword. Default is `t`. This is an extremely useful feature for automatically verifying that the keywords match—if they don't, an error message is displayed.

`octave-block-offset`
: Extra indentation applied to statements in block structures. Default is 2.

`octave-continuation-offset`
: Extra indentation applied to Octave continuation lines. Default is 4.

`octave-continuation-string`
: String used for Octave continuation lines. Normally '\'.

`octave-mode-startup-message`
: If `t` (default), a startup message is displayed when Octave mode is called.

If Font Lock mode is enabled, Octave mode will display

- strings in `font-lock-string-face`
- comments in `font-lock-comment-face`
- the Octave reserved words (such as all block keywords) and the text functions (such as '`cd`' or '`who`') which are also reserved using `font-lock-keyword-face`

- the builtin operators (‘&&’, ‘<>’, ...) using `font-lock-reference-face`
- the builtin variables (such as ‘`warn_fortran_indexing`’, ‘`NaN`’ or ‘`LOADPATH`’) in `font-lock-variable-name-face`
- and the function names in function declarations in `font-lock-function-name-face`.

There is also rudimentary support for Imenu (currently, function names can be indexed).

You can generate TAGS files for Emacs from Octave ‘`.m`’ files using the shell script `otags` that is installed alongside your copy of Octave.

Customization of Octave mode can be performed by modification of the variable `octave-mode-hook`. If the value of this variable is non-`nil`, turning on Octave mode calls its value.

If you discover a problem with Octave mode, you can conveniently send a bug report using *C-c C-b* (`octave-submit-bug-report`). This automatically sets up a mail buffer with version information already added. You just need to add a description of the problem, including a reproducible test case and send the message.

D.3 Running Octave From Within Emacs

The package ‘`octave`’ provides commands for running an inferior Octave process in a special Emacs buffer. Use

```
M-x run-octave
```

to directly start an inferior Octave process. If Emacs does not know about this command, add the line

```
(autoload 'run-octave "octave-inf" nil t)
```

to your ‘`.emacs`’ file.

This will start Octave in a special buffer the name of which is specified by the variable `inferior-octave-buffer` and defaults to `"*Inferior Octave*"`. From within this buffer, you can interact with the inferior Octave process ‘as usual’, i.e., by entering Octave commands at the prompt. The buffer is in Inferior Octave mode, which is derived from the standard Comint mode, a major mode for interacting with an inferior interpreter. See the documentation for `comint-mode` for more details, and use *C-h b* to find out about available special keybindings.

You can also communicate with an inferior Octave process from within files with Octave code (i.e., buffers in Octave mode), using the following commands.

C-c i l Send the current line to the inferior Octave process (`octave-send-line`). With positive prefix argument *N*, send that many lines. If `octave-send-line-auto-forward` is non-`nil`, go to the next unsent code line.

C-c i b Send the current block to the inferior Octave process (`octave-send-block`).

C-c i f Send the current function to the inferior Octave process (`octave-send-defun`).

C-c i r Send the region to the inferior Octave process (`octave-send-region`).

C-c i s Make sure that ‘inferior-octave-buffer’ is displayed (`octave-show-process-buffer`).

C-c i h Delete all windows that display the inferior Octave buffer (`octave-hide-process-buffer`).

`C-c i k` Kill the inferior Octave process and its buffer (`octave-kill-process`).

The effect of the commands which send code to the Octave process can be customized by the following variables.

`octave-send-echo-input`
: Non-`nil` means echo input sent to the inferior Octave process. Default is `t`.

`octave-send-show-buffer`
: Non-`nil` means display the buffer running the Octave process after sending a command (but without selecting it). Default is `t`.

If you send code and there is no inferior Octave process yet, it will be started automatically.

The startup of the inferior Octave process is highly customizable. The variable `inferior-octave-startup-args` can be used for specifying command lines arguments to be passed to Octave on startup as a list of strings. For example, to suppress the startup message and use 'traditional' mode, set this to `'("-q" "--traditional")`. You can also specify a startup file of Octave commands to be loaded on startup; note that these commands will not produce any visible output in the process buffer. Which file to use is controlled by the variable `inferior-octave-startup-file`. If this is `nil`, the file '`~/.emacs-octave`' is used if it exists.

And finally, `inferior-octave-mode-hook` is run after starting the process and putting its buffer into Inferior Octave mode. Hence, if you like the up and down arrow keys to behave in the interaction buffer as in the shell, and you want this buffer to use nice colors, add

```
(add-hook 'inferior-octave-mode-hook
          (lambda ()
            (turn-on-font-lock)
            (define-key inferior-octave-mode-map [up]
              'comint-previous-input)
            (define-key inferior-octave-mode-map [down]
              'comint-next-input)))
```

to your '`.emacs`' file. You could also swap the roles of `C-a` (`beginning-of-line`) and `C-c C-a` (`comint-bol`) using this hook.

> **Note:** If you set your Octave prompts to something different from the defaults, make sure that `inferior-octave-prompt` matches them. Otherwise, *nothing* will work, because Emacs will have no idea when Octave is waiting for input, or done sending output.

D.4 Using the Emacs Info Reader for Octave

You can also set up the Emacs Info reader for dealing with the results of Octave's '`help -i`'. For this, the package '`gnuserv`' needs to be installed, which unfortunately still does not come with GNU Emacs (it does with XEmacs). It can be retrieved from any GNU Emacs Lisp Code Directory archive, e.g. `ftp://ftp.cis.ohio-state.edu/pub/gnu/emacs/elisp-archive`, in the '`packages`' subdirectory. A recent version of gnuserv is available from `http://www.meltin.net/hacks/emacs/src/gnuserv-3.12.2.tar.gz`.

If ‘gnuserv’ is installed, add the lines

```
(autoload 'octave-help "octave-hlp" nil t)
(require 'gnuserv)
(gnuserv-start)
```

to your ‘.emacs’ file.

You can use either ‘plain’ Emacs Info or the function octave-help as your Octave info reader (for ‘help -i’). In the former case, set the Octave variable INFO_PROGRAM to "info-emacs-info". The latter is perhaps more attractive because it allows to look up keys in the indices of *several* info files related to Octave (provided that the Emacs variable octave-help-files is set correctly). In this case, set INFO_PROGRAM to "info-emacs-octave-help".

If you use Octave from within Emacs, these settings are best done in the ‘~/.emacs-octave’ startup file (or the file pointed to by the Emacs variable inferior-octave-startup-file).

付記 E Grammar

Someday I hope to expand this to include a semi-formal description of Octave's language.

E.1 Keywords

The following identifiers are keywords, and may not be used as variable or function names:

```
all_va_args             endwhile
break                   for
case                    function
catch                   global
continue                gplot
else                    gsplot
elseif                  if
end                     otherwise
end_try_catch           return
end_unwind_protect      switch
endfor                  try
endfunction             unwind_protect
endif                   unwind_protect_cleanup
endswitch               while
```

The following command-like functions are also speical. They may be used as simple variable names, but not as formal parameters for functions, or as the names of structure variables. Failed assignments leave them undefined (you can recover the orginal definition as a function using clear).

```
casesen       echo          load          show
cd            edit_history  ls            type
chdir         format        more          which
clear         help          run_history   who
diary         history       save          whos
dir           hold          set
```

付記 F GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software

which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) yyyy  name of author

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

Concept Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Variable Index

P

R

S

T

V

W

Function Index

=

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Operator Index