MATLAB®

The Language of Technical Comput-

Computation

Visualization

Programming

The MATH WORKS Inc.

Using MATLAB

Version 6

サイバネットシステム株式会社との連絡法

03-5978-5411	電話
03-5978-5440	ファクシミリ
T 東京都文京区大塚 2 丁目 15 番地 6 号 ニッセイ音羽ビル	Mail
http://www.cybernet.co.jp	Web
infomatlab@cybermet.co.jp	営業部
techmatlab@cybermet.co.jp	技術部

Using MATLAB

© COPYRIGHT 1984 - 2000 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by or for the federal government of the United States. By accepting delivery of the Program, the government hereby agrees that this software qualifies as "commercial" computer software within the meaning of FAR Part 12.212, DFARS Part 227.7202-1, DFARS Part 227.7202-3, DFARS Part 252.227-7013, and DFARS Part 252.227-7014. The terms and conditions of The MathWorks, Inc. Software License Agreement shall pertain to the government¤s use and disclosure of the Program and Documentation, and shall supersede any conflicting contractual terms or conditions. If this license fails to meet the government¤s minimum needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to MathWorks.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and Target Language Compiler is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History:

December 1996	First printing	First Printing for MATLAB 5
June 1997	Second printing	Revised for MATLAB 5.1
January 1998	Third printing	Revised for MATLAB 5.2
January 1999	Fourth printing	Revised for MATLAB 5.3 (Release 11)
November 2000	Fifth printing	Revised for MATLAB 6 (Release 12)
March 2001	翻訳	

作業環境

MATLAB® 作業環境は、MATLAB 関数、かファイルを使用する場合に役立つツールセットです。開発環境を構成するツールの多くが、グラフィカルユーザインタフェースで提供されています。

基本知識

- "MATLAB の起動と終了 " – MATLAB の実行方法、スタートアップオプション、シャットダウンオプションについて説明します。
- " デスクトップの使用 " – MATLA デスクトップは､MATLAB を起動したときに最初に表示されます。MATLAB デスクトップでは、ここで説明される Launch Pad などのその他ツールを制御します。
- "MATLAB 関数の実行 " – コマンドウィンドウおよびコマンド履歴ウィンドウでの作業について説明します。
- " ヘルプの参照 " – ヘルプブラウザ、ヘルプ関数、マニュアルの印刷、ヘルプを参照するためのその他の手法について説明します。

その他の作業環境ツール

- " ワークスペース、パス、ファイル操作 " – ワークスペースブラウザ、配列エディタ、パスブラウザ、カレントディレクトリブラウザ、及びこれらと同等の機能を提供する関数について説明します。
- " データのインポートとエクスポート " – 他のアプリケーションで作成したデータを MATLAB ワークスペースに読み込んだり、MATLAB ワークスペース変数を他のアプリケーションで使うための手法を紹介します。インポートウィザードについても説明します。
- "M-ファイルのエディットとデバッグ" – M-ファイル(MATLAB関数からなるプログラムファイル) の作成、変更に使用する MATLAB のグラフィカルエディタ / デバッガ、デバッグ関数について説明します。
- "M- ファイルのパフォーマンスを改善 – プロファイラ " – M- ファイルの処理にどれだけの時間がかかっているのかを計測するツールを紹介します。このツールを使って、実行速度を改善することができます。

- " ソースコントロールシステムとのインタフェース " – MATLAB 内部からユーザのソースコントロールシステム (Simulink、Stateflow など) にアクセスします。
- " ノートブックの使用 " – Microsoft Word のようなワードプロセッサから MATLAB の数値計算、画像表示を行うソフトウェアにアクセスします。

1

MATLAB の起動と終了

MATLAB の起動 1-3
Windows プラットフォームで MATLAB の起動 1-3
UNIX プラットフォームで MATLAB の起動 1-3
MATLAB での起動ディレクトリ 1-3
スタートアップオプション 1-4
ツールボックスのパスのキャッシュによる起動時間の削減 . . .1-9

MATLAB の終了 1-14
MATLAB 終了時にスクリプトの実行 1-14

この節では、MATLAB を起動、終了の方法と、起動、終了時に関連付けるオプションの設定法も含めて解説します。

- 1-3 ページの "Windows プラットフォームで、MATLAB の起動 "
- 1-3 ページの "UNIX プラットフォームで MATLAB の起動 "
- 1-3 ページの "MATLAB の起動ディレクトリ "
- 1-4 ページの " スタートアップオプション "
- 1-9 ページの " ツールボックスパスキャッシュと共に起動時間を短縮 "
- 1-14 ページの "MATLAB の終了 "
- 1-14 ページの "MATLAB 終了時にスクリプトを実行 "

MATLAB の起動

MATLAB の起動方法は、ユーザの使用するプラットフォームによって異なります。

Windows プラットフォームで、MATLAB の起動

Microsoft Windows プラットフォームで、MATLAB を起動するには、Windows デスクトップ上の MATLAB ショートカットアイコン をクリックします。ショートカットは、MATLAB をインストールすると、インストールしたディレクトリに自動的に作成されます。

DOS ウインドウから MATLAB を起動するには、DOS プロンプトで、matlab と入力します。

MATLAB を起動すると、MATLAB デスクトップが開きます。MATLAB デスクトップは、第 2 章，" デスクトップの使用 " を参照してください。

UNIX プラットフォームで MATLAB の起動

UNIX プラットフォームで MATLAB を起動するには、オペレーションシステムプロンプトで、matlab と入力します。

MATLAB を起動すると、MATLAB デスクトップが開きます。第 2 章，" デスクトップの使用 " を参照してください。UNIX プラットフォーム上では、DISPLAY 環境変数が、設定されていなかったり、不適切な値になっていると、デスクトップは開きません。また、UNIX プラットフォームの中には、デスクトップをサポートしていないものもあります。詳細については、*R12 Release Notes* を参照してください。

MATLAB の起動ディレクトリ

MATLAB の最初のカレントディレクトリは、プラットフォームとインストール操作によって決定されます。異なるディレクトリで、MATLAB を起動させることも可能です。

Windows プラットフォームでの起動ディレクトリ

Windows プラットフォームでは、MATLAB をインストールした場合、デフォルトの起動ディレクトリとして、$matlabroot\work を設定します。ここで、$matlabroot は、MATLAB ファイルがインストールされたディレクトリです。

UNIX プラットフォームでの起動ディレクトリ

UNIX プラットフォームでは、初期のカレントディレクトリは、MATLAB を呼び出したときに UNIX ファイルシステム上でユーザが配置していたディレクトリに設定さ

れます。

起動ディレクトリの変更

デフォルトのディレクトリとは、異なる様々なディレクトリから MATLAB を起動させることができます。ユーザが、指定したディレクトリを MATLAB 起動時の作業ディレクトリにするには、つぎの操作を行ってください。

1 startup.m ファイルを作成します。1-5 ページの "MATLAB のスタートアップファイル startup.m の使用 " を参照してください。

2 startup.m ファイル内に新しいディレクトリに変更するための関数 cd を設定します。

3 startup.m ファイルをカレントスタートアップディレクトリに格納します。

Windows プラットフォームのみ有効 . Windows プラットフォームでは、つぎの操作で起動ディレクトリを変更することもできます。

1 MATLAB ショートカットアイコンを右クリックし、コンテキストメニューから、**プロパティ (Properties)** を選択します。

matlab.exe のプロパティ（**Properties**）ダイアログボックスが、**ショートカット (Shortcut)** ページに開きます。

2 新しい起動ディレクトリを、**スタート (Start in)** フィールドに入力して、**OK** を選択します。

つぎに、ショートカットアイコンを使って、MATLAB を起動すると、ステップ 2 で指定したディレクトリがカレントディレクトリに設定されます。

MATLAB を起動するために、複数のショートカットを作成し、そのそれぞれの起動ディレクトリが異なり、各ディレクトリに異なるスタートアップオプションを設定する、というような操作も行うことができます。

スタートアップオプション

起動時に、MATLAB がいくつかの操作を自動的に行うようにするためのスタートアップオプションを定義することができます。MATLAB にスタートアップオプションを定義する方法としては、つぎの 2 つの方法があります。

- 1-5 ページの "MATLAB のスタートアップファイル startup.m の使用 "

- 1-5 ページの "Windows プラットフォームにスタートアップオプションの追加 " または 1-7 ページの "UNIX プラットフォームにスタートアップオプションの追加 "

MATLAB のスタートアップファイル startup.m の使用

起動時に、MATLAB は自動的にマスタ　M- ファイル matlabrc.m を実行し、存在している場合、スタートアップファイル startup.m を起動します。local ディレクトリに格納されているファイル matlabrc.m は、MathWorks(または、マルチユーザシステムの場合は、ユーザのシステム管理者) によって、使用が制限されているファイルです。

ファイル startup.m は、スタートアップオプションをユーザが指定するために使用するファイルです。ユーザは、このファイルを使って、デフォルトサーチパスを変更したり、ユーザワークスペース内の変数をあらかじめ定義したり、Handle Graphics® のデフォルト値を定義することができます。たとえば、つぎのようなラインからなる startup.m ファイルを作成すると、

```
addpath \home\me\mytools
cd \home\me\mytools
```

ユーザのデフォルトサーチパスに、\home\me\mytools が付加され、そのディレクトリをカレントの起動ディレクトリに設定します。

Windows プラットフォームでは、startup.m ファイルは、$matlabroot\toolbox\local ディレクトリに格納してください。なお、ここで、$matlabroot は、MATLAB がインストールされたディレクトリです。

UNIX ワークステーションでは、startup.m ファイルは、ホームディレクトリの matlab ディレクトリにおいてください (例： ~/matlab)。

Windows プラットフォームにスタートアップオプションの追加

選択したスタートアップオプション (コマンドフラグとも呼びます) を MATLAB の Windows ショートカットに対するターゲットパスに付加することができます。また、DOS ウィンドウから MATLAB を起動するときのオプションを付加することも可能です。スタートアップオプションを追加するには、つぎの操作を行ってください。

1 MATLAB ショートカットアイコン を右クリックして、コンテキストメニューから、**プロパティ (Properties)** を選択してください。

matlab.exe の**プロパティ (Properties)** ダイアログボックスが、**ショートカット (Shortcut)** パネルに開きます。

2 **ターゲット (Target)** フィールドに表示される matlab.exe のターゲットパスに、つぎにリスト表示されている入力可能なスタートアップションを1つ以上追加してください。

オプション	詳細
/automation	MATLAB をオートメーションサーバとして起動します。すなわち、最小化の状態、MATLAB スプラッシュスクリーンを表示することなく、MATLAB を起動します。詳細については、*Application Program Interface Guide*のクライアント/サーバアプリケーションを参照してください。
/logfile logfilename	MATLAB からの出力を、指定したログファイルに自動的に書きこみます。
/minimize	MATLAB スプラッシュスクリーンを表示せずに、MATLAB を最小化して起動します。
/nosplash	MATLAB スプラッシュスクリーンを表示せずに、MATLAB を起動します。
/r M_file	MATLAB を起動した直後に、指定した M- ファイルを自動的に実行します。これをバッチモードで、MATLAB を実行するといいます。
/regserver	MATLAB に合った適切な ActiveX エントリを加える形で Windows のレジストリを変更します。詳細は、*Application Program Interface Guide* のクライアント/サーバアプリケーションを参照してください。
/unregserver	MATLAB に対する ActiveX エントリを除去する形で Windows のレジストリを変更します。詳細は、*Application Program Interface Guide* のクライアント/サーバアプリケーションを参照してください。

3 OK をクリックしてください。

例題 – M- ファイルを自動実行するスタートアップオプションの設定 . MATLAB を起動し、自動的に M- ファイル results.m を実行するには、ユーザの Windows ショートカットに対するつぎのターゲットパスを使用してください。

```
D:\matlabr12\bin\win32\matlab.exe /r results
```

DOS ウインドウから MATLAB を実行する場合のスタートアップオプション . MATLAB を DOS ウインドウから実行する場合は、先の表に記載されたスタートアップオプションを MATLAB スタートアップ関数 matlab の後ろに追加してください。

たとえば、MATLAB を起動し、自動的に M- ファイル results.m を実行するには、つぎのようにタイプしてください。

```
matlab /r results
```

UNIX プラットフォームにスタートアップオプションの追加

スタートアップオプション (コマンドフラグとも呼びます) を、matlab スタートアップ関数の後に挿入しますことができます。UNIX 上で指定できるスタートアップオプションは、つぎの表のとおりです。

オプション	詳細
-arch	アーキテクチャー arch を仮定して、MATLAB を実行
-arch/ext	拡張子 ext を伴うバージョンの MATLAB を実行する。また、もし存在していれば、仮想アーキテクチャー arch を実行する
-c licensefile	LM_LICENSE_FILE を licensefile に設定する。このファイルは、port@host という形式をとります。
-Ddebugger [options]	指定したデバッガを MATLAB と共に起動します。
-debug	MATLAB の内部デバッギングを有効にします。
-display Xserver	X コマンドを Xserver に送ります。
-ext	拡張子 ext が伴うバージョンの MATLAB が、存在していれば、それを実行します。
-h or -help	スタートアップオプションを表示します。

オプション	詳細 (Continued)
-mwvisual visualid	フィギュアウィンドウに、使用するデフォルトの X ビジュアルを指定します。
-n	環境変数のみを印刷
-nodesktop	MATLABデスクトップを開かずにMATLABを起動します。このオプションを使うと、X- ウィンドウなし（例：VT100 モード）か、またはバッチ処理モードで、MATLAB を実行することができます。但し、> コンストラクタを使って、MATLAB と連結した場合には、nodesktop オプションが自動的に使用されます。 nodesktop オプションでは、ヘルプブラウザなどの作業環境ツールは依然として使用することが出来ます。関数 helpbrowser を使って起動する、ヘルプブラウザのような作業環境ツールを使用することもできます。 コマンドラインのインタフェースに切り替えるために、nodesktop オプションを使ってはいけません。デスクトップツールではなく、コマンドラインから操作を行いたい場合、**View -> Desktop Layout -> Command Window Only** を選択してください。
-nojvm	Java VM をロードせずに MATLAB を起動します。これによってメモリ使用量が最小化され、最初の起動スピードも改善されます。nojvm オプションを使用すると、デスクトップのみならず Java を必要とする全てのツールが使用できなくなります。ここでは、*R12 リリースノート*の UNIX プラットフォームでの制限に記載されたものと同一の制限が適用されます。
-nosplash	起動中にスプラッシュスクリーンを表示せずに MATLAB を起動します。

たとえばスプラッシュスクリーンなしで MATLAB を起動するには、つぎのように入力してください。

```
matlab -nosplash
```

ツールボックスパスキャッシュと共に起動時間を短縮

ネットワークサーバから MATLAB を実行する場合、MATLAB ツールボックスパスキャッシュを使うと、起動時間を大きく削減することができます。ツールボックスパスキャッシュは、MATLAB ルートディレクトリの下にあるツールボックスディレクトリのパス情報を全て格納します。起動時に、リモートファイルシステムから読むのではなく、MATLAB は、キャッシュから情報を読み込みます。

ツールボックスパスキャッシュは、ユーザの MATLAB セッションの起動中のみ使用できます。ツールボックスディレクトリの多くを含むようにユーザの MATLAB パスを定義している場合、特に、この機能は有用な機能となります。リモートファイルシステムからディレクトリをスキャンし、パス情報を全て取得するには、かなりの時間を要します。しかし、この情報を前もって作成されたキャッシュから読み込めば、パス情報の取得に要する時間は、飛躍的に短くなります。これに対し、短いツールボックスパスを使っている場合、キャッシュを使うメリットは小さくなりますが、依然として時間の短縮効果は得られます。

ユーザのファイルがリモートシステムから供給されない、ローカルディスク上で MATLAB を実行している場合、キャッシュを使っても起動時間はほとんど短縮されません。この場合、ツールボックスパスキャッシュは無効にしておいてください。

ツールボックスパスキャッシュの設定を選択

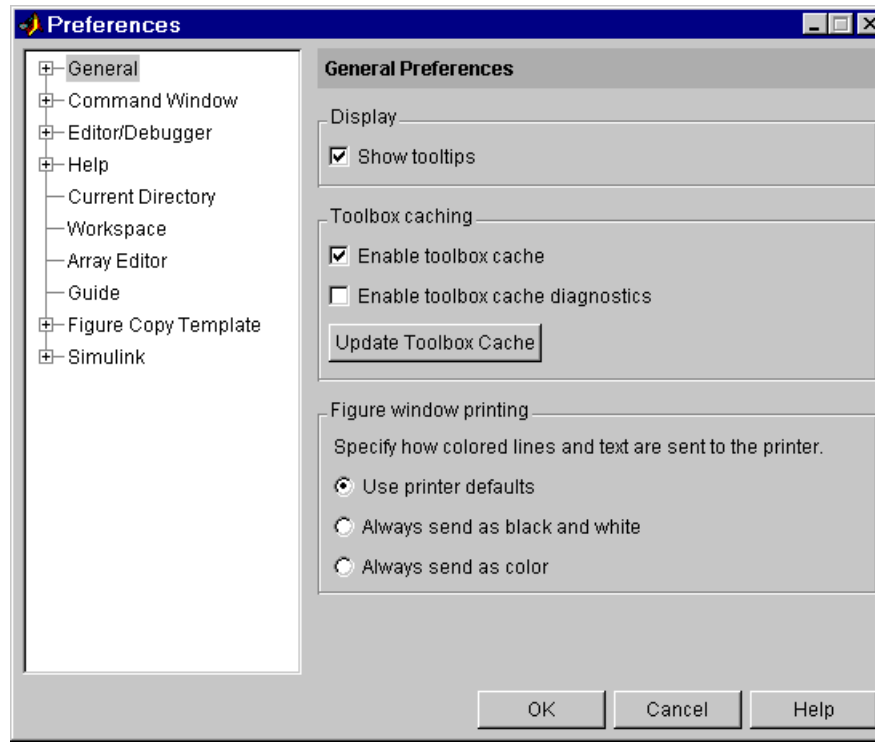
MATLAB の設定 (Preferences) ウィンドウからツールボックスパスキャッシュを作成し、有効にすることができます。

1 MATLABデスクトップから、**ファイル(File)**メニューから**設定(Preferences)**を選択します。

  設定 (Preferences) ダイアログボックスが開きます。

2 ダイアログボックスの左枠で、**General** をクリックすると、下図の **General**

Preferences パネルが右枠に表示されます。



3 キャッシュを使用する場合、または、キャッシュを再び作成する場合は、Toolbox caching の Enable toolbox cache を選択してください。

Enable toolbox cache が選択されると、MATLAB は、起動中に、どのような操作をしているかについての簡単な表示を行います。MATLAB がツールボックスディレクトリを、いつキャッシュからロードしているかを、これによって知ることができます。また、ツールボックスパスキャッシュが見つからない場合、MATLAB は警告表示を行います。

4 詳細については、Enable toolbox cache diagnostics. を選択してください。

5 **Enable toolbox cache diagnostics** が選択されると、MATLAB は起動時に付加的な情報を表示します。

MATLAB は、起動時にキャッシュからではなく、リモートファイルシステムからツールボックスディレクトリが追加されると常に警告を表示します。ツールボックスディレクトリの変更にキャッシュ内容の更新が追いついていない場合、このようなことが起こります。

6 キャッシュの新しいコピーを作成するためには、**Update Toolbox Cache** を選択します。詳細は、1-11 ページの " ツールボックスパスキャッシュの作成 " を参照してください。

これを利用するには、キャッシュファイルをもつディレクトリのアクセス権が必要です。書き込みアクセス権をもたない場合は、ボタンはグレイになります。

7 **OK** をクリックします。

ツールボックスパスキャッシュの作成

ツールボックスパスキャッシュは、MATLAB ファイルを供給するシステムの toolbox/local ディレクトリに位置する MAT- ファイルに格納されています。このシステムに、最初に MATLAB をインストールした場合、ユーザ、または、ユーザのシステム管理者がキャッシュを作成する必要があります (MATLAB は、前もって設定したキャッシュを用意していません)。ツールボックスディレクトリが追加された場合、または、削除された場合、常にキャッシュファイルを再び作成し直さなければなりません。作成し直さない場合、キャッシュには古いパス情報が保存されてしまうことになります。そうするためには、1-9 ページの " ツールボックスパスキャッシュの設定を選択 " で説明したように **Update Toolbox Cache** を選択します。

等価な関数 . キャッシュをアップデートするには、コマンドウィンドウに以下を入力します。

```
rehash toolboxcache
```

キャッシュを有効にします

MATLAB が最初にインストールされた時点では、ツールボックスパスキャッシュ機能は無効になっています。分散型ファイルシステム上でキャッシュを有効にするには、つぎのように操作してください。

1 システム管理者は、キャッシングを有効にし、MATLAB ファイルを供給するシステム上に初期ツールボックスパスキャッシュを作成することができます。この

点については、1-11 ページの " ツールボックスパスキャッシュの作成 " のとおりです。

2 ツールボックスディレクトリパスキャッシュを使いたいユーザが、自分のシステム上でキャッシュを有効にすることができます。

分散型でないシステム上では、システムユーザが上記の操作を 2 つとも行うことができます。

ツールボックスパスキャッシングを有効にするには、1-9 ページの " ツールボックスパスキャッシュの設定を選択 " を参照してください。

ツールボックスパスキャッシュ機能を無効にするには、ユーザベースで行う方法と、グローバルベースで行う方法の 2 つの方法があります。ユーザベースで無効にするには、**設定 (Preferences)** ダイアログボックスの**ツールボックスキャッシュの使用可 (Enable toolbox cache)** のチェックボックスを外してください。グローバルベースで無効にするには、toolbox/local ディレクトリから toolbox_cache.mat を削除してください。

キャッシュを更新

ツールボックスディレクトリを変更すると、ツールボックスパスキャッシュファイルの内容は時代遅れのものとなってしまいます。この理由のために、ユーザは、キャッシュを更新する必要があるという内容の警告表示を見る場合があります。この節では、いつユーザのツールボックスパスキャッシュを更新する必要があるのか、そして、時代遅れのキャッシュファイルによって引き起こされる問題を回避するには、どうすれば良いのかについて解説します。

製品のインストール、または、アップデートに伴いキャッシュを更新．I 新しいツールボックスをインストールしたり、MathWorks からのアップデートファイルをインストールすると、ツールボックスパスキャッシュに格納されている情報は、もはや、ユーザのツールボックスディレクトリを正確に反映したものではなくなってしまいます。そのため、インストールをしている最中に、MATLAB インストーラは、キャッシュの内容が無効であると認識します。

ユーザのキャッシュが無効であると認識されると、MATLAB は起動中にキャッシュの使用を中止し、ファイルシステムを使って、ディレクトリにアクセスすることによってパス情報を収集します。MATLAB は起動時に、ユーザのツールボックスパスキャッシュの内容が時代遅れのものであるため、パスキャッシュを参照せずに、起動操作を行っている、いう内容の警告メッセージを表示します。キャッシュの使用をもう 1 度有効にするには、前述の方法 (1-11 ページの " ツールボックスパスキャッシュの作成 ") でキャッシュを再び作成し直す必要があります。

ツールボックスディレクトリの変更に伴いキャッシュの更新． I(たとえば、ファイルを追加、削除しますといった操作で)MATLAB ツールボックスディレクトリを変更すると、ツールボックスパスキャッシュファイルに格納されたパス情報は現状を反映しない内容になってしまいます。ここで、ユーザが、つぎに MATLAB セッションを起動すると、キャッシュに格納された古いパス情報がメモリーにロードされてしまいます。この時にも、MATLAB は警告メッセージを表示します。

ツールボックスパスに影響を与える変更を行う場合、前述の方法 (1-3 ページの "Windows プラットフォームでの起動ディレクトリ ") でキャッシュファイルを再び作成しなおさなくてはなりません。

注意 MathWorks は、ツールボックスパスキャッシュが有効になっている場合、ツールボックスエリアで開発作業を行わないようにすることを強くお勧めします。開発作業でツールボックスエリアを使って、後でキャッシュを再び作成しないでおくと、MATLAB は次の MATLAB セッションでツールボックスディレクトリに格納された不正確なレコードを使用する可能性があります。その結果、MATLAB は、ユーザが追加した新しいファイルを配置することができなくなってしまいます。

MATLAB の終了

MATLAB を終了するには、つぎのいずれかの操作を行います。

- MATLAB デスクトップのクローズボックス を選択します。
- デスクトップの**ファイル(File)**メニューから**MATLABの終了(Exit MATLAB)**を選択します。
- コマンドウィンドウプロンプトで、quit と入力します。

MATLAB 終了時にスクリプトを実行

MATLAB 終了時に、MATLAB はスクリプト finish.m がカレントディレクトリ、または、MATLAB サーチパスの任意の場所に存在すれば、このスクリプト finish.m を実行します。先ず、ファイル finish.m を作成してください。このファイルは、MATLAB 終了時に実行する関数 (たとえば、ワークスペースをセーブする関数、確認ダイアログボックスを表示する関数など)から構成されています。$matlabroot/toolbox/local に格納された 2 つのサンプルファイルは、ユーザが自分自身の finish.m ファイルを作成するためのベースとして使用してください。

- finishsav.m – MATLAB 終了時に、ワークスペースを MAT- ファイルにセーブする save 関数が含まれています。
- finishdlg.m – 終了をキャンセルするか、どうかを訊ねる確認ダイアログボックスを表示します。

2

デスクトップの使用

デスクトップツール 2-4
Launch Pad 2-5

デスクトップの設定 2-7
デスクトップツールのオープンとクローズ 2-7
ウィンドウのサイズの変更 2-9
ウィンドウの移動 2-10
既定のレイアウトで MATLAB デスクトップ上にツールを配置する 2-16

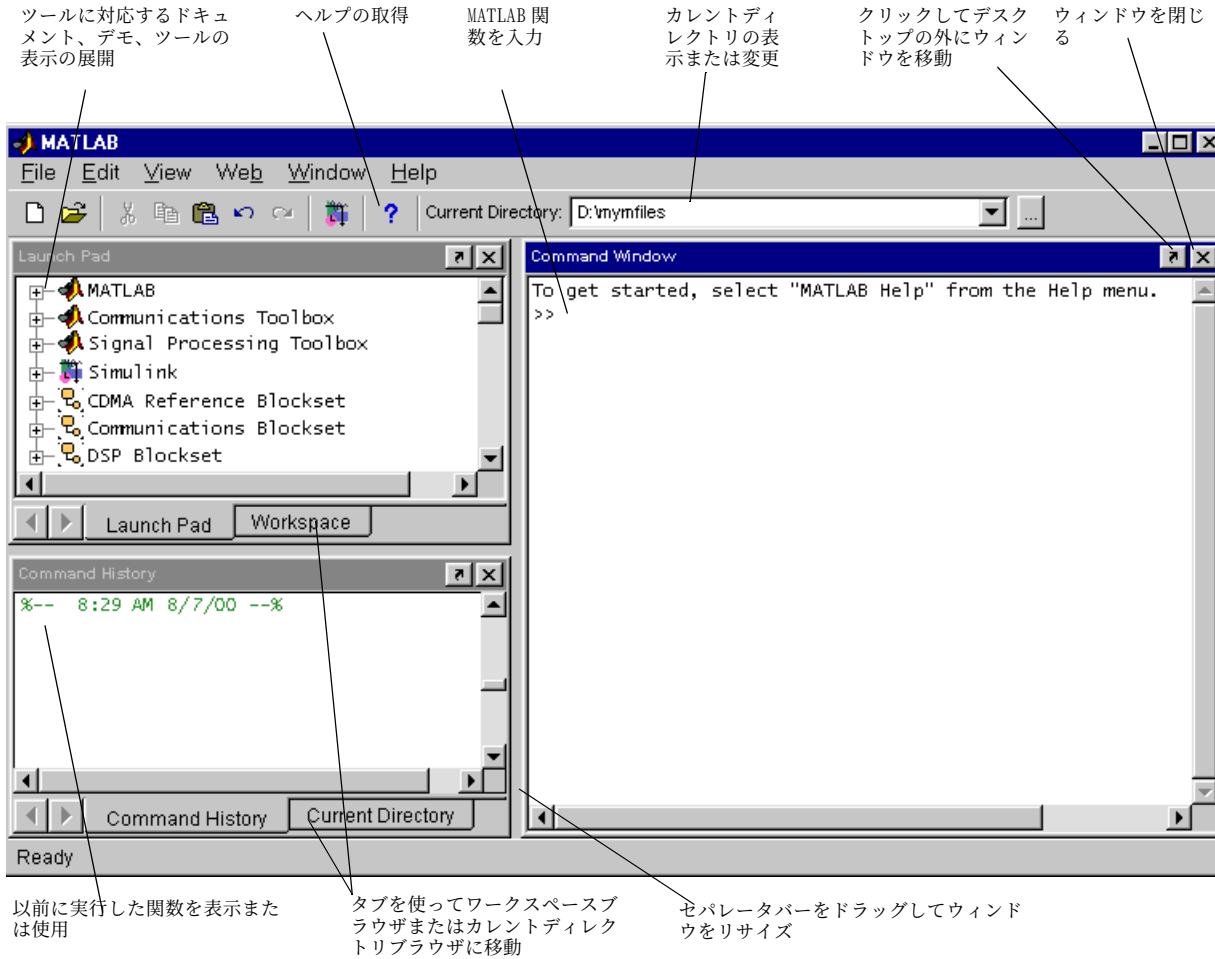
デスクトップ共通の機能 2-17
デスクトップツールバー 2-17
コンテキストメニュー 2-18
キーボードショートカットとアクセラレータ 2-18
複数項目の選択 2-19
クリップボードの利用 2-20
Web 上で The MathWorks にアクセス 2-20

ユーザ設定の変更 2-22
MATLAB の一般設定 2-24

MATLAB を起動すると、MATLAB デスクトップが表示されます。MATLAB デスクトップは、ファイル、ツール、MATLAB に関連するアプリケーションを管理するユーザインタフェースです。MATLAB に対して、ユーザのダッシュボードのような役割を果たすものです。デスクトップについて、つぎの事柄を知っておいてください。

- 2-4 ページの " デスクトップツール " – デスクトップによって管理されるすべてのツール
- 2-7 ページの " デスクトップの設定 " – デスクトップでのツールのアレンジ
- 2-17ページの"デスクトップ共通の機能" – ツール上で使用できる機能 (例:コンテキストメニュー)

MATLAB を最初に起動させると、デスクトップが下図のように表示されますが、Launch Pad は、入力により異なって表示されます。


ツールに対応するドキュメント、デモ、ツールの表示の展開
ヘルプの取得
MATLAB 関数を入力
カレントディレクトリの表示または変更
クリックしてデスクトップの外にウィンドウを移動
ウィンドウを閉じる
MATLAB
File Edit View Web Window Help
Current Directory: D:\mymfiles
Launch Pad
MATLAB
Communications Toolbox
Signal Processing Toolbox
Simulink
CDMA Reference Blockset
Communications Blockset
DSP Blockset
Launch Pad
Workspace
Command History
%-- 8:29 AM 8/7/00 --%
Command History
Current Directory
Command Window
To get started, select "MATLAB Help" from the Help menu.
>>
Ready
以前に実行した関数を表示または使用
タブを使ってワークスペースブラウザまたはカレントディレクトリブラウザに移動
セパレータバーをドラッグしてウィンドウをリサイズ

デスクトップツール

つぎのツールは、MATLAB デスクトップで管理できるツールです。しかし、これらのツールすべてが、MATLAB を最初に起動させた時に、デフォルトで表示されるわけではありません。コマンドラインインタフェースを使用したい場合、MATLAB デスクトップツールの有する機能のほとんどを関数を用いて実行することができます。このような等価な機能をもった関数の利用法ついての情報は、マニュアルのそれぞれのツールに関する節を参照してください。

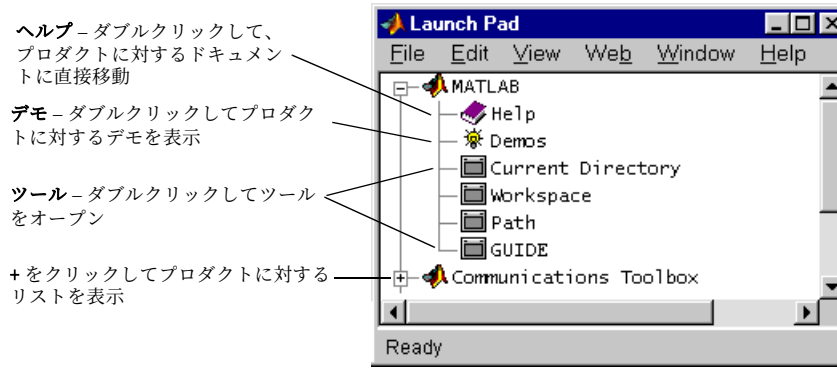
- コマンドウィンドウ – MATLAB 関数を実行します。
- コマンドの履歴 – コマンドウィンドウで入力した関数のログを表示したり、関数のコピーや、それらを実行します。
- Launch Pad – ユーザの所有する MathWorks 製品すべてについて、ツールのラウンチ、マニュアルへのアクセスを容易にします。
- カレントディレクトリブラウザ – MATLAB ファイル、及び、関連付けられたファイルを表示し、オープン、内容表示といった操作を、それらのファイルに対して実行します。
- ヘルプブラウザ – すべての MATLAB 関連製品に関するマニュアルの検索、表示を行います。
- ワークスペースブラウザ – ワークスペースの内容表示、変更を行います。
- 配列エディタ – 表フォーマットで配列の内容を表示したり、配列の値をエディットすることができます。
- エディタ / デバッガ – M- ファイルの作成、エディット、デバッグを行います。

フィギュアウィンドウのような他の MATLAB ツール、及び、MATLAB ウィンドウは、デスクトップから管理することができません。

Launch Pad

MATLAB の Launch Pad を使って、ユーザが所有する、すべての MathWorks 製品のツール、デモ、マニュアルに容易にアクセスすることができます。Launch Pad を開くには、MATLAB デスクトップの**ビュー (View)** メニューから**ラウンチパッド (Launch Pad)** を選択してください。ユーザのシステムにインストールされている MATLAB 製品の全リストが表示されます。

Launch Pad の一覧の例 – システムにインストールされたすべてのプロダクトの一覧が保湯時されます。



ある製品に関するリストを表示するには、製品の左側にある + 印をクリックしてください。リストアップを中止するには、製品の左側にある - 印をクリックしてください。

リストアップされた中の 1 つをオープンするには、選択してダブルクリックするか、選択して右クリックし、表示されたコンテクストメニューから Open を選択してください。オープンしたときにどのようなアクションが実行されるかは、選択したアイテムによって異なり、つぎの表のとおりとなります。

アイコン	オープンしたときの挙動の説明
	ヘルプブラウザ内にオープンした製品に対するドキュメンテーションのロードマップ
	選択された製品に関するデモと共に、デモランチャをオープン
	選択したツールをオープン

info.xml ファイルを作成することによってLaunch Padにユーザ独自の項目を追加することができます。Launch Pad の既存のエントリを 1 つ選択し、右クリックし、コンテキストメニューから **Edit Source** を選択します。そのプロダクトに対するツールが選択されたラインをもつ info.xml ファイルが現われます。アプリケーションに対する同様に info.xml ファイルを作成し、サーチパス上のフォルダに置きます。Launch Pad を右クリックして、コンテキストメニューから **Refresh** を選択すると、項目を含むように Launch Pad が更新されます。

デスクトップの設定

ユーザは、ニーズに合うようにデスクトップの設定を変更することができます。使わないツールをクローズし、使用するツールをオープンし、サイズを変更し、ツールの位置を変更することができます。MATLAB デスクトップは、つぎの操作によって設定を変更できます。

- 2-7 ページの " デスクトップツールのオープンとクローズ "
- 2-9 ページの " ウィンドウのサイズの変更 "
- 2-10 ページの " ウィンドウの移動 "
- 2-16 ページの " 既定のレイアウトで、MATLAB デスクトップ上にツールを配置する "

セッションの終了時に、MATLAB は、デスクトップ上の配置設定を記録します。つぎに、MATLAB を起動した場合、デスクトップはセッション終了時と同様の配置で表示されます。

デスクトップツールのオープンとクローズ

MATLAB デスクトップをユーザのニーズに合うように設定する過程で、つぎの機能を利用することができます。

- 2-7 ページの " デスクトップツールのオープン " – 使用するツールのみオープンします。
- 2-8ページの"デスクトップツールのドキュメントにジャンプ" – M- ファイル、フィギュア、その他のドキュメントに直接ジャンプします。
- 2-9 ページの " デスクトップツールのクローズ " – 使用しないデスクトップツールをクローズします。

デスクトップツールのオープン

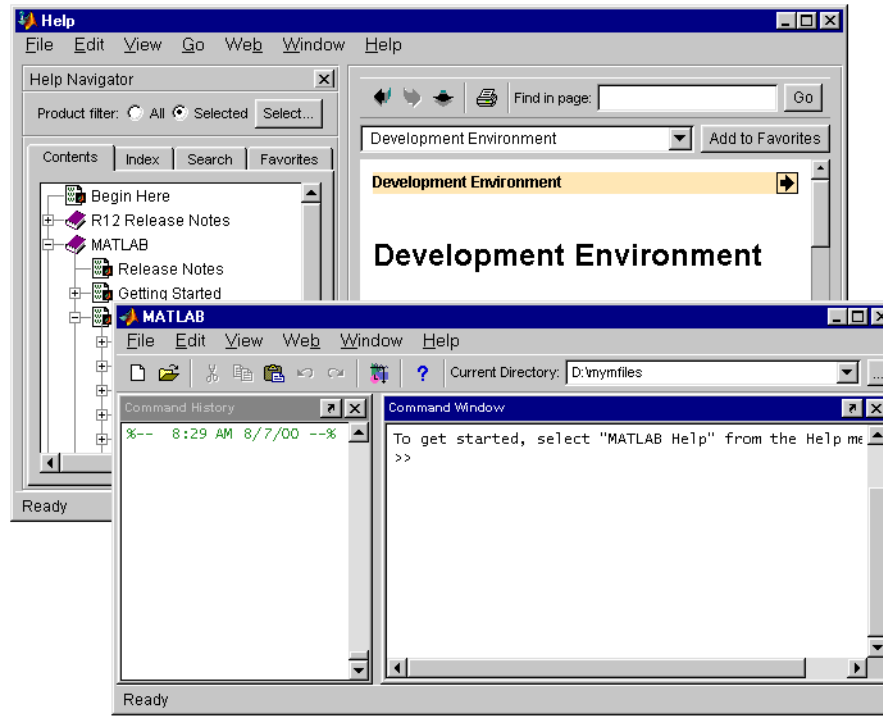
デスクトップからツールをオープンするには、**ビュー (View)** メニューから、オープンしたいツールを選択するか、または、MATLAB の Launch Pad に表示されるツールリストからオープンしたいツールをダブルクリックで選択します。ツールをオープンすると、以前、オープンしていたときと同一の位置にツールが開かれます。

ビュー (View)、または、Launch Pad からオープンできない、デスクトップで管理できないツールもあります。

- 配列エディタ – 配列エディタは、ワークスペースブラウザ上で変数をダブルクリックして、オープンします。
- エディタ / デバッガ – 新しい M- ファイルを作成するか、既存の M- ファイルをオープンすると、エディタやデバッガがオープンします。詳細については、7-3 ページの " エディタ / デバッガの起動 " を参照してください。

ツールをオープンする、もう 1 つの方法は、関数を使用する方法です。たとえば、関数 filebrowser は、カレントディレクトリブラウザをオープンします。これらの関数は、各ツールと共に説明されています。

下図の例は、MATLAB デスクトップが、コマンドウィンドウ、コマンド履歴、ヘルプブラウザをオープンした状態では、どのように表示されるかを示したものです。

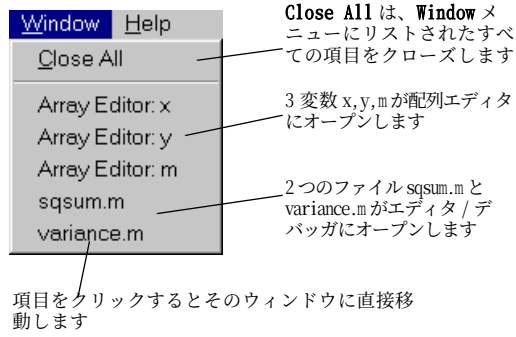


デスクトップツールのドキュメントにジャンプ

Window メニューは、オープンしているすべてのエディタ / デバッガドキュメント、配列エディタの変数、フィギュアウィンドウを表示します。Window メニューの中

にあるエントリを選択すると、選択したウィンドウ、または、タブの付いたドキュメントに直接ジャンプできます。**Window** メニューにリストアップされたすべてのアイテムをクローズするには、**Close All** を選択してください。

たとえば、下図の **Window** メニューは、配列エディタ上でオープンしている 3 つのドキュメントとエディタ / デバッガ上でオープンしている 2 つのドキュメントを表示しています。たとえば、ここでメニューから、variance.m を選択すると、ファイル variance.m を開いた状態のエディタ / デバッガウィンドウがアクティブなウィンドウとして表示されます。



デスクトップツールのクローズ

デスクトップツールをクローズするには、つぎの操作を行ってください。

- **ビュー(View)** メニューからクローズしたいアイテムを選択します (このとき、選択したアイテムはチェックが外れた状態になります)。
- ウィンドウのタイトルバーのウィンドウ上で、クローズボックス をクリックします。
- **ファイル(File)** メニューから **クローズ(Close)** を選択してカレントウィンドウをクローズしてください。

この操作で、ウィンドウはクローズします。

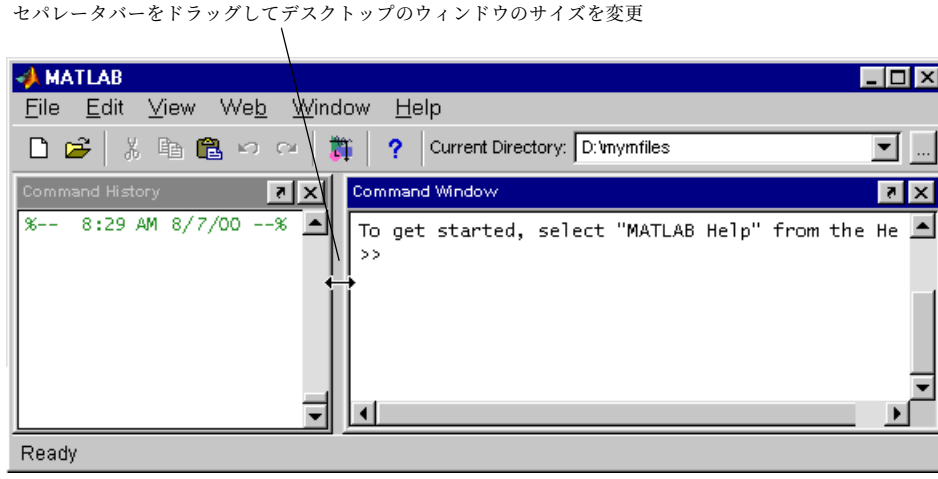
ウィンドウのサイズの変更

MATLAB デスクトップ上で、ウィンドウのサイズを変更するには、2 つのウィンドウを分けているセパレータバーを使用します。

1 カーソルをセパレータバーの上に移動します。

カーソルは、プラットフォームによって異なる形状になります。Windows プラットフォーム上では、カーソルは双頭の矢印 ↔の形で表示され、UNIX プラットフォームでは、バーを伴う矢印として表示されます。

2 セパレータバーをドラッグすると、ウィンドウのサイズが変更されます。



デスクトップの外にある MATLAB ツールのウィンドウ、または、MATLAB デスクトップそれ自体のサイズを変更するには、ウィンドウのいずれかのエッジ、または、コーナをドラッグしてください。

ウィンドウの移動

MATLAB デスクトップウィンドウを移動する簡単な 3 つの方法があります。

- 2-10 ページの "MATLAB デスクトップの中でウィンドウを移動する "
- 2-12ページの "ウィンドウをMATLABデスクトップの外へ移動する " と2-13ページの " ウィンドウを MATLAB デスクトップの中に移動する "
- 2-13 ページの " ウィンドウをまとめる (ウィンドウにタブを付ける)"

MATLAB デスクトップの中でウィンドウを移動する

MATLAB デスクトップ内の他の場所へウィンドウを移動するには、つぎの操作を行います。

1 ウィンドウのタイトルバーを、移動したい位置にドラッグします。

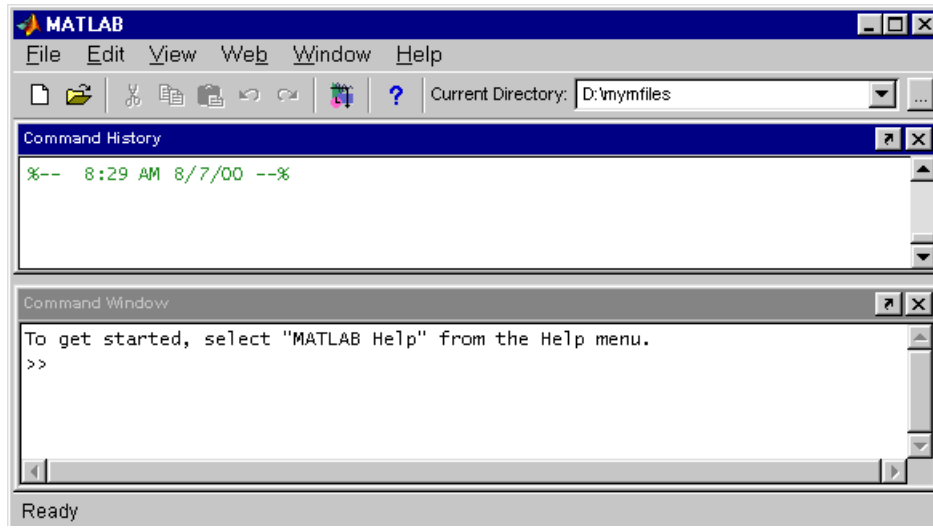
ウィンドウをドラッグすると、ウィンドウの外枠が表示されます。ウィンドウの外枠がウィンドウを表示できる位置に近付くと、ウィンドウの外枠は、その位置へと即座に移動します。外枠を移動している時に、ステータスバーは、ウィンドウの動きに関する情報を表示します。

下図の例では、当初コマンドウィンドウの左側に位置したコマンド履歴 (Command History) ウィンドウをコマンドウィンドウの上にドラッグする操作を示しています。コマンド履歴 (Command History) ウィンドウの外枠は、コマンド履歴 (Command History) ウィンドウがツールバーの下部に触れると同時に表示されます。

2 ウィンドウを新しい位置に固定するために、マウスを解除します。

デスクトップに表示されているもう 1 つのウィンドウも、ウィンドウの配置の変化に合わせて、そのサイズが変化します。下図の例では、コマンド履歴

(Command History) ウィンドウをコマンドウィンドウの上に移動した場合、デスクトップがどのように表示されるかが図示されています。

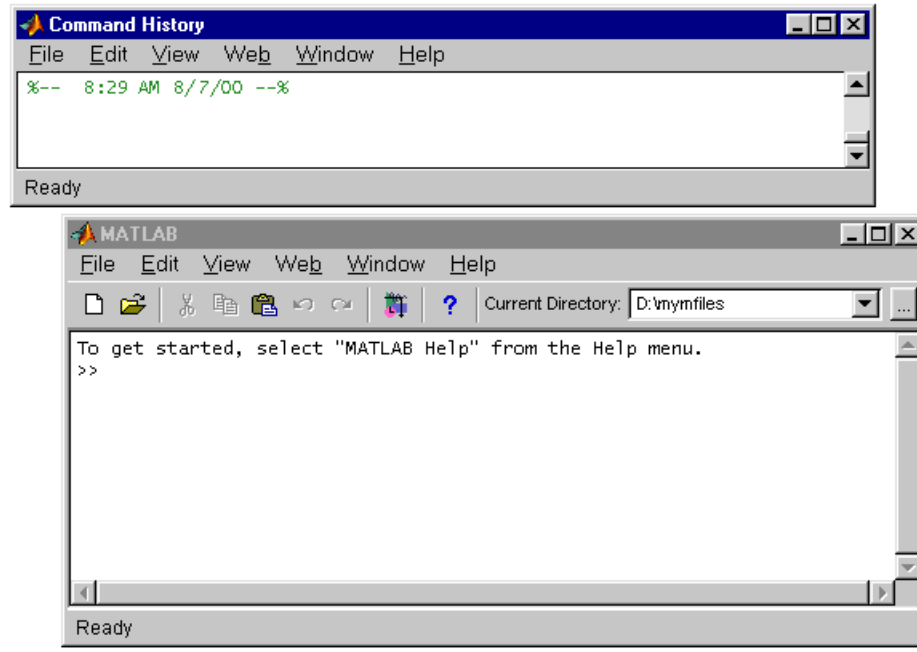


ウィンドウを MATLAB デスクトップの外へ移動する

MATLAB デスクトップの外へウィンドウを移動するには、つぎの操作のうちの 1 つを実行してください。

- デスクトップの外に移動したいウィンドウのタイトルバーに表記されている矢印 をクリックしてください。
- **ビュー (View)** メニューから移動したいツールに対して、**Undock** を選択します。この操作を行う場合には、移動するウィンドウがカレントのアクティブなウィンドウとなっていなければなりません。
- デスクトップの外へウィンドウのタイトルバーをドラッグします。ドラッグすると、ウィンドウの外枠が表れます。カーソルが MATLAB デスクトップの外に表示されると同時に、マウスを解除してください。

この操作により、MATLAB デスクトップの外部にウィンドウが表示されます。下図の例では、コマンド履歴 (Command History) ウィンドウを、デスクトップの外へ移動しています。



ウィンドウを MATLAB デスクトップの中に移動する

ウィンドウを MATLAB デスクトップの外から中へ移動するには、ウィンドウの **ビュー (View)** メニューから、そのツールに対して、**Dock** を選択します。

ウィンドウをまとめる (ウィンドウにタブを付ける)

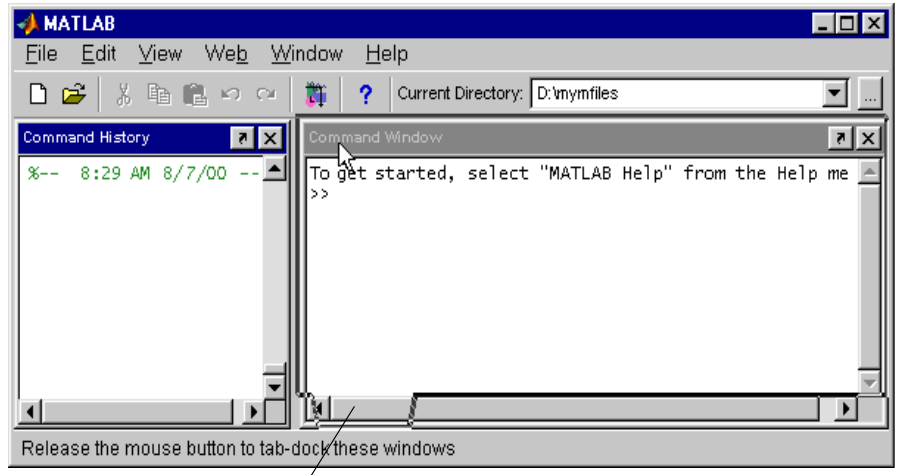
MATLAB デスクトップの中の同一スペースに複数のウィンドウをまとめて表示し、タブをクリックすることにより個々のウィンドウにアクセスすることも可能です。タブ付きウィンドウの取扱いに関する主要な機能は、つぎのとおりです。

- 2-14 ページの " ウィンドウのグルーピング "
- 2-15 ページの " タブの付いたウィンドウを表示します "
- 2-15 ページの " タブの付いたウィンドウを移動します "
- 2-15 ページの " タブの付いたウィンドウをクローズします "

ウィンドウのグルーピング．ウィンドウをまとめます ("タブを付ける" とも云います)。

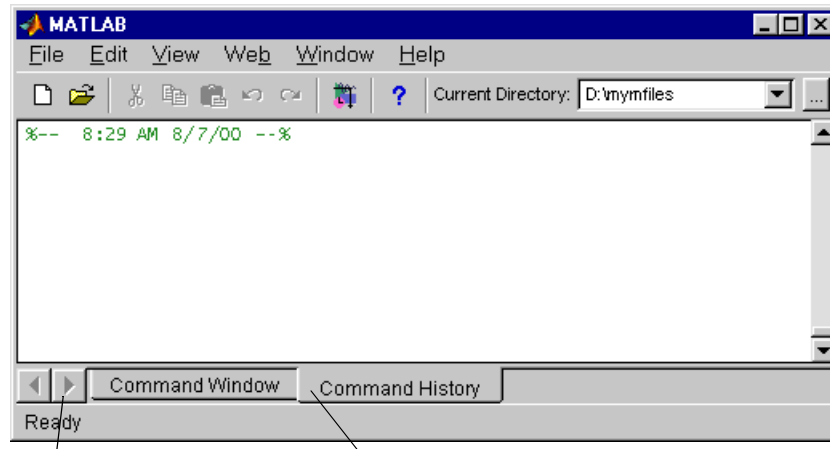
デスクトップ上に表示された特定ウィンドウのタイトルバーを、デスクトップ上に表示されたもう一方のウィンドウのタイトルバー上にドラッグします。

ユーザがドラッグしたウィンドウの外枠がターゲットウィンドウに重ね合わさると、外枠の下部にタブが表示されます。以下の例では、当初コマンドウィンドウの左側に位置していたコマンド履歴 (Command History) ウィンドウのタイトルバーを、コマンドウィンドウのタイトルバー上にドラッグしています。



3 マウスを解除します。

この状態でマウスを離すと、2 つのウィンドウが同一スペースに収まり、スペースの下部にタブが表示されます。下の例では、コマンド履歴 (Command History) ウィンドウとコマンドウィンドウ (Command Window) の双方にタブが付けられ、カレントウィンドウとして、コマンド履歴 (Command History) ウィンドウが選択されている状態が図示されています。



タブの付いたウィンドウを表示します．タブの付いたウィンドウを表示するには、ウィンドウのタブをクリックします。この操作により、クリックしたウィンドウが前面に表示され、カレントのアクティブウィンドウとして選択されます。カレントで表示可能な数以上のタブが存在するケースでは、タブの左側に表示された矢印をクリックすることにより、隠れているタブを表示させることができます。

タブの付いたウィンドウを移動します．タブの付いたウィンドウを他の位置に移動するには、移動したいウィンドウのタイトルバー、または、タブを新しい場所へとドラッグします。ユーザは、ウィンドウを MATLAB デスクトップの内部、または、外部へ動かすことができます。

タブの付いたウィンドウをクローズします．クローズボックス をクリックすると、タブが付けられ、まとめられている複数のウィンドウの内、1 つのウィンドウについて、クローズ操作を行うことができます。ユーザは、タブの付いたウィンドウを一度に全てクローズすることはできません。個々のウィンドウを 1 つずつクローズしてください。

既定のレイアウトで、MATLAB デスクトップ上にツールを配置する

MATLAB は、**View -> Desktop Layout** メニューから選択できる 6 つの前もって設定している MATLAB デスクトップ設定を用意しています。

- **デフォルト (Default)** – コマンドウィンドウ (Command Window)、コマンド履歴 (Command History) ウィンドウ、カレントディレクトリブラウザの 3 つのウィンドウが、タブでまとめてて配置され、Launch Pad とワークスペースブラウザの 2 つのウィンドウが、タブでまとめて配置されます。
- **コマンドウィンドウのみ (Command Window Only)** – コマンドウィンドウのみが配置されます。このレイアウトを適用すると、MATLAB デスクトップは、より前のバージョンのインタフェースに近い形で表示されます。
- **Simple** – コマンド履歴 (Command History) ウィンドウ、及び、コマンドウィンドウ (Command Window) が、並んで配置されます。
- **Short History** – コマンドウィンドウ (Command Window) と小さく表示されたコマンド履歴 (Command History) ウィンドウの組み合わせの上に、カレントディレクトリブラウザとワークスペースブラウザが、タブでまとめて配置されます。
- **Tall History** – 左側にコマンド履歴 (Command History) ウィンドウ、右側にコマンドウィンドウ (Command Window) が配置され、コマンドウィンドウの上部にカレントディレクトリブラウザとワークスペースブラウザがタブで、まとめて配置されます。
- **Five Panel** – 左側にコマンド履歴 (Command History) ウィンドウと、その上に Launch Pad が配置された組み合わせ、中央にカレントディレクトリブラウザとその上にワークスペースブラウザが配置された組み合わせ、右側にコマンドウィンドウが、それぞれ配置されます。

既定のレイアウトを選択した後でも、ウィンドウの移動、サイズ変更、オープン、クローズを行うことは可能です。

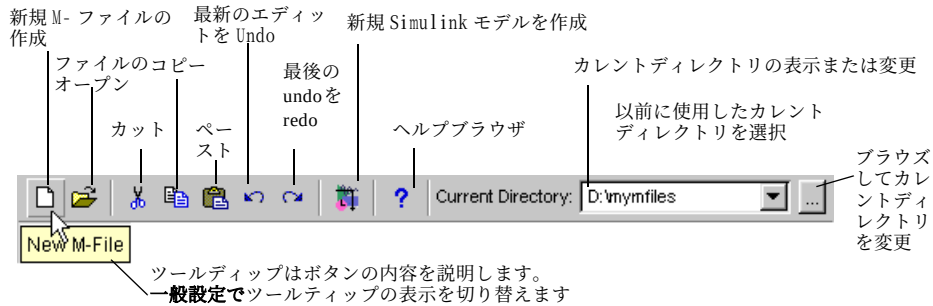
デスクトップ共通の機能

デスクトップツール共通の機能として、つぎの機能が利用できます。

- 2-17 ページの " デスクトップツールバー "
- 2-18 ページの " コンテキストメニュー "
- 2-18 ページの " キーボードショートカットとキーボードアクセラレータ "
- 2-19 ページの " 複数項目の選択 "
- 2-20 ページの " クリップボードの使用 "
- 2-20 ページの "Web 上で MathWorks にアクセス "

デスクトップツールバー

MATLAB デスクトップ上に表示されるツールバーを使って、よく利用される操作に簡単にアクセスすることができます。カーソルをある特定のボタンの上に置くと、そのボタンの機能を説明するバルーンヘルプが表示されます。ツールの中には、ツールのウィンドウ内にそのツール独自のツールバーを表示するものもあることに注意してください。

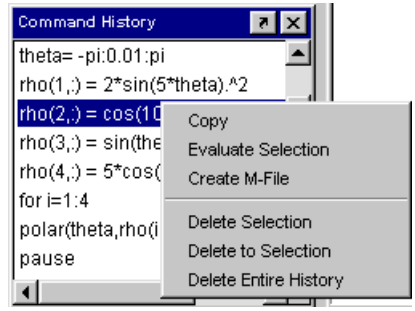


カレントディレクトリフィールド

ツールバー内に表示される**カレントデレクトリ (Current Directory)** フィールドは、MATLAB がカレントで作業しているディレクトリを示しています。このフィールドを使ってカレントディレクトリを変更し、カレントディレクトリブラウザを使ってその他のファイル操作を行うことができます。– 詳細は、5-20 ページの " ファイルオペレーション " を参照してください。

コンテキストメニュー

MATLAB デスクトップツールの機能の多くは、コンテキスト (Context) メニューからも利用できます。コンテキスト (Context) メニューにアクセスするには、選択対象の上で右クリックを行います。この操作により、選択対象に関するコンテキスト (Context) メニューが表示され、利用可能な選択肢が提示されます。下図の例では、コマンド履歴 (Command History) ウィンドウ上の選択対象に対して、コンテキスト (Context) メニューが表示されています。



選択肢を右クリックしてコンテキスト (ポップアップ) メニューにアクセスします

キーボードショートカットとキーボードアクセラレータ

ユーザは、使用しているプラットフォーム固有のキーボードショートカットやアクセレラタを使って、たとえば、Windows では、**カット (Cut)** を行うには、**Ctrl+x** を、**ファイル (File)** をオープンするには、**Alt+f** を使うように、多くのメニューアイテムにアクセスできます。多くのショートカットやアクセレラタは、メニューアイテムと共にリストされます。たとえば、Windows プラットフォームで、**編集 (Edit)** メニューは、**Cut Ctrl+x** を示し、**ファイル (File)** メニューは、**File** の **F** の下に下線を示し、**Alt+f** で、それをオープンすることを示しています。ユーザのプラットフォームで、多くの標準的なショートカットが機能しますが、メニューアイテムと共にリストされないものもあります。
以下に示すのは、メニューアイテムリストされていないショートカットです。

キー	結果
Enter	ダブルクリックと等価で、選択に対するデフォルトの操作を行います。たとえば、コマンド履歴ウィンドウの行が選択されているときに **Enter** を押すとコマンドウィンドウでそのラインを実行します。
Escape	カレントのアクションをキャンセルします。

キー	結果
Ctrl+Tab または Ctrl+F6	デスクトップのつぎのタブに移動します。このときタブは、ツール、エディタ / デバッガ内のファイルに対応します。デスクトップ外でエディタ/デバッガでタブモードで利用するときには、オープンしているつぎのファイルに移動します。
Ctrl+Shift+Tab	デスクトップの前にタブに移動します。このときタブは、ツール、エディタ / デバッガ内のファイルに対応します。デスクトップ外でエディタ / デバッガでタブモードで利用するときには、オープンしている前のファイルに移動します。
Ctrl+Page Up	ツールのグループまたはタブ付けられているファイル内のつぎのタブに移動します。
Ctrl+Page Down	ウィンドウ内の前のタブに移動します。
Alt+F4	デスクトップまたはデスクトップ外のウィンドウをクローズします。
Alt+Space	システムメニューを表示します。

複数項目の選択

多くのデスクトップツールにおいて、複数の項目を選択し、選択したすべての項目について行うアクションを選択することができます。プラットフォームに対応する標準の用法を使って複数項目を選択してください。

たとえば、プラットフォームが Windows の場合は、複数項目を選択するには、以下を行ってください。

1 選択したい最初の項目をクリックします。

2 Ctrl キーを押したまま、つぎに選択したいアイテムをクリックします。選択したいアイテムすべての選択を終えるまで、この操作を繰り返してください。

3 Ctrl キーの代わりに、Shift キーを押しながらアイテムのクリックを行った場合、そのアイテムから最後にクリックしたアイテムまでのすべてのアイテムが選択対象となります。

選択を終了すると、ユーザは選択したアイテムに対して、削除などの何らかの挙動を実行することが可能となります。

クリップボードの使用

ユーザは、デスクトップツールから選択対象をクリップボードへカット、コピーし、それをクリップボードから他のデスクトップツールにペーストすることができます。これらの操作を行うには、**編集 (Edit)** メニュー、コンテキストメニュー、または、標準で用意されたキーボードショートカットを使用してください。たとえば、ユーザは、コマンド履歴 (Command History) ウィンドウから選択したコマンドをコピーして、デスクトップ内にペーストすることができます。

編集 (Edit) メニューから**特殊な貼り付け (Paste Special)** アイテムを選択すると、クリップボードに保持された選択対象をインポートウィザード上にオープンします。この機能を使用すれば、Excel などの外部アプリケーションからコピーしたデータを、MATLAB 上で利用することが可能となります。この点の詳細については、第 6 章 , " データのインポートとエクスポート " を参照してください。

最近行ったカット、コピー、ペーストなどの操作をやり直すには、**編集 (Edit)** メニューから **Undo** を選択してください。また、**Undo** 操作を元に戻すには、**Redo** を選択してください。

選択対象をドラッグすることによって、コピーすることもできます。たとえば、コマンド履歴 (Command History) ウィンドウ上で選択した対象を、コマンドウィンドウにドラッグし、そこにペーストすることも可能です。コマンドウィンドウ上でペーストしたラインをエディットし、必要ならば、**Enter** キーを押せば、コマンドウィンドウから対象のラインを実行することができます。

Web 上で MathWorks にアクセス

MATLAB デスクトップから MathWorks の Web ページにアクセスできます。**Web** メニューの中のつぎの選択肢から 1 つ選択してください – Web ページは、ユーザの使用しているデフォルトの Web ブラウザ内にオープンします。

- **The MathWorks Web サイト** – MathWorks Web サイトのホームページ (http://www.mathworks.com) にリンクします。
- **Technical Support Knowledge Base** – MathWorks サポートページ (http://www.mathworks.com/support) にリンクします。このページでは、直面した問題に対するソリューションを検索することができます。
- **Products** – MathWorks 製品のページ (http://www.mathworks.com/products/) にリンクします。ここでは MATLAB 関連商品に関する情報が入手できます。
- **Membership** – アクセスメンバー向け、MATLAB アクセスページ (http://www.mathworks.com/mla/index.shtml) にリンクします。メンバーでない方も、

オンラインでメンバーとなることができ、メンバーになると MATLAB 製品のアップデートに関する情報が得られます。

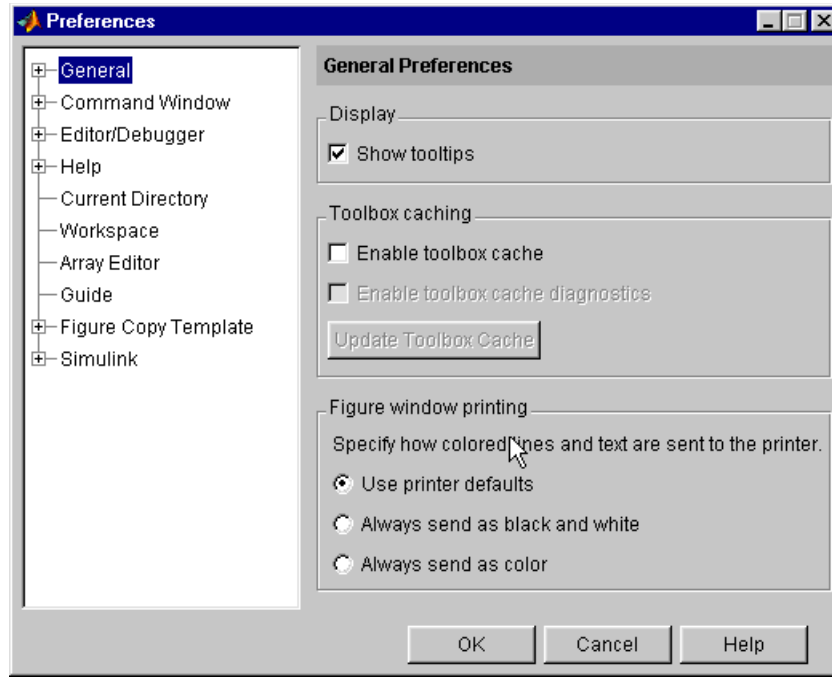
ユーザ設定の変更

ユーザ設定を変更すれば、コマンドウィンドウで表示されるフォント等、MATLABが、ある側面において示す挙動をデフォルトのものから変更することができます。ユーザ設定で定めた値は、MATLABセッションを通じて固定されたままです。ただし、ツールの中には、ユーザ設定を変更することなく、その対象となる領域を設定とは独立に、そのツール内部で制御することを可能にするものもあります。この場合、カレントのセッションでの適用のみに変更を加えたい場合の方法を使ってください。

ユーザ設定を変更するには、つぎの操作を行います。

1 **ファイル (File)** メニューから**ユーザ設定 (Preferences)** を選択します。

この操作により、**ユーザ設定 (Preferences)** ダイアログボックスがオープンします。ここで、開かれるページの内容は、カレントでアクティブとなっているウィンドウを反映したものとなります。



2 左枠から、設定したいユーザ設定のタイプを選択します。上図の例では、General ユーザ設定 (一般設定) が選択されています。

アイテムの左側に、+ 記号が表示された場合、+ 記号をクリックすると、より多くのアイテムが表示されます。ここから、設定値を変更したいアイテムを選択することも可能です。

右枠の内容は、ユーザが選択した設定のタイプを反映したものとなります。

3 右枠で、プリファレンスの設定値を指定し、OK をクリックする。

この操作を終了すると、ユーザ設定の設定値は直ちに有効となります。

ユーザ設定のタイプと説明が記載されている場所	設定値を変更できるアイテム
2-24 ページの "MATLAB の一般設定 "	デスクトップの表示、キャッシュ、印刷、フォント、カラー、ソースコントロールシステム
3-13 ページの " コマンドウィンドウの設定 "	数値の書式、表示、エコー、フォント、カラー
7-34 ページの " エディタ / デバッガの設定 "	スタートアップオプション、フォント、カラー、表示、キーボードショートカット、インデントと印刷
4-23 ページの " マニュアルの印刷 "	ドキュメントの位置、プロダクト、PDF リーダの位置、同期、フォント
5-30 ページの " カレントディレクトリブラウザの設定 "	履歴の入力数、ファイル表示オプション
5-9 ページの " ワークスペースブラウザの設定 "	フォントと、変数の削除の確認
5-12 ページの " 配列エディタの設定 "	フォントと数値表現書式
GUIDE	表示オプション

ユーザ設定のタイプと説明が記載されている場所	設定値を変更できるアイテム (Continued)
Figure Copy Template	アプリケーション、テキスト、ライン、uicontrol、軸、フォーマット、バックグランドカラー、サイズ
Simulink	表示、フォント、シミュレーション

MATLAB の一般設定

つぎの項目で設定した値は、MATLAB デスクトップ上の関連するすべてのツールに適用されます。

表示 (Display)

ツールテップの表示 (Show tooltips) ボックスをチェックすると、ツールバーボタンの上にカーソルを置いたときにバルーンヘルプを表示します。

ツールボックスパスキャッシュ (Toolbox caching)

1-9 ページの " ツールボックスパスキャッシュと共に起動時間を短縮 " を参照してください。

フィギュアウインドウ印刷 (Figure window printing)

印刷する場合に、ラインや文字を何色で印刷するかを指定します。詳細については、ドキュメントの印刷を参照してください。

フォント & カラー

デスクトップフォント . デスクトップフォント設定では、MATLAB デスクトップ上のツールで使用するフォントの特性を指定します。指定できる特性は、つぎのとおりです。

- タイプ、例、Sans Serif
- スタイル、例、Bold
- ポイント単位のサイズ、例、12 ポイント

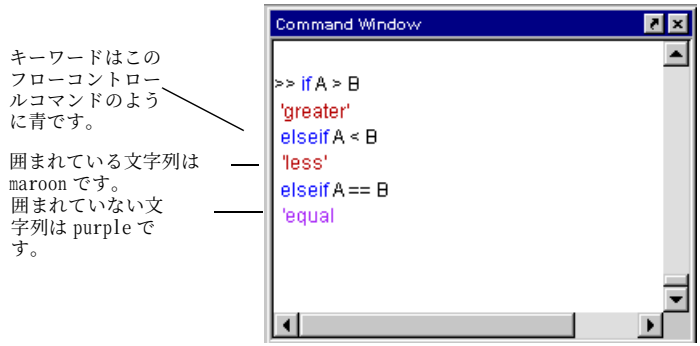
選択すると、**Sample** エリアでフォントがどのように表示されるかを確認できます。

ユーザは、コマンドウィンドウ、エディタ / デバッガ、ヘルプブラウザ、ワークスペースブラウザ、配列エディタの各々に対して、これらのツールに用意された設定アイテムを通じて、異なるフォントを指定することができます。

シンタックスの強調カラー (Syntax highlighting colors). シンタックスを強調するときに使用するカラーを選択します。詳細については、3-5 ページの " シンタックスの強調 " を参照してください。

- **Keywords** – for、if 等のフロー制御関数、及び、その他の関数のカラー
- **Comments** – % で始まるすべてのラインのカラー
- **Strings** – シングルコートで括られた部分のカラー
- **Unterminated strings** – 対応するシングルコートが存在しないシングルコート、及び、そのコーテションの前後の部分に着色されるカラー
- **System commands** – (シェルエスケープ)! のようなコマンドのカラー
- **Errors** – エラーを含むテキストに塗られるカラー

Restore Default Colors をクリックすると、デフォルトの設定が適用されます。下図の例は、カラーの設定にデフォルト値を適用した例を示しています。



Source Control

MATLAB へのインタフェース用のソースコントロールシステムを指定します。詳細については、第 9 章 , " ソースコントロールシステムとのインタフェース " を参照してください。

3

MATLAB 関数の実行

コマンドウィンドウ 3-2
コマンドウィンドウを開く 3-2
関数の実行と変数の入力 3-2
入出力の制御 3-4
プログラムの実行 3-11
セッションログの記録 3-12
コマンドウィンドウの設定 3-13

コマンド履歴 3-16
コマンド履歴ウィンドウに関数を表示 3-16
コマンド履歴ウィンドウから関数を実行 3-17
コマンド履歴ウィンドウから関数をコピー 3-17

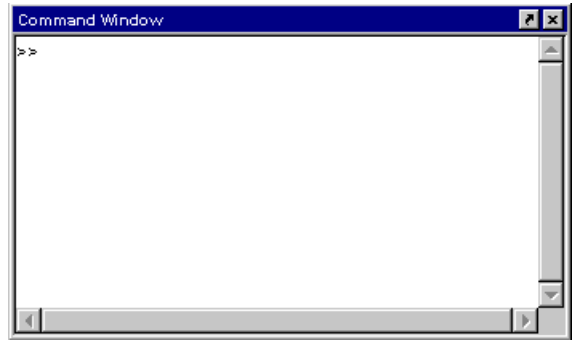
コマンドウィンドウ

MATLAB とコミュニケートするための主要な手段は、コマンドウィンドウです。コマンドウィンドウは、MATLAB を最初に起動したときにデスクトップに表示されます。コマンドウィンドウを使用して、MATLAB 関数 (コマンドとも呼ばれます) を実行し、MATLAB を操作できます。コマンドウィンドウの主要な機能は、つぎの通りです。

- 3-2 ページの " コマンドウィンドウを開く "
- 3-2 ページの " 関数の実行と変数の入力 "
- 出力表示の省略、コマンドラインのエディット等の3-4ページの"入出力の制御"
- M- ファイルと外部プログラムを含む 3-11 ページの " プログラムの実行 "
- 3-12 ページの " セッションログの記録 "
- 3-13 ページの " コマンドウィンドウの設定 "

コマンドウィンドウを開く

MATLAB デスクトップにコマンドウィンドウを表示するには、**ビュー(View)** メニューから**コマンドウインドウ (Command Window)** を選択します。詳細は、2-7 ページの " デスクトップツールのオープンとクローズ " を参照してください。



関数の実行と変数の入力

コマンドウィンドウのプロンプト (>>) は、MATLAB がユーザからのコマンドを受け入れる準備ができたことを示します。>> プロンプトが表示されている時には変数の入力または関数の実行を行えます。たとえば、3 行 3 列の行列 A を入力するには、つぎのようにタイプしてください。

```
A = [1 2 3; 4 5 6; 7 8 10]
```

ここで上記のラインをタイプした後に、**Enter** または **Return** キーのいずれかを押すと、MATLAB は、つぎのような表示を返してきます。

```
A =

     1     2     3
     4     5     6
     7     8    10
```

関数を実行するには、全ての引数を含めた形で関数をタイプし、**Return** または **Enter** キーを押します。これにより、MATLAB は計算結果を表示します。たとえば、つぎのようにタイプすると、

```
magic(2)
```

MATLAB は、つぎの計算結果を出力します。

```
ans =
     1     3
     4     2
```

実行前に複数のラインを入力したい時は、ライン毎に、**Shift+Enter** または **Shift+Return** を、それぞれ入力します。そして、最後のラインまで入力した後、**Enter** または **Return** キーを押すと、入力した全てのラインが実行されます。

コマンドウィンドウ上の K>> プロンプトは、MATLAB がデバッグモードにあることを示しています。詳細については、第 7 章 , "M- ファイルのエディットとデバッグ " を参照してください。

選択した関数を実行

コマンドウィンドウで選択した関数を実行するには、まず関数を選択し、右クリックでコンテキストメニューを表示、コンテキストメニューから**選択したものの実行 (Evaluate Selection)** を選択します。M- ファイルが実行されている場合の MATLAB がビジーの時は、選択した関数を実行することはできません。

選択した関数を開く

ある特定の関数に関する M- ファイルを開くには、つぎの操作を行います。まず、コマンドウィンドウで関数を選択し、右クリックでコンテキストメニューを表示、表示されたコンテキストメニューから**選択したものを開く (Open Selection)** を選択します。これにより、エディタ／デバッガ上で、その関数の M- ファイルが開かれます。

1 つのプロセスしか実行できません

ユーザは、1 回に 1 つのプロセスしか実行できません。MATLAB が 1 つの関数の実行でビジーになっているケースでは、ユーザが入力するいかなるコマンドも実行されずに、一時的に保存されることになります。この場合、1 つのコマンドの実行が終わると同時に、つぎのコマンドの実行が行われます。たとえば、MATLAB がコマンドウィンドウで関数を実行している時に、エディタ／デバッガでブレークポイントの設定を行う、というように 2 つのプロセスを同時に実行することはできません。

入出力の制御

コマンドウィンドウ上で入出力を制御 / 解釈する方法として、つぎの方法が用意されています。

- 3-4 ページの " 大文字 / 小文字の区別とスペースの識別 "
- 3-5 ページの "1 つのラインに複数の関数の入力 "
- 3-5 ページの " 長い関数の入力 "
- 3-5 ページの " シンタックスの強調 "
- 3-6 ページの " コマンドウィンドウで使用するフォントの指定 "
- 3-6 ページの " コマンドラインのエディット "
- 3-9 ページの " コマンドウィンドウの消去 "
- 3-9 ページの " 出力の抑制 "
- 3-9 ページの " コマンドウィンドウ上の出力のページング "
- 3-10 ページの " 数値出力のフォーマット及びスペーシングの制御 "
- 3-11 ページの " コマンドウィンドウの表示内容を印刷 "

大文字 / 小文字の区別とスペースの識別

MATLAB は、入力された文字の大文字 / 小文字の区別を行ないます。たとえば、ユーザは、Plot とタイプしても関数 plot を実行することはできません。関数 plot を実行するには、Plot ではなく、plot と入力しなければなりません。また、変数 a は、変数 A と同一ではありません。但し、help 関数を使用する時には、関数の区別を行うために、関数名は大文字 (例：PLOT) で表示されます。しかし、関数を実行するときには、大文字を使用することは出来ません。

-、:、( ) などの演算子の前後にブランクを付けるかどうかは、オプションで設定できます。ブランクを付ければ、文字列が読み易くなります。

1 つのラインに複数の関数の入力

1 つのラインに複数の関数を入力するには、関数と関数の間にセミコロン (;) を入力します。たとえば、対数表を作成するために、1 つのラインに 3 つの関数を入力するには、つぎのようにタイプします。

```
format short; x = (1:10)'; logs = [x log10(x)]
```

その後、**Enter** または **Return** キーを押せば、関数が左から右へと順番に実行されていきます。

長い関数の入力

ステートメントが 1 行で終わらない長さの場合、つぎのように操作してください。先ず、省略符号 (3 つのピリオド、...) を用いて、ステートメントが、つぎの行まで続いていることを示します。つぎに、**Enter** または **Return** を押して、つぎのラインに移り、ステートメントの入力を続けてください。たとえば、つぎのように入力します。

```
s = 1 – 1/2 + 1/3 – 1/4 + 1/5 – 1/6 + 1/7 ...
    – 1/8 + 1/9 – 1/10 + 1/11 – 1/12;
```

シングルコーテション (' ') で括られた文字列のような項目については、行を変える毎に引用符で括りなおして下さい。たとえば、つぎのような長い文字列は、このように入力してください。

```
headers = ['Author Last Name, Author First Name, ' ...
'Author Middle Initial']
```

上記のように入力すると、つぎの式と同一の結果が得られます。

```
headers =
Author Last Name, Author First Name, Author Middle Initial
```

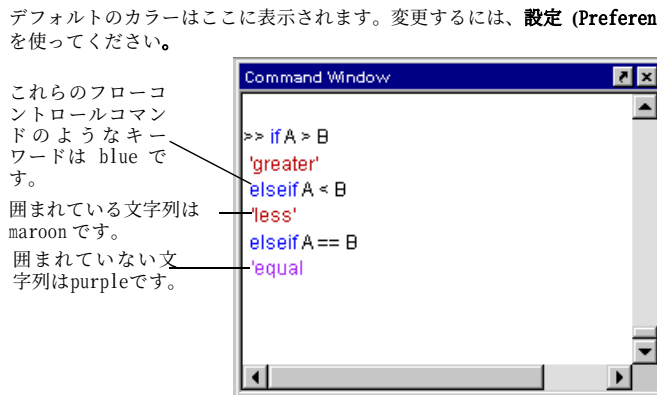
なお、一つの行で使える最大文字数は 4096 です。

シンタックスの強調

if/else ステートメントを照合するような場合、要素が探し易くなるように、入力を異なる色で表示します。

- 文字列をタイプすると、その文字列は紫色で表示されます。文字列を閉じると、文字列は栗色で表示されます。
- フロー制御関数 for のようなキーワード、または、省略符号などで、つぎに続けた部分については、青色で表示されます。フロー制御関数の始まりと終わりの間に入力したラインについては、インデントされます。

- トークンの始まり、または、終わりを示す丸括弧 ( )、角型括弧 [ ]、ブレース { } をダブルクリックすると、そのトークンで囲まれた文字列を選択することができます。
- トークンの終わりを示す符号をタイプすると、その符号と組となるトークンの最初を示す符号が一時的に強調表示されます。
- コメント符号 % をタイプすると、その行は緑色で表示されます。
- ! (シェルエスケープ) 等のシステムコマンドをタイプすると、その行は金色で表示されます。
- エラーは、赤で表示されます。



シンタックスを強調する際に用いる色を変更したい時は、3-14 ページの " コマンドウィンドウ上の表示に適用されるフォントと色の設定 " を参照してください。

コマンドウィンドウで使用するフォントの指定

ユーザは、コマンドウィンドウで使用するフォントのタイプ、スタイル、サイズを自由に指定できます。詳細については、3-14 ページの " コマンドウィンドウ上の表示に適用されるフォントと色の設定 " を参照してください。

コマンドラインのエディット

コマンドウィンドウ上でのエディットにかける時間を節約したい、そんなニーズに答えるために、つぎの機能が用意されています。

- クリップボード機能
- 最近タイプしたラインを再び呼び出す

- タブでステートメントを完成させる

クリップボード機能．コマンドウィンドウ上で作業している時に、**Edit** メニューの **Cut**, **Copy**, **Paste**, **Undo**, **Redo** 機能を使用することが出来ます。これらの機能の一部は、コマンドウィンドウのコンテクストメニューからも使用することが出来ます。

最近タイプしたラインを再び呼び出す．最近タイプした関数を再び呼び出すには矢印キーを、エディットするにはタブキーを、再び使用するにはコントロールキーをそれぞれ押してください。たとえば、誤ってつぎのように入力したと仮定します。

```
rho = (1+ sqt(5))/2
```

この時、sqrt のスペルが間違っているために、MATLAB は、つぎのような出力を返してきます。

```
Undefined function or variable 'sqt'.
```

このようなケースで、ライン全体を再びタイピングするのではなく、↑キーを押してください。これにより、前にタイプしたラインが再び表示されます。その後、キーを押しカーソルを動かし、間違って落としてしまった r を付け加えてください。なお、再びキーを使えば前にタイプしたラインを再び呼び出すことができます。

ユーザが入力した関数はバッファに記憶されています。最近使用した関数の最初の数文字を指定するだけでその関数を呼び出すスマートリコール機能を利用することもできます。たとえば、文字列 plo をタイプし、キーを押すと、最近使用した関数の中から、plo という文字列で始まる名前の関数 (たとえば、最近に使用した関数 plot) を自動的に選択、再び呼び出します。この機能では、大文字小文字の区別が有効として感知されます。

コマンドウィンドウで使用できる矢印キー、コントロールキーによる全ての操作を下のリストにまとめています。下記のキー操作のほとんどは、Emacs エディタと共通のものになっています。

キー	コントロールキー	操作
↑	Ctrl+p	前のラインを呼び出します。前に使用した関数のログについては、3-16 ページの " コマンド履歴 " と 3-12 ページの " セッションログの記録 " を参照してください。
↓	Ctrl+n	つぎのラインを呼び出します。

キー	コントロールキー	操作 (Continued)
←	Ctrl+b	1 キャラクタ分、後ろに移動
→	Ctrl+f	1 キャラクタ分、前に移動
Ctrl+ →	Ctrl+r	右に 1 ワード分移動
Ctrl+ ←	Ctrl+l	左に 1 ワード分移動
Home	Ctrl+a	ラインの最初に移動
End	Ctrl+e	ラインの最後に移動
Esc	Ctrl+u	ラインのクリア
Delete	Ctrl+d	カーソルの位置のキャラクタを削除
Backspace	Ctrl+h	カーソルの前のキャラクタを削除
	Ctrl+k	ラインの最後までを削除
Shift+home		ラインの先頭を強調
Shift+end		ラインの末尾を強調

タブでステートメントを完成． MATLAB には、関数、変数、ファイル名、Handle Graphics のプロパティの名称を入力する場合、最初の数文字を入力し、**タブ (Tab)** キーを押すだけで完全な名称を入力することができる機能が備わっています。たとえば、変数 costs_march を生成しようとする場合、つぎのようにタイプし、

```
costs
```

タブ (Tab) キーを押せば、省略された部分が MATLAB によって自動的に補われ、画面上に、つぎのように表示されます。

```
costs_march
```

ここで、**Return** または **Enter** キーを押せば上記のステートメントを実行することができます。この例では、MATLAB は、costs_march の内容を表示します。

ユーザが、タイプした数文字から始まる名称をもつ関数が 1 つ以上存在する場合では、**タブ (Tab)** キーを押すと、その数文字から始まる関数のリストが表示されます。たとえば、つぎのようにタイプし、

```
cos
```

タブ (Tab) キーを押します。すると、MATLAB は何も表示せず、cos という文字で始まる名前の関数が複数あることを示唆します。ここで、再び、**タブ (Tab)** キーを押すと、MATLAB は、つぎのように表示します。

```
cos          cosh         costfun
cos_tr       cosint       costs_march
```

このリストには、ユーザが作成した関数 costs_march のみならず、cos で始まる全ての関数が記載されています。

コマンドウィンドウの消去

コマンドウィンドウを消去するには、**編集 (Edit)** メニューから**コマンドウインドウを消去 (Clear Command Window)** を選択します。このコマンドは、コマンドウィンドウの表示のみを消去するもので、ワークスペースの情報については消去されません。コマンドウィンドウを消去した後でも、ユーザは↑キーによって前に利用した関数を呼び出すことができます。

等価な機能をもつ関数 . 関数 clc を使って、コマンドウインドウをクリアできます。関数 clc と同様に、関数 home は、コマンドウインドウのトップにプロンプトを移動します。

出力の抑制

セミコロン (;) をラインの終わりに付けて、**Enter** または **Return** キーを押すと、MATLAB は、ステートメントを実行しますが、出力結果を表示しません。これは、特に、大きな行列を作成するとき有効です。たとえば、

```
A = magic(100);
```

とタイプし、**Enter** または **Return** キーを押すと、A は作成されますが、その結果として、出力される行列の表示は行われません。

コマンドウィンドウ上の出力のページング

コマンドウィンドウ上の出力が長すぎると、出力全体がスクリーンに表示しきれず、目で追いきれないほどの速さで表示されてしまいます。このようなケースでは、関数 more を使って、コマンドウインドウで出力のページングを行なえます。デフォルトでは、more は、off です。more on とタイプすれば、MATLAB は出力を 1 回に 1 ページ (スクリーン一杯に入りきる長さ) のみ表示します。最初のスク

リーン表示の後、以下のキーを押します。

キー	アクション
Enter または Return	つぎのラインに進む
Space Bar	つぎのページに進む
q	出力表示を停止する

数値出力のフォーマット及びスペーシングの制御

デフォルトでは、コマンドウィンドウ上の数値出力は、5 桁にスケーリングされ、固定小数点表示で表示されます。数値出力のフォーマットを制御するには、text display プリファレンスを使用してください。text display フォーマットは、数値の表示にのみ適用され、MATLAB における計算結果、セーブされた情報には適用されません。

等価な機能をもつ関数． コマンドウィンドウに表示される数値の出力のフォーマットを制御するには、関数 format を利用することもできます。指定したフォーマットは、カレントのセッションにのみ適用されます。

フォーマットの例題． 以下の例は、異なる大きさをもつ 2 要素からなるベクトル x からの出力を様々なフォーマットで出力したものです。

```
x = [4/3 1.2345e-6]

format short e
   1.3333e+000   1.2345e-006

format short
   1.3333     0.0000

format +
++
```

利用可能なフォーマットを全て網羅したリスト、解説については、関数 format のリファレンスページを参照してください。出力書式をもっと制御したい場合には、sprintf や fprintf を使用して下さい。

スペーシングの制御． 出力のスペーシングを制御するには、text display プリファレンスや関数 format を利用します。

```
format compact
```

は出力からブランクラインを削除し、コマンドウィンドウ上でより多くの情報を参照することを可能にします。逆に出力をより読み易くするために、ブランクラインを出力に含めたい時は、つぎのようにタイプします。

```
format loose
```

コマンドウィンドウの表示内容を印刷

コマンドウィンドウの表示内容を印刷するには、**ファイル (File)** メニューから**印 (Print)** を選択します。選択した部分のみを印刷したい場合には、コマンドウィンドウ上で選択を行った後に、**ファイル (File)** メニューから**選択したものの印刷 (Print Selection)** を選択します。

プログラムの実行

M- ファイルの実行

M- ファイルの実行、MATLAB 言語コードを含むファイルである M- ファイルを実行するには、他の MATLAB 関数を実行する場合と同様の方法を用います。すなわち、コマンドウィンドウで、M- ファイルの名前をタイプし、**Enter** または **Return** キーを押せば、その M- ファイルを実行することができるのです。

M- ファイルに含まれる関数を、その実行と同時に個々表示するには、**Display** から **Echo on** をチェックするか、echo 関数を on に設定します。

プログラムの実行を中断

Ctrl+c または **Ctrl+Break** を押せば、いつでもプログラムの実行を中断できます。

Windows プラットフォーム上では、Ctrl+c を押しても、組み込み関数の実行が完了するか、または、MEX- ファイルが演算を完了するまでプログラムの実行は中断されません。UNIX システム上では、この操作によってプログラムの実行は直ちに終了します。

外部プログラムの実行

感嘆符！は、シェルエスケープで、その後に続く入力はオペレーティングシステムへのコマンドとして解釈されます。感嘆符 ！ を使用すれば、MATLAB を終了せずに、他のプログラムやユーティリティを実行することができます。たとえば、上では、

```
!vi darwin.m
```

とタイプすると、darwin.m という名称のファイルを vi エディタを使って呼び出します。そして、ユーザがこのプログラムを終了すると、オペレーティングシステムは、MATLAB に戻ります。結果とステータスを出力する外部プログラムの実行については、オンラインヘルプの関数 unix、及び、dos に関する項を参照してください。

M- ファイルを開く

M- ファイルを開くには、コマンドウィンドウで開きたいファイル、または、関数の名前を選択し、右クリックを行った後、コンテキストウィンドウから**選択したものを開く (Open Selection)** を選択してください。M- ファイルは、エディタ、または、デバッガ上で開かれます。

エラーの吟味

M- ファイルの実行中にエラーメッセージが出た場合には、カーソルをエラーメッセージの上に動かして、**Enter** または **Return** キーを押して下さい。それにより、不当な M- ファイルが、エディタ上で開かれ、エラーを含むラインがスクロール表示されます。

セッションログの記録

関数 diary

関数 diary は、ディスクファイルにユーザが、MATLAB を起動していた時の履歴を、ユーザがキーボードで何を入力し、それに対してシステムがどのような反応を返したかに至るまで逐一記録します (但し、表示されたグラフィックスの記録は行いません)。これにより、ユーザは何らかのワードプロセッサを使って作成したテキストファイルを、表示、編集することが可能となります。ユーザが入力した全ての関数と、その入力に対する MATLAB の出力結果からなる sept23.out という名称のファイルをディスク上に作成するには、つぎのようにタイプします。

```
diary('sept23.out')
```

セッションの履歴の記録を中止するには、つぎのようにタイプします。

```
diary('off')
```

他のセッションログ

セッションの情報を参照するための上記以外の方法として、つぎの 2 つの方法があります。

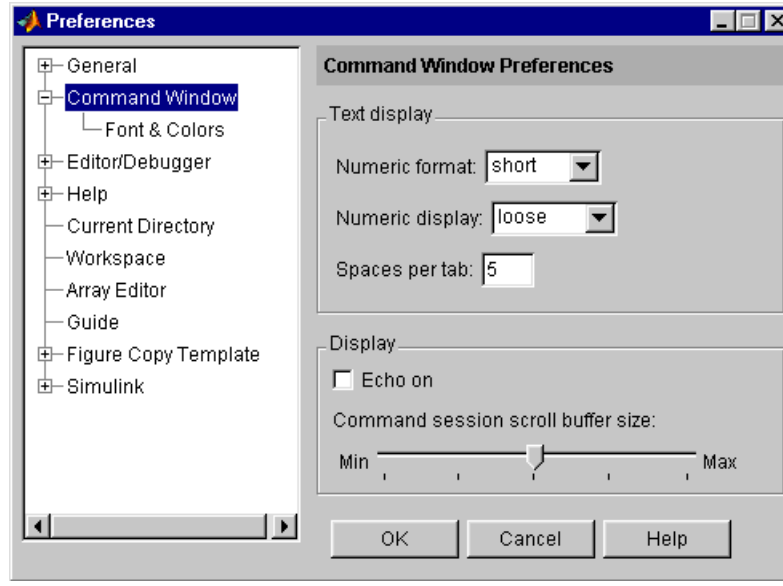
- カレント、または、その前のセッションで実行された全ての関数のログを示すコマンド履歴 (Command History) ウィンドウ。

- Windows プラットフォーム上では、logfile をスタートアップのオプションとして追加する。– 1-5 ページの "Windows プラットフォームにスタートアップオプションの追加 " を参照してください。

コマンドウィンドウの設定

様々な設定を使用して、ユーザは、数値の表示形式を指定したり、それぞれのセッションで実行される関数が自動的に表示されるよう設定したり、コマンドウィンドウ上に表示されるフォントのタイプ、スタイル、サイズ、強調する部分に用いられる色などの指定を行うことができます。

コマンドウィンドウに関する様々な設定を行うには、コマンドウィンドウの**ファイル (File)** メニューから**プリファレンス (Preferences)** を選択してください。**プリファレンス (Preferences)** ダイアログボックスをオープンすると、**コマンドウインドウプリファレンス (Command Window Preferences)** が表示されます。



コマンドウインドウに対するテキスト表示と表示の設定

テキスト表示 (Text display). コマンドウィンドウ上にどのように出力結果を表示するかを指定します。

- 数値フォーマット – コマンドウィンドウに表示される数値の出力書式を指定します。ここで指定された書式は、数値の表示書式に対してのみ適用され、MATLAB の計算結果、及び、MATLAB がセーブした情報に対しては適用されません。関数 format のリファレンスページに利用可能な書式のリストが記載されています。
- 数値表示 – コマンドウィンドウ上の出力をいかにスペーシングするかについて指定します。ブランクラインを省略したい時には、compact を選択します。またブランクラインを表示したい場合には、loose を選択します。数値出力のスペーシングの詳細は、関数 format のリファレンスページを参照してください。
- 1 タブによるスペース数 – 出力表示を行う際のタブ位置として割り当てられる空白の数を指定します。デフォルトは、4 です。

表示 . 反復表示に関するオプション及びバッファサイズを指定します。

- **Echo on** – このボックスをチェックすれば、M- ファイルを実行しながら、M- ファイルに含まれる関数を実行と同時にコマンドウィンドウ上に表示することが可能となります。詳細は、関数 echo のリファレンスページを参照してください。
- コマンドセッションスクロールバッファサイズ – コマンドを再び呼び出す際に使用される、以前に実行したコマンドのリストを記憶するバッファのサイズを指定します。この点に関するさらなる情報については、3-7 ページの " 最近タイプしたラインを再び呼び出す " を参照してください。

コマンドウィンドウ上の表示に適用されるフォントと色の設定

フォント . コマンドウィンドウ上で用いられるフォントを指定するには、コマンドウィンドウのフォント設定を用います。**General Font & Colors** 設定で指定したフォントと同一のフォントをコマンドウィンドウ上の表示にも適用したい場合には、**デスクトップフォントの使用 (Use desktop font)** を選択します。一方、コマンドウィンドウ上の表示には別途異なるフォントを適用したい時には、**カスタムフォントの使用 (Use custom font)** を選択し、コマンドウィンドウで使用するフォントを指定します。

- タイプ、たとえば、Sans Serif
- スタイル、たとえば、Bold
- ポイント数で表したサイズ、たとえば、12 ポイント

フォントの指定を行うと、**サンプル (Sample)** エリアに、フォントがどのように表示されるかが表示されます。

カラー . コマンドウィンドウで用いられるカラーを指定します。

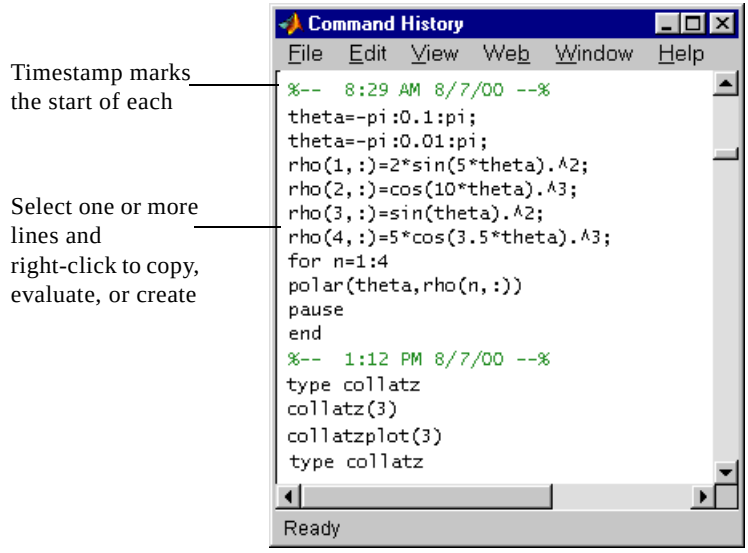
- **テキストカラー** – 通常のテキスト表示に用いられる色を指定します。強調されている部分に適用される色については、**シンタックスの強調表示 (Syntax highlighting)** で指定します。
- **バックグラウンドカラー** – ウィンドウの背景色を指定します。
- **Syntax highlighting** – 強調されている部分に適用される色を指定します。このボックスがチェックされているケースで強調色を指定するには、**カラーの設定 (Set Colors)** をクリックします。シンタックスの強調に関する詳細は、3-5 ページの " シンタックスの強調 " を参照してください。

コマンド履歴

MATLAB を最初に実行すると、コマンド履歴 (Command History) ウィンドウが表示されます。このコマンド履歴 (Command History) ウィンドウは、コマンドウィンドウ上で最近実行した関数のログを表示します。コマンド履歴 (Command History) を表示、または、消去するには、**ビュー (View)** メニューを使用します。詳細については、2-7 ページの " デスクトップツールのオープンとクローズ " を参照してください。

コマンド履歴 (Command History) ウィンドウを用いると、つぎのことが可能となります。

- 3-16 ページの " コマンド履歴 (Command History) ウィンドウに関数を表示 "
- 3-17 ページの " コマンド履歴 (Command History) ウィンドウから関数を実行 "
- 3-17ページの"コマンド履歴(Command History)ウィンドウから関数をコピー"



コマンド履歴 (Command History) ウィンドウに関数を表示

コマンド履歴 (Command History) ウィンドウに表示されるログには、カレントセッションで実行されている関数は、もちろんのこと、その前のセッションで実行された関数も記録されています。また、それぞれのセッションが実行された時間及び日付も、当該セッションで実行された関数の履歴のトップに表示されます。ス

クロールバー、または、↑ ↓キーによって、コマンド履歴 (Command History) ウィンドウの表示を自由にスクロールさせることも可能です。

コマンド履歴 (Command History) ウィンドウ上で入力を削除

入力の数が多すぎたり、望ましい結果を求めるにはその入力では不適切であるような場合があります。そのような時には、コマンド履歴 (Command History) ウィンドウ上で、入力を削除してください。ユーザが入力値を削除するまで、全ての入力は保持されます。

コマンド履歴 (Command History) ウィンドウ上で入力を削除するには、先ず削除対象となる関数を選択してください。この時複数の関数を選択する場合、**Shift**- クリックまたは **Ctrl**- クリックのいずれかの操作を行ってください。関数の選択を行った後、右クリックを行い、コンテクストメニューに表示されるつぎの選択肢から１つ選択してください。

- **Delete Selection** – 選択した関数を削除します。
- **Delete to Selection** – 選択した関数よりも前に実行した全ての関数を削除します。
- **Delete Entire History** – コマンド履歴 (Command History) ウィンドウ上の全ての関数を削除します。

全ての履歴を削除するもう１つの方法として、**編集 (Edit)** メニューから**コマンド履歴のクリア (Clear Command History)** を選択するという方法も利用できます。

コマンド履歴 (Command History) ウィンドウから関数を実行

コマンド履歴(Command History)ウィンドウ上に表示される全ての関数について、その関数をダブルクリックすることによって実行することができます。たとえば、myfile.m をエディタで開くには、edit myfile をダブルクリックします。また、実行したい関数の上で右クリックし、そこで表示されたコンテクストメニューから**選択したものを計算 (Evaluate Selection)** を選択することによっても、関数を実行することができます。さらにつぎのセクションで説明する方法で、関数をコマンドウィンドウにコピーして実行することも可能です。

コマンド履歴 (Command History) ウィンドウから関数をコピー

コマンド履歴 (Command History) ウィンドウから関数をコピーするには、つぎの操作を行います。先ず関数を選択します。複数の関数を選択するには、**Shift**- ク

リックまたは Ctrl- クリックを行います。関数を選択した後、つぎのいずれかの操作を実行することができます。

操作	操作を行うための方法
コマンドウィンドウ上で関数を実行	コピーしたい関数の上で右クリックし、表示されたコンテクストメニューから**コピー (Copy)** を選択し、関数をクリップボードにコピーする。つぎに、クリップボードにコピーした関数をコマンドウィンドウ内にペーストする。関数をエディットしたい場合、エディット操作を行い、Enter または Return キーを押すと、コピーした関数が実行されます。
他のウィンドウに関数をコピー	コピーしたい関数の上で右クリックし、表示されたコンテクストメニューから**コピー (Copy)** を選択し、関数をクリップボードにコピーする。つぎに、エディタ、または、他のアプリケーションで開いている M- ファイルにコピーした関数をペーストする。
選択した関数から M- ファイルを生成	選択した関数の上で右クリックし、表示されたコンテクストメニューから**M-ファイルを作成 (Create M-File)** を選択します。この操作により、エディタは、ユーザがコマンド履歴 (Command History) ウィンドウで選択した関数からなる新しい M- ファイルを開きます。

4

ヘルプの参照

情報のタイプ 4-3

ヘルプブラウザの使用 4-4

ヘルプナビゲータの使用 4-6

ヘルプ表示ウィンドウにマニュアルを表示 4-14

ヘルプブラウザの設定 4-18

マニュアルの印刷 4-23

関数 Help の使用 4-25

ヘルプを参照するその他の方法 4-28

The MathWorks は、全ての製品について数冊の印刷されたマニュアルとオンラインヘルプを提供しています。オンラインのマニュアルには、印刷版に含まれない、あるいは、印刷版よりも新しい情報が含まれている場合があります。

オンラインヘルプにアクセスする主な方法として、つぎの方法があります。

- 4-4 ページの " ヘルプブラウザの使用 " - ヘルプブラウザを使用すると、ユーザが所有する MathWorks 製品に関する情報を検索、表示することができます。索引表示、グローバルインデックス、サーチなどの機能がヘルプブラウザには用意されています。
- 4-25 ページの "関数 help の使用 " - help functionname とタイプして、M- ファイルヘルプが表示されます。このヘルプは、関数の簡単な説明と使用法をコマンドウィンドウ上に表示します。他のヘルプ関数も利用することができます。
- 4-28 ページの " ヘルプを参照するその他の方法 " - その他にもヘルプを参照するために、製品固有のヘルプ機能 (デモの実行など) を使用することもできます。また、テクニカルサポートとコンタクトしたり、他の MathWorks 製品のマニュアルを検索したり、他の本のリストを表示したり、MATLAB ニュースグループに参加する、等の方法も活用できます。

オンラインヘルプの活用に加えて、マニュアルを印刷することも可能です。- 4-23 ページの " マニュアルの印刷 " を参照してください。

情報のタイプ

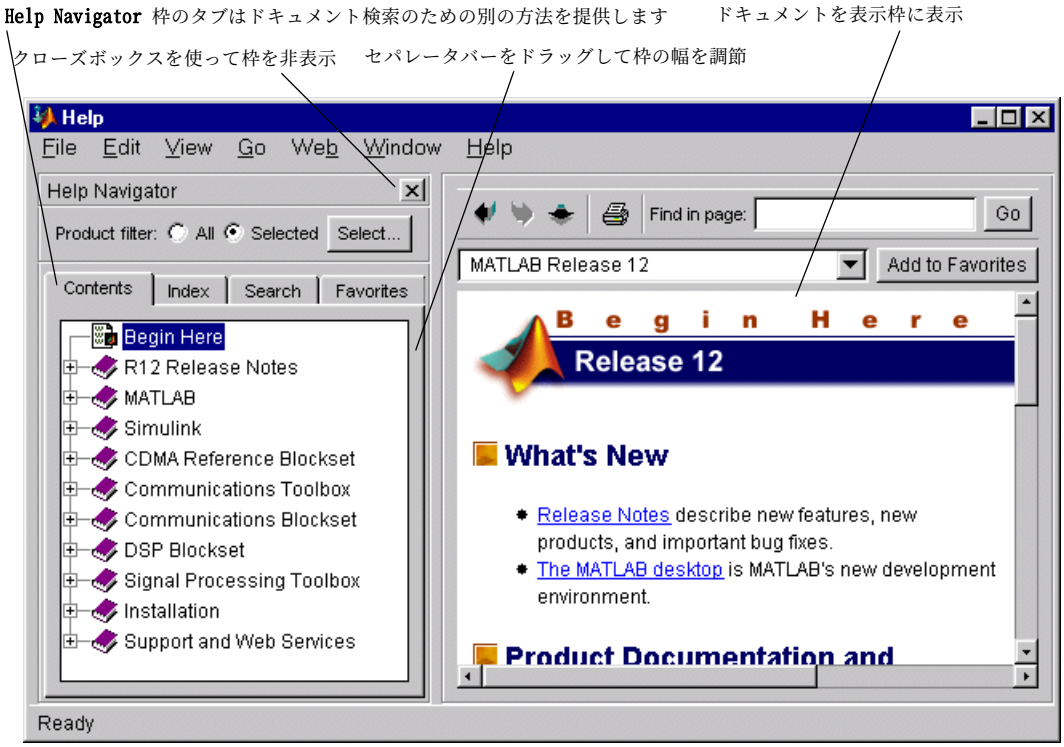
ヘルプブラウザとヘルプ関数によって、つぎのタイプのマニュアルにアクセスできます。この中で、ユーザのニーズに合ったマニュアルを活用してください。

- リリースノート - このリリースで新たに用意された製品と機能の概要を提供します。リリースノートには、アップグレード情報や既知の問題、制約が記載されます。新しいリリースを使い始めるときには、ユーザの製品のリリースノートを、もう一度見返してみることをお薦めします。
- さあ始めましょう - 初心者向けのマニュアルです。このマニュアルは、製品の主要機能の解説が記載されています。製品、または、機能を使い始めるときは、是非 " さあはじめよう " を読み返してみることをお薦めします。より多くの情報を得るには、チュートリアルインフォメーションかリファレンスページを活用してください。
- Using ... とチュートリアルについて - このマニュアルには、製品の概要のみならず製品を使用する上で必要な情報が完全網羅されています。" さあはじめよう " を読み終えたら、つぎはこのマニュアルを参照してください。
- リファレンスページ - 全ての関数について、使用法、説明、例題、及び、その他の情報を記載したリファレンスページが用意されています。このマニュアルは、関連する関数や追加的な情報にもリンクしています。リファレンスページは、ブロックについても提供されています。関数とその使用法に関する情報を得るには、リファレンスページを活用してください。
- M- ファイルヘルプ - M- ファイルヘルプを使って、関数に関する基本的な情報にすぐにアクセスすることができます。M- ファイルヘルプは、関数と使用法に関する簡単な解説を提供します。これは、関数に対して、M- ファイルの最初の部分に一連のコメントを記載しているので、このヘルプを構成するテキストを、M- ファイルヘルプと呼びます。
- オンライン情報データベース - MathWorks テクニカルサポートオンライン情報データベース。ユーザから提起された質問にして回答します。

ヘルプブラウザの使用

ヘルプブラウザを使って、MATLAB、及び、ユーザの所有する MathWorks 製品のマニュアルを検索、表示することができます。ヘルプブラウザは、MATLAB デスクトップに統合された Web ブラウザで、HTML ドキュメントを表示することが可能です。

ヘルプブラウザを開くには、ツールバーのヘルプボタン?をクリックするか、コマンドウィンドウで、helpbrowser とタイプしてください。**ビュー(View)** メニューから**ヘルプ (Help)** を選択するか、または、ツールの中の **Help** メニューを使って、ヘルプブラウザにアクセスできます。



ヘルプブラウザは、2 つの表示枠 (パン) で構成されています。:

- 左側の枠は、ヘルプナビゲータで、情報を検索するときに使用します。ヘルプナビゲータは、**プロダクトフィルタ (Product Filter)**、**目次 (Contents)**、**インデックス (Index)**、**サーチ (Search)**、**お気に入り (Favorites)** タブによって構成されます。詳細については、4-6 ページの " ヘルプナビゲータの使用 " を参照してく

ださい。

- ブラウザの右側の枠は、ヘルプ表示ウィンドウで、マニュアルが表示されます。

ヘルプブラウザのサイズの変更

2 つの表示ウィンドウの相対幅を調整したい場合、2 つのウィンドウを隔てているセパレータバーをドラッグしてください。また、ウィンドウに表示されるフォントを変更することもできます。フォントの変更は、4-20 ページの " ヘルプフォントの設定 - フォントの名前、スタイル、サイズを指定 " を使用してください。

参照したいマニュアルを検出した後であれば、**ヘルプナビゲータ (Help Navigator)** ウィンドウを閉じてマニュアル自体を表示するスペースをより広くすることも可能です。下図のように使用できます。**ヘルプナビゲータ (Help Navigator)** を閉じるには、ヘルプナビゲータウィンドウの右隅にあるクローズボックス をクリックするか、**View->Help View Options-> Show Help Navigator** をメニューから選択し、チェックを外してください。ヘルプ表示ウィンドウから **ヘルプナビゲータ**をオープンするには、ヘルプブラウザの左上隅にあるヘルプナビゲータボタン **View->Help View Options->Show Help Navigator** を選択し、チェックを入れるかのいずれかの操作を行ってください。

このイラストのように表示枠のみを表示するには、**ヘルプナビゲータ (Help Navigator)** 枠のクローズボックスをクリックします。

このボタンをクリックして **ヘルプナビゲータ (Help Navigator)** 枠を表示します。

ヘルプナビゲータの使用

ヘルプブラウザの左枠に表示されるヘルプナビゲータを使って、オンラインヘルプ上の情報を検索することができます。主要機能の説明については、つぎの節を参照してください。4-6 ページの " プロダクトフィルタの使用 " - 指定した製品についてのみマニュアルを表示します。

- 4-7ページの"ヘルプブラウザに目次リストの表示" - マニュアルの目次を表示します。目次は、拡張可能です。
- 4-9 ページの " インデックスを使って、マニュアルの検索 " - キーワードを使って、情報を検索します。
- 4-11 ページの " マニュアルの検索 " - フルテキスト及び他の検索フォームを使って、マニュアルを検索します。
- 4-13 ページの " お気に入りのページをブックマークに登録する " - 後で参照するお気に入りページを指定します。

プロダクトフィルタの使用

ヘルプナビゲータ (Help Navigator) の**プロダクトフィルタ (Product filter)** を使って、指定した製品についてのみマニュアルを表示することができます。

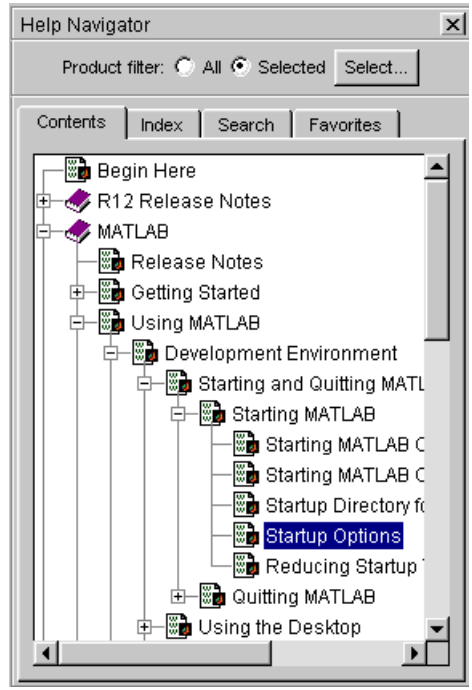
ユーザのシステムにインストールされた全ての MathWorks 製品についてのマニュアルを表示したい場合、**All** を選択してください。

ユーザのシステムにインストールされた MathWorks 製品の一部についてマニュアルを表示したい場合、**Help Navigator** の **Select** ボタンをクリックし、プロダクトフィルタに対するヘルプリファレンスを使って製品のサブセットを指定します。ヘルプナビゲータで **Product filter** を **Selected** に設定すると、つぎのようになります。

- **目次 (Contents)** は、指定した製品のサブセットについてのみ、目次を表示します。
- **インデックス (Index)** は、指定した製品のサブセットについてのみ、インデックスを表示します。
- **サーチ (Search)** 機能は、指定した製品のサブセットについてのみ、有効となります。

ヘルプブラウザに目次リストの表示

全ての製品マニュアルについて、タイトルリストと目次リストを表示するには、**ヘルプナビゲータ**パンの**目次 (Contents)** タブをクリックします。



目次 (Contents) リストでは、つぎの操作を行うことができます。

- ある特定のアイテムについてリスト表示を広げるには、+ をクリックしてください。
- ある特定のアイテムについてリスト表示を隠すには、- をクリックしてください。
- アイテムを選択すれば、マニュアルのトップページか、セクションのトップページをページ表示ウィンドウに表示します。
- アイテムをダブルクリックすると、そのアイテムについて、リスト表示が拡張され、マニュアルのトップページ、または、セクションのトップページをページ表示ウィンドウに表示します。

- 下矢印と上矢印を使って、リストの中を移動することができます。

目次リストは、ユーザのシステムにインストールされた全製品のマニュアルを表示します。しかし、**プロダクトフィルタ (Product filter)** を **Selected** に設定した場合、選択した製品のマニュアルのみしか表示しません。

プロダクトロードマップ

MATLAB または Communications Toolbox のような製品 (本のアイコン で表示されます) を**目次表示 (Contents)** ウィンドウで選択したとき、*ロードマップ*は、その製品に最も関係のあるマニュアルを提供するものです。ロードマップでは、さらにつぎの内容へのリンクも提供されます。

- 対象製品の例題インデックス
- 印刷に適した PDF バージョンのマニュアル

目次表示ウィンドウとページ表示ウィンドウとの同期をとる

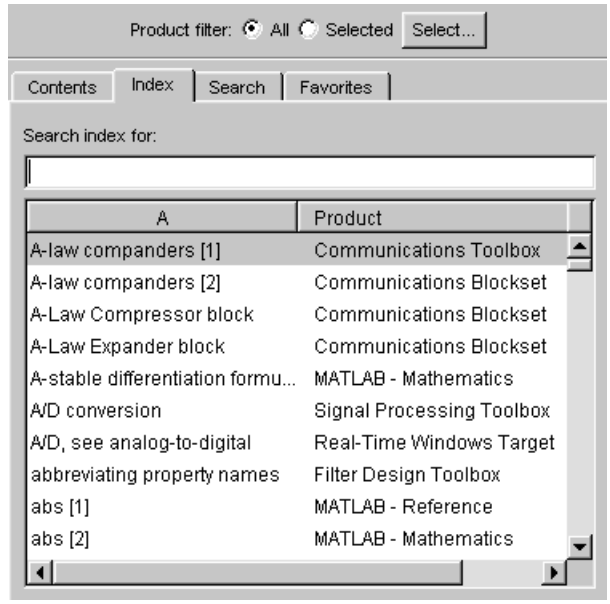
デフォルトでは、**目次表示 (Contents)** ウィンドウで強調表示されたトピックは、常にページ表示ウィンドウに表示されるタイトルと整合性を保っています。すなわち、目次リストは表示されるドキュメントと同期しています。この機能は、目次表示ウィンドウ以外の方法でマニュアルにアクセスする場合 (たとえば、ページ表示ウィンドウに表示されたページに組み込まれたリンクを使用する場合など) に、便利な機能です。同期を使って、表示されたページが、どのマニュアルのどの節のページなのかを、知ることができるのです。

同期を無効にすることも可能です。同期を無効にするには、つぎの設定を使用してください。

同期が**目次表示 (Contents)** ウィンドウにのみ適用されることに注意してください。ページ表示ウィンドウに表示されたページは、**サーチ**、**インデックス**、**お気に入り**タブで選択した対象と必ずしも一致しません。

インデックスを使って、マニュアルの検索

入力した特定の索引語（選択したキーワード）を、MathWorks マニュアルで検索するには、ヘルプナビゲータウィンドウの**インデックス (Index)** タブをクリックしてください。



1. **プロダクトフィルタ (Product filter)** を **All**、または、**Selected** に設定します。
2. **インデックス (Index)** タブをクリックします。
3. **Search index for** に、1 つ、または、複数の単語をタイプします。入力すると、インデックスは、整合する見出し語、副見出し語をインデント表示します。表示されるまでに、数分かかる場合もあります。

   整合する見出し語を含む製品、及び、マニュアルは、索引見出し語の隣にリスト表示されます。これは、整合する見出し語を含む製品、マニュアルが複数あった場合に便利な機能です。ヘルプナビゲータウィンドウを広げて、製品、及び、マニュアルを参照し易くすることも可能です。

4 リストから見出し語を選択し、対象のページの表示

これにより、ページ表示ウィンドウに対象のページが表示され、見出し語が使用されている箇所へと自動的にスクロールします。

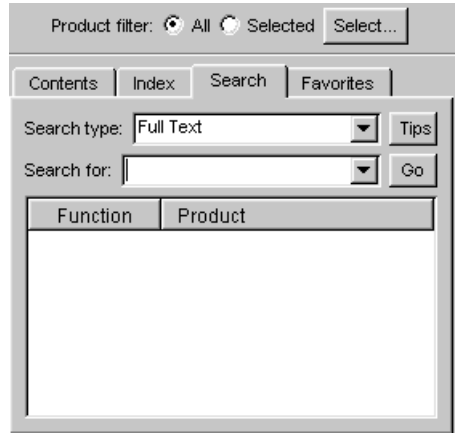
5 より多くの整合する見出し語を参照したい場合、検索結果をスクロールしてください。

索引を使うヒント． 検索の結果表示された見出しアイテムに満足がいかない場合、つぎの操作を試してください。

- **ヘルプナビゲータ (Help Navigator)** ウィンドウをより広げると、索引見出し語に関連する製品、マニュアルの名前を参照することができます。
- 表示されたページから語句を検索するには、**ページで検索 (Find in page)** フィールドを使用します。
- 異なる語句をタイプするか、先にタイプした語句の順序を変えてもう一度検索します。たとえば、writing M-files を探すときに、writing M-files とタイプするのではなく、M-files とタイプし、副見出し語に、creating を割り当てて検索する、といった操作を試してみましょう。
- **プロダクトフィルタ (Product filter)** を **Selected** に選択している場合、それを **All** に変更すると、より多くの結果を参照することができます。逆に、**All** と設定していた場合、それを **Selected** に変更すると表示される検索結果は、より少なくなります。
- サーチタブをクリックします。- 詳しくは、4-11 ページの " マニュアルの検索 " を参照してください。

マニュアルの検索

マニュアルから特定の語句を検索するには、ヘルプナビゲータウィンドウの**サーチ (Search)** タブをクリックします。



1 **プロダクトフィルタ (Product filter)** を、**All** または、**Selected** に設定します。

2 **Search** タブをクリックします。

3 **Search type** を、つぎの中から選択します。

- **Full Text** は、全文検索します。
- **Document Titles** は、指定した見出しの部分のみ検索します。
- **Function Name** は、関数リファレンスページから検索します。この検索は、関数 doc と同等の検索機能を提供します。
- **Online Knowledge Base** は、MATLAB Web サイトに接続し、テクニカルサポート情報を検索します。

4 検索したい 1 つ、または、複数の語句を **Search for** フィールドに入力し、**Go** をクリックします (または、**Enter** あるいは **Return** キーを押します)。

この操作により、検索語句を含むドキュメントが製品毎にまとめられてリスト表示されます。ステータスバーは、検索語を含むページ数を表示します。各検索結果について、検索語を含むマニュアルの**タイトル (Title)**、**節 (Section)**

が表示されます。**ヘルプナビゲータ (Help Navigator)** ウィンドウを広げれば、**節 (Section)** の名称も参照することができます。

5 検索結果のリストから見出し語を選択してください。

この操作により、検索語を含むページがページ表示ウィンドウに表示され、そのページで検索語が最初に出現する箇所を強調表示します。

6 より多くの検索結果を参照するには、リストをスクロールしてください。

検索に関するアドバイス

検索結果に満足がいかない場合、つぎのヒントを試してみてください。

- **ページ内の検索 (Find in page)** フィールドを使用すると、表示されたページで検索語、または、他の語句が使われている箇所をよりたくさん検索することができます。
- 検索結果のリストをスクロールしてください。検索結果は、各製品毎にまとめられ、製品名がラベル表示されたバナーで製品グループが区別されています。
- **ヘルプナビゲータ (Help Navigator)** ウィンドウを広げれば、2列目の情報を参照できます。ウィンドウの 2 列目には、検索語が使用されているマニュアルの節が表示されています。
- **インデックス (Index)** を使用してみてください。演算子や特殊なキャラクタについては、通常の検索では**検出 (Search)** できないので、**インデックス (Index)** を使う必要があります。インデックスで検索した結果は、数の上では少数ですが、より信頼性の高いものとなります。これは、検索結果は指定した語句をどのような形であっても含んでいればそのページを表示するのに対し、インデックス検索では、指定した語句がキーワードになっているからです。
- **タイプの検索 (Search type)** を変更してください。たとえば、**関数名 (Function name)** をタイプの検索 (search type) に指定すると、関数のリファレンスページが検索対象となります。
- 検索対象を絞るには、たとえば、font preferences のように、**検索 (Search for)** フィールドに 1 つ以上の単語をタイプします。1 つ以上の単語で検索すると、検索語間で論理演算 AND が実行されます。たとえば、font preferences とタイプすると、font と preferences という 2 つの語を含むページのみ表示されます。ただし、そのページの全てが、"font preferences" という完全な語句を含んでいるとは限りません。
- より多くの検索結果を得るには、いろいろな検索語を試してみてください。これは、検索は、正確に整合するものを表示するためです。たとえば、preferences

を検索して、対象となるものが検出されない場合、preference とタイプしてもう一度検索してみることをお薦めします。

- **プロダクトフィルタ (Product filter)** が **Selected** に設定されている場合、より多くの結果を参照するために **All** に変更してください。逆に、**All** と設定していた場合、それを、**Selected** に変更すると表示される検索結果は、より少なくなります。

お気に入りのページをブックマークに登録する

ヘルプナビゲータ (Help Navigator) の**お気に入り (Favorites)** タブをクリックすると、以前お気に入りに指定したドキュメントのリストを表示することができます。お気に入りリストでは、つぎの操作を行うことが可能です。

- お気に入りを選択 - 選択したドキュメントがページ表示ウィンドウに表示されます。
- お気に入りを削除 - お気に入りリストに表示されたアイテムを右クリックし、コンテキストメニューから**削除 (Remove)** を選択するか **Delete** キーを押します。
- お気に入りの名前を変更 - お気に入りリストに表示されたアイテムを右クリックし、コンテキストメニューから**リネーム (Rename)** を選択します。新しい名前をタイプして、既存の名前をどのような名前に変更するかを決定してください。

お気に入りに追加

ドキュメントページをお気に入りに指定するには、つぎのいずれかの操作を行ってください。

- ページがページ表示ウィンドウに表示されている場合、表示パンツールバーの**お気に入りに追加 (Add to Favorites)** ボタンをクリックしてください。
- コンテキストが表示されている場合、コンテキストメニューから**お気に入りに追加 (Add to Favorites)** を選択してください。
- ヘルプブラウザにインデックス検索結果、または、検索結果が表示されている場合、表示されたコンテキストメニューから**お気に入りに追加 (Add to Favorites)** を選択してください。

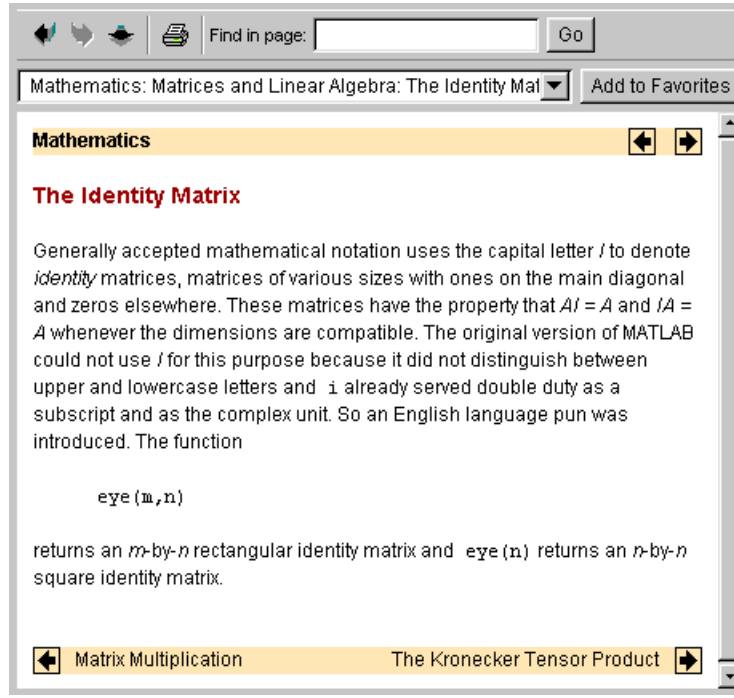
ヘルプ表示ウィンドウにマニュアルを表示

ヘルプナビゲータ (Help Navigator) でマニュアルを検索した後、表示パンの中で、マニュアルを見ることができます。マニュアルを表示する場合につぎの機能を利用することができます。

- 4-15 ページの " 他のページをブラウズ "
- 4-16 ページの " ページをブックマークに登録 "
- 4-16 ページの " ページの再表示 "
- 4-16 ページの " ページの印刷 "
- 4-16 ページの " 表示されたページから語句の検索 "
- 4-16 ページの " 情報のコピー "
- 4-17 ページの " 選択結果の評価 "
- 4-17 ページの " ページソースを HTML で表示 "
- 4-17 ページの "Web ページの表示 "

他のページをブラウズ

ページとツールバーの矢印ボタンを使って他のページに移動します。



マニュアルのつぎのページを表示するには、ページの上部、または、下部に表示されている右矢印をクリックします。前のページを表示するには、ページの上部、または、下部にあるをクリックします。ページの下部の矢印には、クリックによってジャンプするページのタイトルがラベル表示されています。

前に表示していたページに戻るには、ヘルプ表示ウィンドウツールバーのバックボタンをクリックします。バックボタンを使った後、つぎのページを表示するには、ヘルプ表示ウィンドウツールバーの進むボタンをクリックします。ページ上で右クリックし、表示されたコンテキストメニューから**後ろ (Back)**、または、**前へ (Forward)** を選択するか、**進む (Go)** メニューから**後ろ (Back)** または、**前へ (Forward)** を選択する方法でも、前のページ、または、つぎのページを表示させることができます。

ページをブックマークに登録

ヘルプ表示ウィンドウツールバーの**お気に入りに追加 (Add to Favorites)** ボタンをクリックすると、現在表示しているページがお気に入りリストに追加されます。

ページの再表示

カレントの MATLAB セッションで以前表示したページをもう一度表示するには、ヘルプ表示ウィンドウツールバーのドロップダウンリストから表示したいページのタイトルを選択してください。

ページの印刷

カレントで表示されているページを印刷する方法については、4-23 ページの " ヘルプブラウザからページの印刷 " を参照してください。

表示されたページから語句の検索

現在表示されているページから語句を検索するには、つぎの操作を行ってください。

1 ヘルプ表示ウィンドウツールバーの**ページの検索 (Find in page)** フィールドに検索したい語句をタイプし、**Enter** または、**Return** キーを押すか、**Go** をクリックしてください。ユーザは、語句の一部をタイプして検索することもできます。たとえば、preference とタイプして、preference と preferences という語句の全てを検索することも可能です。

   検索すると、ページが検索対象の語句を含む位置まで自動的にスクロールし、検索対象を強調表示します。

2 続いて、再び、**Enter**、または、**Return** キーを押すと、そのページ上で、つぎにその語句が含まれている位置の検索を行います。

1ページではなく全マニュアルを対象に検索を行う際の方法については、4-11ページの " マニュアルの検索 " を参照してください。

情報のコピー

ヘルプ表示ウィンドウから情報をコピーするには、最初にコピーしたい情報を選択します。つぎに、右クリックで表示されたコンテキストメニューから**コピー (Copy)** を選択するか、**編集 (Edit)** メニューから**コピー (Copy)** を選択します。コ

マンドウィンドウなどの他のツールから情報をペーストしたり、ワープロなど他のアプリケーションに情報を移動することも可能です。

選択結果の評価

マニュアルに表示されているコードの例題を実行するには、先ず、ヘルプ表示ウィンドウでコードを選択してください。つぎに、右クリックを行い、コンテキストメニューから**選択結果の評価 (Evaluate Selection)** を選択するか、Go メニューから**選択結果の評価 (Evaluate Selection)** を選択してください。この操作により、選択した関数がコマンドウィンドウ上で実行されます。

ページソースを HTML で表示

現在表示されているページの HTML ソースを表示するには、**View -> Help View Options -> Page Source** を選択してください。これにより、エディタ / デバッガ上に、HTML でページが表示されます。ここで、HTML を修正したりコピーしたりすることができます。修正したページを確認するには、ヘルプ表示ウィンドウツールバーの再読み込みボタンをクリックするか、Go メニューから**再読込 (Reload)** を選択してください。

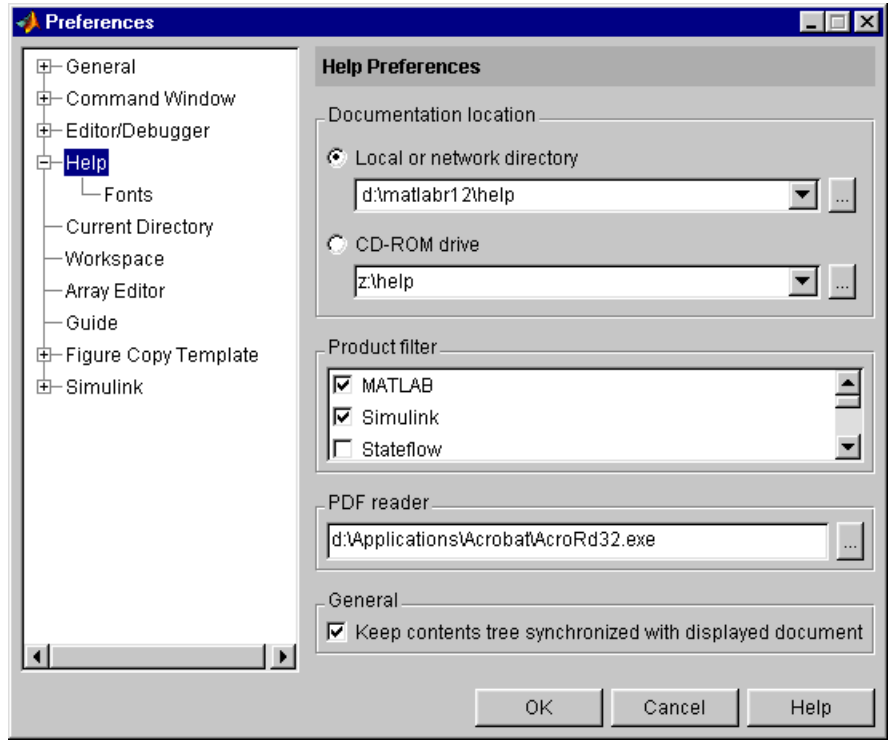
Web ページの表示

ヘルプブラウザを使って、Web ページを表示することもできます。ヘルプ表示ウィンドウツールバーのタイトルフィールドに URL をタイプして、**Enter** キーを押してください。ここでは、URL を完全な状態 (たとえば、http://www.mathworks.com.) で入力するようにしてください。この操作で、指定した Web ページがヘルプブラウザ上に表示されます。

ヘルプブラウザの設定

ユーザ設定を使って、ヘルプファイルの位置、ヘルプブラウザで使用するフォント、ヘルプブラウザにマニュアルを取り込む製品を指定することができます。

ヘルプブラウザの設定を変更するには、**ファイル (File)** メニューから**ユーザ設定 (Preferences)** を選択します。**ユーザ設定 (Preferences)** ダイアログボックスが開き、**ユーザ設定ヘルプ (Help Preferences)** を表示します。



ここで、設定を決定して、**OK** をクリックしてください。ヘルプブラウザのユーザ設定では、つぎの設定を変更できます。

- 4-19 ページの " マニュアルの設置位置 -help ディレクトリの指定 "
- 4-19ページの "プロダクトフィル - どの製品のマニュアルを使用するかを決定する "

- 4-20 ページの "PDF Reader - PDF reader の位置の設定 "
- 4-20 ページの "General - 目次表示ウィンドウと表示ページを同期させる "
- 4-20 ページの " ヘルプフォントの設定 - フォントの名前、スタイル、サイズを指定 "

マニュアルの設置位置 -help ディレクトリの指定

マニュアルの設置位置 (Documentation location) 設定では、MATLAB の help ディレクトリがユーザのシステムのどこにあるのか、その位置を指定します。help ディレクトリは、ヘルプブラウザで使用するオンラインヘルプファイルで構成されています。

- MATLAB のインストール時にヘルプファイルをインストールすると、マニュアルの設置位置は、既にヘルプファイルの場所と対応する値に設定されています。ヘルプファイルの位置が変わった場合や、異なるヘルプファイルにアクセスしたい場合、**ローカル、または、ネットワークディレクトリ (Local or network directory)** に**マニュアルの設置位置 (Documentation location)** を選択してください。... ボタンを使用すると、ファイルシステムをブラウズできます。新しい位置を選択する際に使用してください。
- MATLAB インストール時に、ヘルプファイルをインストールしなかった場合は、ヘルプブラウザはマニュアル CD からファイルを検索する必要があります。そのため、**マニュアルの設置位置 (Documentation location)** は、**CD-ROM drive** に設定しておく必要があります。... ボタンを使用すると、ファイルシステムをブラウズできます。CD ドライブの位置を選択する際に使用してください。

Java GUI をサポートしていない UNIX プラットフォームでは、関数 docopt を用いて、help ディレクトリの位置を指定してください。

プロダクトフィル - どの製品のマニュアルを使用するかを決定する

MATLAB の他に Simulink、ツールボックスなどの MathWorks 製品をもっている場合、**プロダクトフィルタ (Product filter)** を使うと、使用するマニュアルを制限することができます。たとえば、検索を行って、求める情報が Communications Toolbox のマニュアルにあるとわかっているような場合では、プロダクトフィルタを使えば、この Toolbox のマニュアルのみ表示されるよう設定することができます。

Help Preferences の**プロダクトフィルタ (Product filter)** では、ヘルプブラウザで使用されるマニュアルが用意されている全ての製品についてチェックマークが付けられています。先ず、これらのマニュアルの中で、どのマニュアルを使用して

(チェックを入れる)、どのマニュアルを使わないのか (チェックをはずす) を選択してください。つぎに、ヘルプブラウザで指定した製品のマニュアルだけを使用するために、**プロダクトフィルタ (Product filter)** を **Selected** に設定します。また、全ての製品マニュアルをヘルプブラウザで使用したい時は、**プロダクトフィルタ (Product filter)** を **All** に設定します。また、ヘルプブラウザの **Select** ボタンを押せば、**ヘルプ設定プロダクトフィルタ (Help Preferences Product filter)** にアクセスすることができます。この点にも注意しておいてください。

PDF Reader - PDF reader の位置の設定

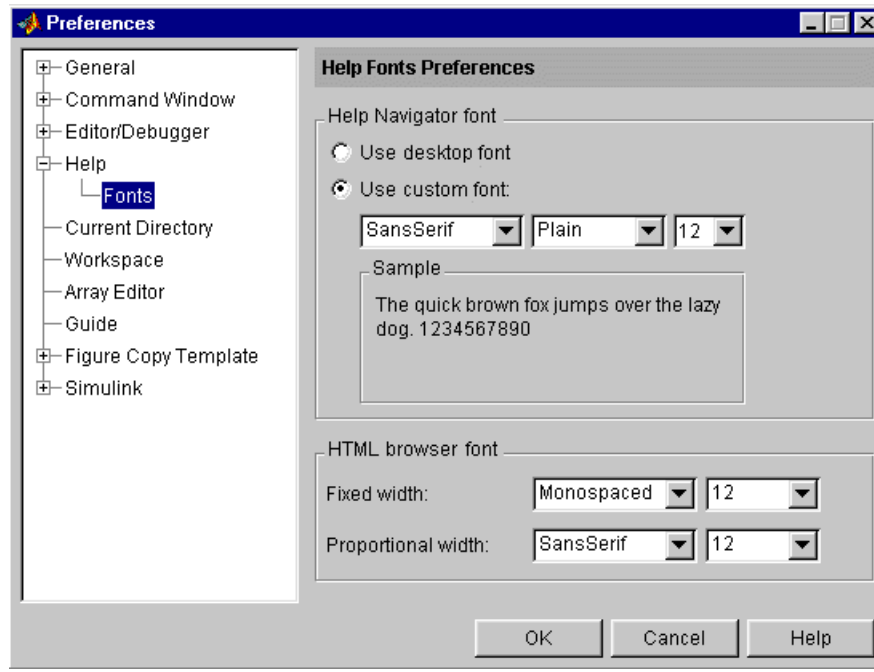
マニュアルの PDF バージョンにアクセスするには、ヘルプシステムに、ユーザの PDF reader(Adobe Acrobat) が、どの位置に配置されているのかを知らせておく必要があります。MATLAB インストール時に、MATLAB はユーザのシステムのどこに PDF reader があるのかを検索し、PDF reader がインストールされていた場合、そのロケーションを自動的に、**PDF reader** フィールドに設定します。MATLAB がユーザの PDF reader を発見できなかったり、ユーザがインストール後に PDF reader を移動してしまった場合は、**PDF reader** フィールドでロケーションの値を変更してください。... ボタンを使えば、ユーザのファイルシステムをブラウズできます。PDF reader の位置を選択する際に活用してください。

General - 目次表示ウィンドウと表示ページを同期させる

目次表示ウィンドウと表示ページとの同期を無効にするには、**ヘルプの設定 (Help Preferences)** の**一般設定 (General)** から**表示マニュアルと同期をもつ目次ツリーの維持 (Keep contents tree synchronized with displayed document)** のチェックを外してください。このアイテムにチェックを入れると、同期が有効となります。詳しくは、4-8 ページの " 目次表示ウィンドウとページ表示ウィンドウとの同期をとる " を参照してください。

ヘルプフォントの設定 - フォントの名前、スタイル、サイズを指定

この設定により、ヘルプナビゲータで使用するフォントの名前、スタイル、サイズを指定することができます。**ユーザ設定 (Preferences)** ダイアログボックスの左ウィンドウを拡大して、**フォント (Fonts)** アイテムを選択してください。つぎのように、**ヘルプフォントの設定 (Help Fonts Preferences)** アイテムが表示されます。



ヘルプナビゲーションフォント

ヘルプナビゲーションフォント (Help Navigator font) 設定を使って、ヘルプナビゲータで使用するフォントの特性を指定することができます。たとえば、スクロールせずにより多くの情報を参照できるように、ヘルプナビゲータのフォントをより小さなサイズのものに変更する、といった操作が可能です。

デスクトップフォントの使用 (Use desktop font) を選択すると、ヘルプナビゲータで使用するフォントは、**General - Font & Colors** で指定したフォントと同一のものに設定されます。**General - Font & Colors** で指定したフォントと異なるフォントをヘルプナビゲータで使用する場合には、**カスタムフォントの使用 (Use custom font)** を選択して、ヘルプナビゲータで使用するフォントの特性を指定してください。

- タイプ、たとえば、Sans Serif
- スタイル、たとえば、Bold
- ポインタ単位のサイズ、たとえば、12 ポイント

選択した後、**サンプル (Sample)** エリアで、フォントがどのように表示されるのか、確認することができます。

HTML ブラウザフォント

ヘルプ表示ウィンドウで使用するフォントのタイプやサイズを指定するには、**固定幅のフォント (Fixed width)** と**可変幅のフォント (Proportional width)** の設定を使用します。MathWorks のマニュアルでは、テキストのほとんどについて、可変幅のフォントを使用しています。固定幅のフォントは、コードの例題や関数名、システム I/O を表示するときに使用しています。固定幅のフォントの使用例については、つぎの例を参照してください。

```
t = 0:pi/20:2*pi;
y = exp(sin(t));
plotyy(t,y,t,y,'plot','stem')
```

コード、関数名、システム I/O をマニュアルに記載されているその他のテキストと容易に区別できるように、固定幅フォントには、可変幅のフォントとは異なるフォントを指定することをお勧めします。

マニュアルの印刷

ユーザは、ヘルプブラウザに表示されているカレントページを印刷したり、マニュアルの PDF バージョンからページ単位、マニュアル全体の印刷を行うことができます。

ヘルプブラウザからページの印刷

ヘルプブラウザで現在表示されているページを印刷するには、ヘルプ表示ウィンドウツールバーから印刷ボタン を選択するか、**ファイル (File)** メニューから**印刷 (Print)** を選択します。これにより、**印刷 (Print)** ダイアログボックスが表示されます。ダイアログボックスの入力を終え、OK を押すと、そのページを印刷します。

マニュアルの PDF バージョンの印刷

印刷するページ数が少ない場合、活字本と同様の品質を求めない場合では、MATLAB のヘルプブラウザから直接印刷することをお薦めします。ヘルプブラウザから直接印刷する方法については、4-16 ページの " ページの印刷 " を参照してください。

印刷したいページ数が多い場合、活字本同様の品質が求められる場合では、マニュアルの PDF バージョンを印刷してください。PDF マニュアルは、Adobe Acrobat Reader 等、ユーザの PDF reader で表示、印刷してください。PDF マニュアルにはフォント、グラフィック、フォーマット、イメージが完璧に埋め込まれており、印刷された本のように再現します。PDF ドキュメントでは、目次、インデックスからのリンクを利用して、関心のあるページに直接ジャンプすることができます。

マニュアルの PDF バージョンを印刷するには、つぎの操作をしてください。

1. Windows システムの場合では、つぎの操作を行ってください。先ず、MATLAB に付属するマニュアル CD をユーザの CD-ROM ドライバに挿入します。PDF ファイルは、CD の中にあり、ユーザのシステムにはインストールされません (UNIX システムでは、PDF ファイルはインストールされています)。ここで、何か問題が起きた場合は、ヘルプの設定をチェックしてください。詳細については、4-19 ページの " マニュアルの設置位置 -help ディレクトリの指定 " を参照してください。

2. ヘルプブラウザで、**目次 (Contents)** タブをクリックし、製品のタイトルを選択してください。

   選択した製品に関するロードマップページが開きます。

3 **マニュアルの印刷 (Printing the Documentation)**の下にリスト表示されたロードマップページの下部に、印刷へのリンクが張られているので、このリンクをクリックします。

対象製品に関するマニュアルのみ印刷する場合、Acrobat Reader を開き、目次とマニュアルの最初のページを表示してください。

対象製品について、複数のマニュアルを印刷する場合、選択肢をリストアップしたページが表示されます。この中から印刷したいアイテムを選択してください。Acrobat Reader がオープンし、マニュアルが表示されます。

問題が生じた場合は、ヘルプの設定をチェックしてください。- 4-20 ページの "PDF Reader - PDF reader の位置の設定 " を参照してください。

4 Acrobat の**ファイル (File)**メニューから**印刷 (Print)**を選択すると、マニュアルの印刷が始まります。

関数 help の使用

ヘルプブラウザとは異なる形式のヘルプ、または、ヘルプにアクセスするもう 1 つの方法、のいずれかを提供するヘルプ関数もいくつか用意されています。

関数	説明
doc	ヘルプブラウザ上で、指定した関数のリファレンスページを表示します。リファレンスページには、関数の使用法、詳細、例題、関連する関数へのリンクが記載されています。
docopt	ava GUIs をサポートしていない UNIX プラットフォーム上で、関数 docopt を使って、ヘルプファイルの設定位置を指定できます。
help	指定した関数の M- ファイルヘルプ (詳細とシンタックス) をコマンドウィンドウ上に表示します。
helpbrowser	MATLAB 上からマニュアルにアクセスする際に使用するヘルプブラウザを開きます。
helpdesk	ヘルプブラウザを開きます。古いリリースでは、関数 helpdesk にはヘルプデスク (ヘルプブラウザの原型) を呼び出す機能が割り当てられていました。今後のリリースで、この helpdesk 関数は削除されることになります。
helpwin	ヘルプブラウザに全関数のリストを表示し、それらの関数の M- ファイルヘルプにアクセスできるようにします。
lookfor	キーワードを指定すると、説明にそのキーワードが含まれている関数全てのリストと簡単な説明がヘルプブラウザに表示されます。
web	指定した Web ブラウザ (デフォルトでは MATLAB ヘルプブラウザが選択されています) に指定した URL のページを開きます。ユーザは、自分自身で作成した M- ファイル内に web 関数を取り込むことによって、マニュアルを表示させることができます。

関数リファレンスページを表示 - doc 関数

ヘルプブラウザ上で、指定した関数のリファレンスページを開くには、関数 doc を使用します。この関数は、ヘルプブラウザの検索機能 (**検索タイプ (Search type)** を**関数名 (Function name)** に設定した時の機能) と同等の機能を提供します。たとえば、つぎのようにタイプすると、

```
doc format
```

関数 format のリファレンスページが表示されます。

多重定義された関数

ある単一の関数名が複数の製品で使用されているとき、その関数を多重定義された関数と呼びます。関数 doc は、パス上で対象の名前をもつ最初に検出される関数に対するリファレンスページを表示し、コマンドウインドウ内に多重定義関数をリストします。多重定義された関数のヘルプを参照するには、リファレンスページを参照したい関数が置かれたディレクトリの名前を指定してください。たとえば、Database Toolbox の set 関数についてリファレンスページを参照したい場合には、つぎのようにタイプします。

```
doc database/set
```

コマンドウィンドウでヘルプの参照 - 関数 help

コマンドウィンドウで関数の簡単な説明や使用法をすぐに参照したい場合、関数 help を使用してください。たとえば、つぎのようにタイプすると、

```
help addpath
```

コマンドウィンドウに、関数 addpath の説明と使用法を表示します。これは、いわゆる、M- ファイルヘルプと言われるものです。関数 help が提供する以上の情報にアクセスしたい場合、関数 doc を使用してください。

注意 . コマンドウィンドウに表示される M- ファイルヘルプは、関数や変数名の表示に大文字を使用しています。これは、関数や変数名が、それ以外の文字列から容易に区別できるようにするためのものですが、関数名をタイプするときには、大文字は使用できません。必ず小文字を使用してください。MATLAB は、大文字と小文字の区別を感知し、関数名は実際には小文字で登録されているため、大文字を使用すると関数名として認識されません。

多重定義された関数

ある単一の関数名が、複数の製品で使用されているとき、その関数を多重定義された関数と呼びます。関数 help は、パス上で対象となる名前に最初に検出される関数に対する M- ファイルヘルプを表示し、最後に、多重定義関数をリストします。多重定義した関数のヘルプを参照するには、参照したい関数を含むディレクトリ名を、関数名の前に指定してください。たとえば、Database Toolbox の中の関数 set に対するヘルプを参照したい場合、つぎのようにします。

```
help database/set
```

ユーザ自身の M- ファイルの M- ファイルヘルプを作成

ユーザが作成した M- ファイルの M- ファイルヘルプを作成し、help コマンドでアクセスできるようにすることも可能です。この点の詳細については、関数 help のリファレンスページを参照してください。

ヘルプを参照するその他の方法

MathWorks 製品のヘルプを参照するためのヘルプブラウザとヘルプ関数を使用する以外の方法として、つぎの方法が用意されています。

- 4-28 ページの " 製品固有のヘルプ機能 "
- 4-28 ページの " デモの実行 "
- 4-29 ページの " テクニカルサポートへのコンタクト "
- 4-29 ページの " ヘルプに関する情報を MathWorks へフィードバック "
- 4-29 ページの " バージョン及びライセンス情報を参照 "
- 4-30 ページの " 他製品のマニュアルへのアクセス "
- 4-30 ページの "MathWorks 製品のニュースグループに参加 "

製品固有のヘルプ機能

ヘルプブラウザとヘルプ関数の他にも、製品やツールの中にはヘルプを参照するためのその他の手段を用意しているものもあります。製品を使用している場合、**ヘルプ (Help)** メニューに何らかの入力をする、ダイアログボックスに**ヘルプ (Help)** ボタンが表示される、コンテキストメニューから**ヘルプ (Help)** を選択することができる、といったことに遭遇するかもしれません。これらの方法では、コンテキストに応じた全てのヘルプをヘルプブラウザに表示します。一方、**F1** キーを押す等のヘルプを参照するその他の方法については、このような方法を使用する製品、または、ツールのマニュアルに記載されています。

デモの実行

多くの製品に、その主要な特徴を例示するためのデモンストレーションが用意されています。最初にその製品を使用するときに、デモンストレーションを実行するといろいろな示唆が与えられることもしばしばです。ある製品について利用可能なデモンストレーションのリストを参照し、デモンストレーションを実行するには、Launch Pad を使用するか、対象となる製品の **Help** メニューを活用してください。

テクニカルサポートへのコンタクト

ユーザのコンピュータがインターネットとつながっていれば、MathWorks テクニカルサポートとコンタクトをとって、製品の使用で遭遇した問題についてヘルプ情報を得ることも可能です。

- テクニカルサポート情報をヘルプブラウザの**検索 (Search)** 機能を使って検索するには、**検索タイプ (Search type)** を**オンラインヘルプ情報データベース (Online Knowledge Base)** に設定します。オンラインヘルプ情報データベースは、ユーザが提起する問題について最新のソリューションを提供します。
- **Web** メニューから **Technical Support Knowledge Base** を選択し、テクニカルサポートのホームページにジャンプしてください。このホームページでは、サードパーティーの本、Q & A、アドバイス、バグレポートなどを入手することができます。

ヘルプに関する情報を MathWorks へフィードバック

マニュアルやヘルプ機能に関する問題のレポートやコメント、提案を MathWorks にフィードバックするには、doc@mathworks.com に E メールを送ってください。

E メールを送る代わりに、ホームページ上のフォームからフィードバックを行うこともできます。このフォームにアクセスするには、ヘルプブラウザの**目次 (Contents)** ウィンドウを使用してください。最後の入力項目 **Support and Web Services** を開き、**ヘルプに関するフィードバック (Feedback on Help)** を選択すればフォームにアクセスできます。

バージョン及びライセンス情報を参照

製品バージョンまたはライセンス情報を参照するには、参照したい製品の **Help** メニューから **About** を選択します。すると **About** ダイアログボックス上にバージョン情報が表示されます。ライセンス情報を表示するには、ダイアログボックスの **Show License** をクリックします。**Help** メニューをもたない製品については、関数 ver を使ってバージョン情報を表示します。MATLAB のライセンスナンバーを確認するには、コマンドウィンドウ上で license とタイプしてください。

他製品のマニュアルへのアクセス

ヘルプブラウザを使って、ユーザのシステムにインストールされている全製品のマニュアルにアクセスすることができます。ユーザがもっていない MathWorks 製品のマニュアルを参照するには、つぎの操作をしてください。

- 全ての製品のオンラインマニュアルを MathWorks のホームページ http://www.mathworks.com から参照することができます。アクセスログインを使用してください。ユーザがアクセスメンバーになっていない時は、**Web** メニューから **Membership** を選択し、アドバイスに従って会員になるための操作を行ってください。
- MATLAB に付属するドキュメント CD からすべての製品のドキュメントにアクセスできます。これは、全製品に対する PDF ファイルを含みます。ヘルプブラウザの設定を使って、ドキュメントの位置を CD-ROM ドライブに設定し、プロダクトフィルタを使ってドキュメントを見たい製品を指定します。詳細は、4-18 ページの " ヘルプブラウザの設定 " を参照してください。

MathWorks 製品のニュースグループに参加

MATLAB 及び関連製品のニュースグループである USENET ニュースグループ comp.soft-sys.matlab は、世界中に散らばる数千人のユーザに読まれています。ニュースグループにアクセスすれば、質問をしたり、ヘルプまたはアドバイスを得るといったことはもちろんのこと、コードや例題を共有することも可能となります。テクニカルサポートホームページの comp.soft-sys.matlab リンクをたどれば、ポスティングされたファイルで構成される大きなアーカイブを参照、検索することもできます。**Web** メニューから **Technical Support Knowledge Base** を選択してください。

5

ワークスペース、パス、ファイル操作

MATLAB ワークスペース 5-3
ワークスペースブラウザ 5-3
配列エディタを使ってワークスペース変数を参照、エディット 5-10

サーチパス 5-14
サーチパスの機能 5-14
サーチパスの表示と設定 5-15

ファイルオペレーション 5-20
カレントディレクトリフィールド 5-20
カレントディレクトリブラウザ 5-20
ディレクトリの参照、変更 5-22
ディレクトリとファイルの作成、リネーム、コピー、削除 . 5-23
ファイルのオープン、実行、内容の参照 5-26
ファイル内で内容の検索、置換 5-28
カレントディレクトリブラウザの設定 5-30

MATLAB で作業するには、つぎの重要項目について理解する必要があります。

- ワークスペース – ワークスペースは、MATLAB セッション実行中にメモリに格納される変数群です。ワークスペースブラウザ、または、それと等価な機能をもつ関数を利用すると、ワークスペースの内容を参照することができます。
- サーチパス – MATLAB は、M- ファイルや他の MATLAB 関連ファイルを探索するためにサーチパスやカレントディレクトリを使用します。パスを参照、変更するには、**Set Path** ダイアログボックス、または、それと等価な機能をもつ関数を使用してください。
- ファイルオペレーション – MATLAB に関連するディレクトリやファイルの検索、参照、オープン、変更などを行うには、MATLAB カレントディレクトリブラウザ、または、それと等価な機能をもつ関数を使用してください。

MATLAB ワークスペース

MATLAB ワークスペースは、MATLAB セッション実行中に生成され、メモリに格納された (配列と名付けられた) 変数群で構成されています。ユーザは、関数を使ったり、M- ファイルを実行したり、セーブされたワークスペースをロードする、などの操作によって変数をワークスペースに追加しています。たとえば、つぎのように入力すると、

```
t = 0:pi/4:2*pi;
y = sin(t);
```

ワークスペースには、それぞれが9つの値をもつ2つの変数yとtが格納されます。

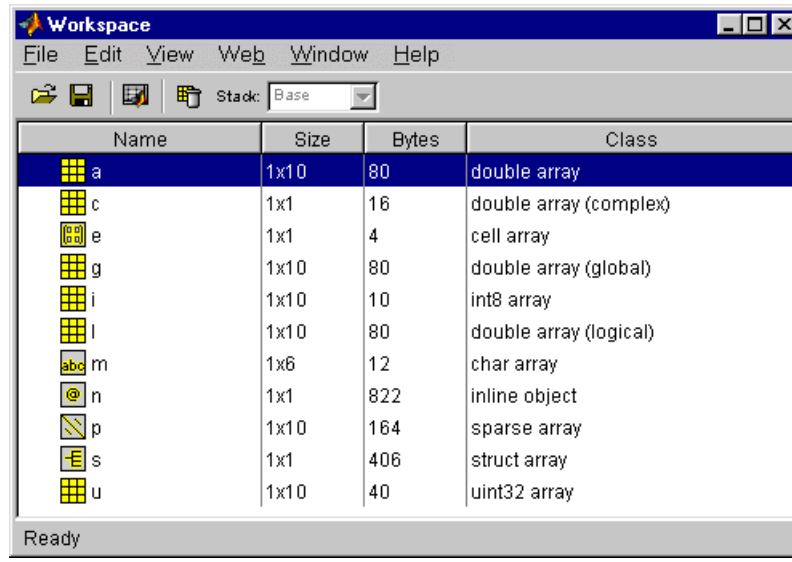
ワークスペースブラウザ

ワークスペースb ブラウザを使うと、MATLAB ワークスペース上で操作を行うことができるようになります。ワークスペースと等価な機能をもつ関数も利用でき、ワークスペースブラウザの各機能について説明する箇所で紹介されています。

ワークスペースブラウザをオープンするには、つぎの操作を行います。

- MATLAB デスクトップの**ビュー (View)** メニューから **Workspace** を選択します。
- MATLAB の下にある Launch Pad 上で、**Workspace** をダブルクリックします。
- コマンドウィンドウプロンプトで、workspace と入力します。

下のワークスペースブラウザが開きます。



ワークスペースブラウザ、または、それと等価な機能をもつ関数から実行できる操作は、つぎのとおりです。

- 5-4 ページの " カレントのワークスペースの参照 "
- 5-5 ページの " カレントのワークスペースのセーブ "
- 5-7 ページの " セーブされたワークスペースのロード "
- 5-8 ページの " ワークスペース変数の消去 "
- 5-8 ページの " スタックを使って、ベースワークスペースと関数ワークスペースの参照 "
- 5-9 ページの " ワークスペースブラウザからグラフィックスを作成 "
- 5-10 ページの " 配列エディタを使って、ワークスペース変数を参照、エディット "

また、ユーザは、5-9 ページの " ワークスペースブラウザの設定 " を指定できます。

カレントのワークスペースの参照

ワークスペースブラウザは、ワークスペースに格納された各変数の名前、配列のサイズ、バイトで表示されたサイズ、クラスを表示します。それぞれの変数に割

り当てられたアイコンが、その変数のタイプを示しています。

情報を示す列のサイズを変更するには、列の見出しの境界をドラッグします。任意の列を表示、または、非表示にする、ソートの順序を指定する、といった操作を行うには、**ビュー (View)** メニューから **Workspace View Options** を選択してください。

等価な機能をもつ関数 . 関数 who を使用すれば、カレントのワークスペース変数をリスト表示できます。また、関数 whos を使って、変数のサイズとクラスに関する情報を変数と共にリスト表示します。

```
who
   Your variables are:
   A          M          S          v

whos
   Name       Size             Bytes  Class
   A          4x4                128  double array
   M          8x1               2368  cell array
   S          1x1                398  struct array
   v          5x9                 90  char array
   Grand total is 286 elements using 2984 bytes
```

ワークスペース内に指定した変数が格納されているかどうかを確認するには、関数 exist を使用してください。

カレントのワークスペースのセーブ

ワークスペースは、複数の MATLAB セッション間で、その値が保持されることはありません。MATLAB を一度終了すれば、ワークスペースは消去されます。カレントワークスペースに格納された変数の全て、または、一部を MAT- ファイル (MATLAB 固有のバイナリファイル) にセーブしておくことは可能です。一度セーブしておけば、後々、カレントセッション、または、他のセッション中に MAT- ファイルをロードしてワークスペース変数を再び使用することができます。

すべての変数のセーブ . ワークスペースブラウザを使って、ワークスペース変数をセーブするには、

1 **ファイル (File)** またはコンテキストメニューから **Save Workspace As** を選択するか、ワークスペースブラウザツールバーのセーブボタン をクリックしてく

ださい。

この操作により、Save ダイアログボックスがオープンします。

2 セーブする位置と**ファイル名 (File name)** を指定します。MATLAB は、自動的に拡張子 .mat をファイル名に付加します。

3 Save をクリックします。

ワークスペース変数は、指定した MAT- ファイル名でセーブされています。

選択した変数のセーブ . ワークスペース変数の全てではなく、一部をセーブするには、つぎの操作を行います。

1 ワークスペースブラウザで変数を選択します。複数の変数を選択するには、Shift- クリック、または、Ctrl- クリックを行います。

2 右クリックおよびコンテキストメニューからSave Selection Asを選択します。

Save to MAT-File ダイアログボックスがオープンします。

3 セーブする位置と**ファイル名 (File name)** を指定します。MATLAB は、自動的にファイル名に拡張子 .mat を付加します。

4 Save をクリックします。

ワークスペース変数は、指定した MAT- ファイル名でセーブされます。

等価な機能をもつ関数 . ワークスペース変数をセーブするには、セーブしたいファイル名を後ろに付けて、save 関数を実行します。たとえば、

```
save('june10')
```

は、全てのカレントワークスペース変数を、ファイル june10.mat にセーブします。

ファイル名を指定しなかった場合、ワークスペースはカレント作業ディレクトリに、matlab.mat にセーブされます。ユーザは、セーブする変数を選び、セーブするデータのフォーマット、たとえば、ascii のように、設定することができます。これらの関数の使用法やその他の使用法については、関数 save のレファレンスページを参照してください。MATLAB は、情報をセーブする他の関数も用意しています。第 6 章 , " データのインポートとエクスポート " を参照してください。

セーブされたワークスペースのロード

以前セーブしたワークスペースをロードするには、つぎの操作を行います。:

1 ワークスペースブラウザのツールバーの " 開く " ボタン をクリックするか、ワークスペースブラウザを右クリックして、コンテキストメニューから **Import Data** を選択します。

   Open ダイアログボックスがオープンします。

2 ロードしたい MAT- ファイルを選択して**開く (Open)** をクリックします。

   この操作により、MAT- ファイルに保存された変数とそれらの値がワークスペースにロードされます。

ユーザは、インポートウイザードを使って、セーブされたワークスペースをロードすることもできます。**ファイル (File)** メニューから**データのインポート (Import Data)** を選択すると、インポートウイザードがオープンします。ロードウイザードを使って、セーブしているワークスペースをオープンします。詳細については、6-20 ページの " インポートウィザードを使ったバイナリデータファイルのインポート " を参照してください。

等価な機能をもつ関数 . セーブされたワークスペースを開くには、関数 load を使用します。たとえば、

```
load('june10')
```

とタイプすると、ファイル june10.mat に全てのワークスペース変数をロードします。

注意 セーブされた MAT- ファイル june10 が、変数 A, B, C から構成されているとします。この場合、june10 をロードすると、変数 A, B,C は、ワークスペースに戻されます。既にワークスペースに A,B, C という名の変数が存在していると、これらの変数は、june10からロードされた変数によって上書きされてしまいます。詳細については、関数 load のレファレンスページをを参照してください。また、MATLAB には、情報をロードする目的に使用できる他の関数も用意されています。第 6 章 , " データのインポートとエクスポート " を参照してください。

ワークスペース変数の消去

ユーザは、1つの変数を消去し、ワークスペースから削除することができます。

ワークスペースブラウザを使って、変数を消去するには、つぎの操作を行ってください。

1 ワークスペースブラウザで変数を選択してください。複数の変数を選択するには、**Shift**- クリック、または、**Ctrl**- クリックを行います。全ての変数を選択するには、**編集 (Edit)** メニュー、または、コンテキストメニューから**すべてを選択 (Select All)** を選択します。

2 選択した変数を消去するには、つぎのいずれかの操作を行ってください。

- **削除 (Delete)** キーを押す。
- **編集 (Edit)** メニューから**削除 (Delete)** を選択する。
- ツールバーの削除ボタン をクリックする。
- 右クリックを行い、コンテキストメニューから**Delete Selection**を選択する。

3 削除を確認するダイアログボックスが表示されます。このダイアログボックスの **Yes** をクリックすると変数が消去されます。

確認ダイアログボックスの表示 / 非表示は、ユーザが自由に設定できます。その設定を変更するには、5-9 ページの " ワークスペースブラウザの設定 " を参照してください。

全ての変数を一度に削除するには、ワークスペースブラウザのコンテキストメニュー、または、**編集 (Edit)** メニューから、**Clear Workspace** を選択します。

等価な機能をもつ関数 . 関数 clear を使用すれば、変数を消去できます。たとえば、

```
clear A M
```

は、ワークスペースから変数 A と M を消去します。

スタックを使って、ベースワークスペースと関数ワークスペースの参照

M- ファイルを実行している場合、MATLAB は、実行している M- ファイル関数をそれぞれ関数独自のワークスペースに割り当てます。この関数独自のワークスペースを関数ワークスペースと呼びます。関数ワークスペースは、MATLAB のベースワークスペースとは切り離された形で存在しています。ワークスペースブラウザの **Stack** フィールドを使うと、M- ファイルのデバッギング中にベースワークスペース、関数ワークスペースにアクセスすることができます。**Stack** フィールド

は、デバッグモードでのみ使用できます。他のモードでは、このフィールドは、グレーで塗りつぶされて表示されます。**Stack** フィールドは、エディタ / デバッガからもアクセスできます。詳細は、7-16 ページの "M- ファイルのデバッグ " を参照してください。

ワークスペースブラウザからグラフィックスを作成

ワークスペースブラウザから、変数のグラフを作成することができます。グラフ化したい変数の上で右クリックし、表示されたコンテキストメニューから **Graph Selection** を選択します。そして、作成したいグラフのタイプを選択すれば、フィギュアウィンドウにグラフが表示されます。MATLAB グラフィックスドキュメントを参照してください。

ワークスペースブラウザの設定

ワークスペースブラウザで使用するフォントの設定、ワークスペースブラウザで変数を消去した場合に現れる確認ダイアログボックスの表示 / 非表示設定を指定することができます。

ワークスペースブラウザの**ファイル (File)** メニューから**設定 (Preferences)** を選択します。**設定 (Preferences)** ダイアログボックスが、**Workspace Preferences** パネルにオープンします。

フォント . ワークスペースブラウザのフォント設定では、ワークスペースブラウザで使用するフォントの特性を指定できます。**General Font & Colors** 設定に対するものと同じものを、ワークスペースブラウザの中のフォントとして使用するときは、**Use desktop font** を選択します。

デスクトップと異なるフォントをワークスペースブラウザで使用したいときは、**Use custom font** を選択し、ワークスペースブラウザで使用するフォントの特性を指定してください。

- タイプ 、例： Sans Serif
- スタイル、例： Bold
- ポイント単位のサイズ、例：12 ポイント

選択を行った後に、**Sample** エリアで、フォントがどのように表示されるかを確認することができます。

変数の削除を確認 . 変数の削除を確認 (Confirm deletion of variables) チェックボックスをチェックすれば、変数を削除する時に確認ダイアログボックスが表示されます。

配列エディタを使って、ワークスペース変数を参照、エディット

配列エディタを使って、一次元または二次元の数値配列、文字列、文字列のセル配列を視覚的に参照、エディットすることができます。配列エディタの機能は、つぎのとおりです。

- 5-10 ページの " 配列エディタのオープン "
- 5-12 ページの " 配列エディタで要素の値の変更 "
- 5-12 ページの " 配列エディタでの数値表示を制御 "

さらに、ユーザは、配列エディタの設定を指定できます。5-12 ページの " 配列エディタの設定 " を参照してください。

配列エディタのオープン

ユーザは、ワークスペースブラウザから配列エディタをオープンすることができます。

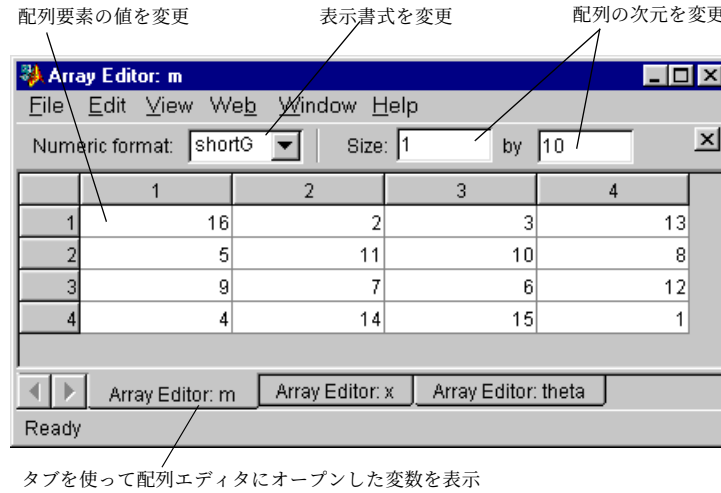
1 ワークスペースブラウザで、開きたい変数を選択します。複数の変数を選択するには、Shift- クリック、または、Ctrl- クリックを使用してください。

2 ツールバーの " 選択対象を開く " ボタン をクリックするか、右クリックで表示されたコンテキストメニューから Open Selection を選択してください。

また、1 つの変数を開く場合には、開きたい変数をダブルクリックすることによっても、その変数を開くことができます。

配列エディタが開き、選択した変数の値を表示します。

配列エディタが開き、選択した変数の値を表示します。



配列エディタに、さらにその他の変数を開くには、このステップを繰り返してください。配列エディタでは、ウィンドウの下部に表示されるタブ、または、**Window** メニューを使うと各変数の内容にアクセスできます。

等価な機能をもつ関数． ワークスペースに格納された変数の内容を参照するには、コマンドウィンドウプロンプトで、その変数の内容を入力するだけで参照できます。たとえば

```
m
```

とタイプすると、MATLAB は、つぎの結果を出力します。

```
m =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

配列エディタで変数を開くには、関数 openvar を開きたい変数名を引数として設定して実行します。たとえば、つぎのように入力すると、

```
openvar('m')
```

MATLAB は変数 m を配列エディタに開きます。

配列エディタで要素の値の変更

配列エディタで変更したい値を含むセルをクリックし、新しい値を入力した後、**Enter** または **Return** キーを押します。または、Enter、または、Return キーを押す代わりに、他のセルをクリックすると値の変更が有効となります。

配列の次元を変更するには、**Size** フィールドに行と列に対する新しい値を入力します。サイズが増加する場合は、新規の行および列は、末尾に追加されてゼロが入力されます。サイズが減少する場合は、データは失われます。MATLAB は末尾から行および列を削除します。

既存の MAT- ファイルを開き、配列エディタで変更を行った場合、変更をセーブするために、その MAT- ファイルをセーブする必要があります。詳細については、5-5 ページの " カレントのワークスペースのセーブ " を参照してください。

配列エディタでの数値表示を制御

配列エディタ上で、数値をどのように表示するのかを制御するには、**Numeric format** リストボックスの要素を選択します。選択できるフォーマットの詳細については、関数 format のレファレンスページを参照してください。ここで、設定したフォーマットは、配列エディタでのカレントセッションの対象変数の表示にのみ適用され、MATLAB の数値計算、MATLAB における数値のセーブに影響を与えることはありません。また、コマンドウィンドウの数値表示のフォーマットも変更されません。

配列エディタ上の全ての変数に対して、ある特定のフォーマットを適用し、複数のセッションに渡って、そのフォーマットを用いるには、次節で説明する設定を使って、配列エデッタにフォーマットを設定してください。

配列エディタの設定

配列エディタの設定を使って、数値表示のフォーマット、使用されるフォントのタイプ、スタイル、サイズを指定することができます。

配列エディタの設定を行うには、**ファイル (File)** メニューから **設定 (Preferences)** を選択してください。**設定 (Preferences)** ダイアログボックスがオープンし、**Array Editor Preferences** が表示されます。

フォント (Font) と **デフォルトフォーマット (Default format)** を指定します。

フォント．配列エディタフォント設定は、配列エディタ上で使用されるフォントの特性を指定します。**General Font & Colors** 設定に対するものと同じものを、ワークスペースブラウザの中のフォントとして使用するときは、**Use desktop font** を選択します。デスクトップと異なるフォントをワークスペースブラウザで使用したいときは、**Use custom font** を選択し、ワークスペースブラウザで使用するフォントの特性を指定してください。

- タイプ、例 Sans Serif
- スタイル、例 Bold
- ポイント単位でのポイント、例：12 ポイント

選択を行った後に、**Sample** エリアで、フォントがどのように表示されるかを確認することができます。

デフォルトフォーマット．配列エディタに表示される数値の出力フォーマットを指定します。ここで、設定したフォーマットは、数値をいかに表示するかにのみ影響を与えるもので、MATLAB が数値をどのように計算、セーブするかという点にまで影響を与えるものではありません。詳細については、5-12 ページの " 配列エディタでの数値表示を制御 " 、または、関数 format のレファレンスページを参照してください。

サーチパス

MATLAB は、*サーチパス*を使って、ユーザファイルシステムの複数のディレクトリに格納された M- ファイル、及び、他の MATLAB 関連ファイルを検索しています。これらのファイルとディレクトリは、MATLAB 及びツールボックスによって提供されたものです。MATLAB 上で実行したいファイルは、全てカレントディレクトリ、または、サーチパス上にあるディレクトリに格納されていなければなりません。デフォルトにより、MATLAB と MathWorks ツールボックスが提供するファイルはは、デフォルトでサーチパスに追加されています。

MATLAB 関連ファイルを作成する場合、作成したファイルを格納するディレクトリを MATLAB のサーチパスに追加しなくてはなりません。ユーザが独自の M- ファイルを作成したり、MATLAB によって提供された M- ファイルを修正したりした場合、これらのファイルを$matlabroot/toolbox/matlab上にないディレクトリに保存してください。ユーザが作成したファイルが $matlabroot/toolbox/matlab 上に保存されていると、MATLAB の新しいバージョンをインストールした時に上書きされてしまいます。新しいバージョンをインストールした場合、新しい、または、更新されたファイルを使用する前に MATLAB を再起動するか、関数 rehash を実行してください。この操作は、新しいファイル、更新されたファイルの位置を、MATLAB セッションの起動時にメモリにロードし、格納することによって性能を改善するために必要な操作です。しかし、ファイルの位置の変更が、常に自動的にキャッシュに反映されるわけではありません。

サーチパスの参照、修正についての詳細は、5-15 ページの " サーチパスの表示と設定 " を参照してください。

サーチパスの機能

サーチパスは、*MATLAB パス*とも呼ばれます。このパスに記載されているファイルは、パス上にあるものと考えます。サーチパスにあるディレクトリを追加した場合、ユーザはディレクトリをパスに追加したことになります。サブディレクトリは、明示的にパスに追加しなくてはなりません。それらは、親ディレクトリが存在しているので、パス上にない可能性もあります。サーチパスは、ファイル pathdef.m に格納されています。

パス上にディレクトリの順序も無関係ではありません。たとえば、MATLAB が、つぎに示すような foo という名前の要素を検索するとしましょう。MATLAB プロンプトで、foo と入力すると、MATLAB は、つぎの操作を行います。

1. foo を変数として探索します。
2. foo を組み込み関数として探索します。

3 foo.m という名のファイルをカレントディレクトリで検索します。

4 MATLAB サーチパス上のディレクトリで foo.m を検索します。

プライベート関数、サブ関数、オブジェクト指向の関数の探索範囲を制限するため、実際の検索ルールは、より複雑なものとなります。しかし、ユーザが作業する通常の M- ファイルについては、この単純化された検索ルールに従って検索が行われます。

同一名称の関数が 1 つ以上存在する場合、サーチパス上に並ぶディレクトリの順序が重要なものとなってきます。MATLAB は、そのような関数を探すときに、サーチパス上の順序でディレクトリを検索し、最初に発見した関数を抽出するからです。この場合、サーチパス上の順序でより後に追加されたディレクトリに格納された同一名称の関数は、実体のないものとみなされ、実行できなくなります。詳細については、MATLAB が実行する方法をどのように決定するかを参照してください。

使用されているパスネームを参照するには、特定の関数に対して、which を使用します。詳細については、関数 which のレファレンスページを参照してください。

サーチパスの表示と設定

パス設定 (Set Path) ダイアログボックスを使って、MATLAB のサーチパスを参照、修正したり、パス上にあるディレクトリに格納されたファイルを参照したりすることができます。このダイアログボックスと等価な機能をもつ関数については、**パス設定 (Set Path)** ダイアログボックスの各機能について解説している節を参照してください。

ファイル (File) メニューから、**パスの設定 (Set Path)** を選択するか、または、コマンドウィンドウプロンプトで、pathtool とタイプすると、つぎの**パスの設定 (Set Path)** ダイアログボックスが開きます。

パスの設定 (Set Path) ダイアログボックスでは、つぎの機能を実行できます。

- 5-17 ページの " サーチパスの参照 "
- 5-17 ページの " ディレクトリをサーチパスに追加 "
- 5-18 ページの " サーチパス内でのディレクトリの移動 "
- 5-18 ページの " サーチパスからディレクトリの削除 "
- 5-18 ページの " サーチパスをデフォルトに戻す "
- 5-18 ページの " セーブされたパスに戻す "
- 5-19 ページの " パス設定の保存 "

サーチパスの参照

パスの設定 (Set Path) ダイアログボックスの **MATLAB search path** に、サーチパス上にある全てのディレクトリがリスト表示されます。

等価な機能をもつ関数 . サーチパスを参照するには、関数 path を実行してください。

ディレクトリをサーチパスに追加

パスの設定 (Set Path) ダイアログボックスを使って、ディレクトリを MATLAB サーチパスに追加することができます。

1 **フォルダの付加 (Add Folder)** または **サブフォルダの付加 (Add with Subfolders)** ボタンをクリックします。

- 選択したディレクトリのみ追加して、サブディレクトリを追加したくない場合、**フォルダを付加 (Add Folder)** をクリックします。
- 選択したディレクトリのみならず、そのサブディレクトリも全て追加したい場合、**サブフォルダの付加 (Add with Subfolders)** ボタンをクリックします。

フォルダを探す (Browse for Folder) ダイアログボックスが開きます。

2 **フォルダを探す (Browse for Folder)** ダイアログボックスで、ユーザのファイルシステム表示を使って、追加したいディレクトリを選択し、**OK** をクリックします。

以上のように、選択したディレクトリや (サブディレクトリが指定された場合) 選択したサブディレクトリが、サーチパスのトップに追加されます。追加したディレクトリは、カレントの MATLAB セッションが終了するまでの間、サーチパス上に置かれることになります。次回以降のセッションに、サーチパスの修正を反映させるには、パスをセーブする必要があります。この点については、5-19 ページの " パス設定の保存 " を参照してください。

メソッドディレクトリ (@ で始まるディレクトリ)、プライベートディレクトリをサーチパスに追加することはできません。この点には注意してください。

等価な機能をもつ関数 . サーチパスにディレクトリを追加するには、関数 addpath を使用します。関数 addpath は、文字列でパスを取得したり、文字列を連結して新しいパスを形成する、といったオプション機能を提供します。

ユーザは、自身の startup M- ファイルに関数 addpath を設定し、MATLAB 起動時に自動的にパスが修正されるように設定することもできます。

サーチパス内でのディレクトリの移動

サーチパス上でのファイルの順序は、無関係ではありません。詳細については、5-14 ページの " サーチパスの機能 " を参照してください。

サーチパス上でのディレクトリの順序を変更するには、最初に移動したいディレクトリを選択します。つぎに **Move to Top** のような**移動 (Move)** ボタンを選択します。リストの先頭が、サーチパスのトップに対応し、リストで一番下に表示される項目が、サーチパスの最後に対応しています。

サーチパス上での新しいファイルの並び順は、カレント MATLAB セッションを終了させるまで有効です。次回以降のセッションで、新しいファイルの並び順を有効にするには、パスの設定をセーブする必要があります。詳細は、5-19 ページの " パス設定の保存 " を参照してください。

サーチパスからディレクトリの削除

パス設定 (Set Path) ダイアログボックスを使って、MATLAB サーチパスからディレクトリを削除することができます。

1 削除するディレクトリを選択します。

2 **削除 (Remove)** をクリックしてください。

この操作により、カレント MATLAB セッションが終了するまでの間、サーチパスからディレクトリが削除されます。次回以降のセッションでもディレクトリを削除したサーチパスを有効にするには、パスをセーブする必要があります。5-19 ページの " パス設定の保存 " を参照してください。

等価な機能をもつ関数 . サーチパスからディレクトリを削除するには、関数 rmpath を使用します。

ユーザが、MATLAB を起動する場合、自動的にパスを変更する startup M- ファイルを関数 rmpath することが設定できます。

サーチパスをデフォルトに戻す

サーチパスをデフォルトに戻すには、**パス設定 (Set Path)** ダイアログボックスの **デフォルト (Default)** をクリックしてください。これにより、サーチパスは出荷時の設定を使うように修正されます。

セーブされたパスに戻す

セーブされたパスに戻すには、**パス設定 (Set Path)** ダイアログボックスの**戻す**

(Revert) をクリックします。この操作により、以前セーブしたサーチパスに戻ります。カレントセッションの実行中に、**パス設定 (Set Path)** ダイアログボックスでパスを変更し、セーブせずに終了してしまった場合、**戻す (Revert)** をクリックすると、パス設定をリストアすることができます。

パス設定の保存

サーチパスを変更しても、その変更内容はカレント MATLAB セッションを終了する時までしか有効とはなりません。次回以降のセッションでも変更内容を有効にするには、変更内容をセーブする必要があります。**パス設定 (Set Path)** ダイアログボックスを使って、変更内容をセーブするには、**保存 (Save)** をクリックしてください。

サーチパスは、ファイル pathdef.m に設定されています。デフォルトでは、ファイル pathdef.m は、$matlabroot\toolbox\local に格納されています。Windows プラットフォーム上で、ユーザが自身の MATLAB 起動ディレクトリにファイル pathdef.m を格納する場合、異なる pathdef.m を使用することができます。1-3 ページの "MATLAB の起動ディレクトリ " を参照してください。

ファイル pathdef.m をテキストエディタで直接エディットしてパスの設定を変更することも可能です。

UNIX ワークステーションでは、ユーザはファイル pathdef.m のエディットを許可するファイルシステムパーミッションを得ていない場合もあります。そのような場合では、path と addpath 関数をユーザの startup M- ファイルに設定すれば、デフォルトのパスを変更することができます。

ファイルオペレーション

MATLAB ファイルオペレーションは、カレントディレクトリを基準点として使用します。実行するファイルは、全てカレントディレクトリ、または、サーチパス上になくてはなりません。さらに、MATLAB でファイルを開いた場合、ファイルオープンダイアログボックスの出発点もカレントディレクトリとなります。カレントディレクトリを最も速く参照、変更するには、デスクトップツールバーに表示されるフィールド (下図) を使用します。

- 5-20 ページの " カレントディレクトリフィールド "
- 5-20 ページの " カレントディレクトリブラウザ "

カレントディレクトリフィールド

カレントディレクトリを表示または変更するための簡単な方法は、デスクトップウールバーの **Current Directory** フィールドを使うことです。

このフィールドからカレントディレクトリを変更するには、つぎのいずれかを行います。

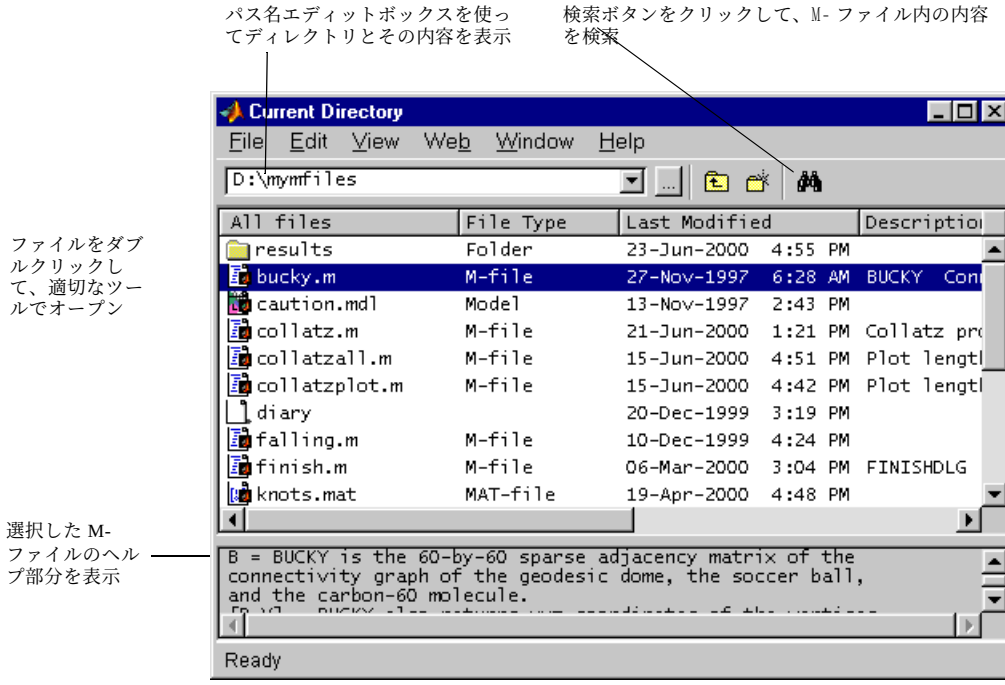
- フィールドで、新規のカレントディレクトリのパスを入力します。
- 下矢印をクリックして、以前の作業ディレクトリのリストを表示し、リストから項目を選択すると、ディレクトリは MATLAB のカレントディレクトリになります。ディレクトリは、最近使用した順に上から並べられた表示されます。リストを消去したり、リストに記載されるディレクトリの数を設定することもできます。5-30 ページの " カレントディレクトリブラウザの設定 " を参照してください。
- ブラウズボタン (...) をクリックして新規のカレントディレクトリを設定します。

カレントディレクトリブラウザ

MATLAB に関連するディレクトリやファイルの検索、表示、オープン、変更を行うには、MATLAB カレントディレクトリブラウザを使います。等価な機能をもつ関数は、カレントディレクトリブラウザの各機能に対して記述されています。

カレントディレクトリブラウザをオープンするには、**ビュー (View)** メニューから **カレントディレクトリ (Current Directory)** を選択するか、コマンドウィンドウプ

ロンプトで、filebrowser と入力します。Launch Pad からカレントディレクトリブラウザを開くこともできます。これらの操作により、つぎのカレントディレクトリブラウザが開きます。



カレントディレクトリブラウザを使って行うことができる主要なファイルオペレーションは、つぎの事柄です。

- 5-22 ページの " ディレクトリの参照、変更 "
- 5-23 ページの " ディレクトリとファイルの作成、リネーム、コピー、削除 "
- 5-26 ページの " ファイルのオープン、実行、内容の参照 "
- 5-28 ページの " ファイル内の内容の検索と置換 "

ユーザは、カレントディレクトリブラウザの設定を変えることもできます。5-30 ページの " カレントディレクトリブラウザの設定 " を参照してください。

ディレクトリの参照、変更

ディレクトリの参照、変更に関連するカレントディレクトリの機能は、つぎの事柄です。

- 5-22 ページの " カレント作業ディレクトリを変更し、その内容の参照 "
- 5-23 ページの " サーチパスにディレクトリを追加 "
- 5-23 ページの " 表示の変更 "

カレント作業ディレクトリを変更し、その内容の参照

カレントディレクトリを変更するには、カレントディレクトリブラウザの Pathname edit ボックスにディレクトリ名を入力し、**入力 (Enter)** または **Return** キーを押します。これにより入力したディレクトリがカレント作業ディレクトリとなり、そのディレクトリに格納されているファイルやサブディレクトリがリスト表示されます。

最近表示されたディレクトリを参照するには、カレントディレクトリブラウザの Pathname edit ボックスの右端に表示される下矢印をクリックします。以前表示されたディレクトリが最近使った順に上から並べた形でリスト表示されます。このリストから、内容を参照したいディレクトリを選択してください。なお、このリストの表示内容をクリアしたり、リストにセーブされるディレクトリの数を指定したりすることも可能です。詳細については、5-30 ページの " カレントディレクトリブラウザの設定 " を参照してください。

表示されたディレクトリに格納されたサブディレクトリの内容を表示するには、カレントディレクトリブラウザでサブディレクトリをダブルクリックするか、サブディレクトリを選択した状態で、**Enter** または **Return** キーのいずれかを押してください。サブディレクトリを右クリックし、コンテキストメニューから**開く (Open)** を選択することによって開くことも可能です。

ディレクトリ構造で、1 段階上に移動するには、カレントディレクトリブラウザツールバーのアップボタン をクリックするか、**バックスペース (Back Space)** キーを押してください。

等価な機能をもつ関数 . カレント作業ディレクトリ、または他の指定したディレクトリの内容を参照するには、関数 dir を使用します。

関数 what を使って、ディレクトリに格納された MATLAB 関連ファイルのみ参照することもできます。引数を設定しないで、what を実行すると、カレント作業ディレクトリに格納された MATLAB 関連ファイルが表示されます。また、指定した関数のパスネームを表示するには、関数 which を使用してください。

サーチパスにディレクトリを追加

カレントディレクトリブラウザから、MATLAB サーチパスにディレクトリを追加することができます。右クリックで表示されるコンテキストメニューから**パスへの追加 (Add to Path)** を選択してください。つぎのオプションの中から 1 つ選択することができます。

- Current Directory – カレントディレクトリをパスに追加します。
- Selected Folders – カレントディレクトリブラウザで選択したディレクトリをパスに追加します。
- Selected Folder and Subfolders – カレントディレクトリブラウザで選択したディレクトリをパスに追加し、全てのサブディレクトリもパスに追加します。

表示の変更

カレントディレクトリブラウザに表示されるファイルのタイプを指定するには、View -> Current Directory Filter を使用します。たとえば、M- ファイルのみ参照するといった作業が可能となります。

ユーザは、カレントディレクトリブラウザに表示される情報を列毎にソートすることもできます。ソートしたい列のタイトルをクリックしてください。クリックした列の情報を昇順でソートして表示します。もう、1 度列のタイトルをクリックすると、表示されている情報が降順でソートされます。

ユーザが自分のファイルシステムでカレントディレクトリを変更すると、カレントディレクトリブラウザは、その変更内容をすぐに反映することができません。コンテキストメニューから**リフレッシュ (Refresh)** を選択して、カレントディレクトリブラウザの表示を更新してください。

ディレクトリとファイルの作成、リネーム、コピー、削除

書き込み許可を得ていれば、カレントディレクトリブラウザに表示されているディレクトリについて、MATLAB 関連ファイルや MATLAB 関連ディレクトリの作成、コピー、削除、リネームを行うことができます。書き込み許可を得ていない場合でも、ファイルやディレクトリを他のディレクトリにコピーすることは可能です。

新規ファイルの作成

カレントディレクトリに新規ファイルを作成するには、つぎの操作を行ってください。

1 コンテキストメニューから**新規 (New)** を選択するか、**File** メニューを選択してから作成するファイルのタイプを選択します。

すると、デフォルトのファイル名 Untitled が付いた選択したファイルタイプのアイコン (例：M- ファイルアイコン) が、カレントディレクトリブラウザのリストの最後に表示されます。

2 新規ファイルに付けたい名前を、Untitled に上書きします。

3 **Enter** または **Return** キーを押します。

この操作により、ファイルが追加されます。

4 作成した新規 M- ファイルの内容を入力するには、このファイルをオープンします。5-26 ページの " ファイルのオープン、実行、内容の参照 " を参照してください。

新規ディレクトリの作成

カレントディレクトリに新規ディレクトリを作成するには、つぎの操作を行います。

1 カレントディレクトリツールバーの新規ディレクトリボタンをクリックするか、コンテキストメニューから **New -> Folder** を選択します。

デフォルト名 NewDirectory が付けられたアイコンが、カレントディレクトリブラウザのリストの最後に表示されます。

2 新規ディレクトリに付けたい名前を NewDirectory に上書きします。

3 **Enter** または **Return** キーを押します。

以上の操作により新規ディレクトリが追加されます。

ファイルやディレクトリのリネーム

ファイルやディレクトリをリネームするには、リネームしたいアイテムを選択、右クリックで表示されるコンテキストメニューから**リネーム (Rename)** を選択します。そして、ファイル、または、ディレクトリの既存名に新しい名前を上書きし、**Enter** または **Return** キーを押してください。ファイルまたはディレクトリはリネームされています。

ファイルとディレクトリの切り取り、または削除

ファイルとディレクトリを切り取る、または削除するには、つぎの操作を行ってください。

1 ファイル、または、ディレクトリを選択します。複数のファイル、ディレクトリを選択するには、**Shift**-クリック、または、**Ctrl**-クリックを押してください。

2 右クリックで表示されるコンテキストメニューから、**Cut** または **Delete** を選択してください。**Cut** については、**編集 (Edit)** メニューから選択することもできます。

   ディレクトリを削除するには、先ず、ディレクトリの内容を先に削除しておかなければなりません。この点には注意してください。

以上の操作により、ファイル、または、ディレクトリが削除されます。

等価な機能をもつ関数 . ファイルを削除するには delete 関数を使用します。たとえば、

```
delete('d:\mymfiles\testfun.m')
```

と入力すると、ファイル testfun.m が削除されます。

ファイルのコピー、ペース

ディレクトリをコピー、ペーストすることはできませんが、ファイルをコピー、ペーストすることは可能です。ファイルをコピー、ペーストするには、つぎの操作を行います。

1 コピーしたいファイルを選択します。複数のファイルを選択するには、**Shift**-クリック、または、**Ctrl**-クリックを押してください。

2 右クリックで表示されるコンテキストメニューから**コピー (Copy)** を選択します。または、**編集 (Edit)** メニューから**コピー (Copy)** を選択します。

3 先に切り取り、または、コピーしたファイルをコピーしたいディレクトリに移動します。

4 ファイルを右クリックし、コンテキストメニューから**貼り付け (Paste)** を選択するか、**編集 (Edit)** メニューから**貼り付け (Paste)** を選択してください。

ファイルのオープン、実行、内容の参照

ファイルのオープン

カレントディレクトリブラウザのファイルオープン機能を使って、ファイルをオープンすることができます。オープンするファイルのタイプに合ったツールをオープンします。

ファイルをオープンするには、1 つ以上のファイルを選択し、つぎの操作のいずれか 1 つを実行してください。

- Enter または Return キーを押す。
- 右クリックし、コンテキストメニューから**開く (Open)** を選択する。ファイルをダブルクリックする。

この操作により、ファイルは適切なツールをオープンします。たとえば、M- ファイルをオープンしたときは、エディタ / デバッガをオープンし、モデルファイル (mdl) をオープンしたときは、Simulink が開きます。

エディタにタイプに関わらずファイルをオープンするには、コンテキストメニューから**テキストとして開く (Open as Text)** を選択します。

ファイルからデータをインポートすることもできます。ファイルを選択し、右クリックで表示されたコンテキストメニューから**データのインポート (Import Data)** を選択してください。インポートウィザードが開きます。データのインポート方法は、第 6 章 ," データのインポートとエクスポート " を参照してください。

等価な機能をもつ関数 . 関数 open を使って、拡張子が与えられた状態で、ファイルをそのファイルに合ったツール上で開くことができます。この関数は、標準 MATLAB ファイルタイプについてはデフォルトの挙動を示します。他のファイルタイプにインターフェースを拡張したり、標準ファイルに対するデフォルトの挙動を無効にする、といった操作を行うことも可能です。

ファイルタイプ	拡張子	動作
フィギュアファイル	fig	フィギュアをフィギュアウィンドウにオープンします。.
HTML ファイル	html	HTML ファイル name をヘルプブラウザにオープンします。
M- ファイル	m	M- ファイル name をエディタにオープンします。

ファイルタイプ	拡張子	動作 （続き）
MAT- ファイル	mat	MAT- ファイル name をロードウィザードにオープンします。
モデル	mdl	モデル name を Simulink にオープンします。
P- ファイル	p	対応する M- ファイル name.m が存在していれば、エディタでオープンします。
変数	拡張子なし	数値配列、文字列配列 name を配列エディタでオープンします。この場合、関数 open は、関数 openvar を実行します。
その他	カスタム	補助関数 opencustom を呼び出して、ファイル name.custom をオープンします。ここで、opencustom は、ユーザ定義関数を示しています。

ASCII ファイル（M- ファイル等）の内容を参照するには、関数 type を使用します。たとえば、

```
type('startup')
```

は、コマンドウィンドウにファイル startup.m の内容を表示します。

M- ファイルの実行

カレントディレクトリブラウザから M- ファイルを実行するには、実行したい M- ファイルを選択し、右クリックで表示されるコンテキストメニューから**実行 (Run)** を実行してください。実行結果がコマンドウィンドウ上に表示されます。

M- ファイルのヘルプの参照

カレントディレクトリブラウザで選択したM-ファイルのヘルプを参照することもできます。コンテキストメニューから**ヘルプのビュー (View Help)** を選択してください。ヘルプブラウザに選択した関数のレファレンスページが表示されます。レファレンスページが存在しない場合、M- ファイルヘルプがその代わりに表示されます。

カレントディレクトリブラウザで M- ファイルヘルプを参照することも可能です。5-30 ページの " カレントディレクトリブラウザの設定 " を参照してください。

ファイル内の内容の検索と置換

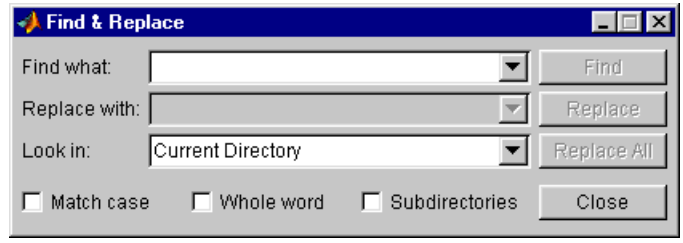
カレントディレクトリブラウザから、ファイル内で指定した文字列の検索を行うことができます。エディタでファイルをオープンしていれば、ファイル内の指定した文字列を置換することも可能です

ファイル内で指定文字列の検索

ファイル内で指定した文字列の検索を行うには、つぎの操作を行います。

1 カレントディレクトリブラウザツールバーの検索ボタン をクリックします。

Find & Replace ダイアログボックスが表示されます。このダイアログボックスは、エディタ / デバッガからアクセスできる Find & Replace ダイアログボックスと同一の機能を提供します。

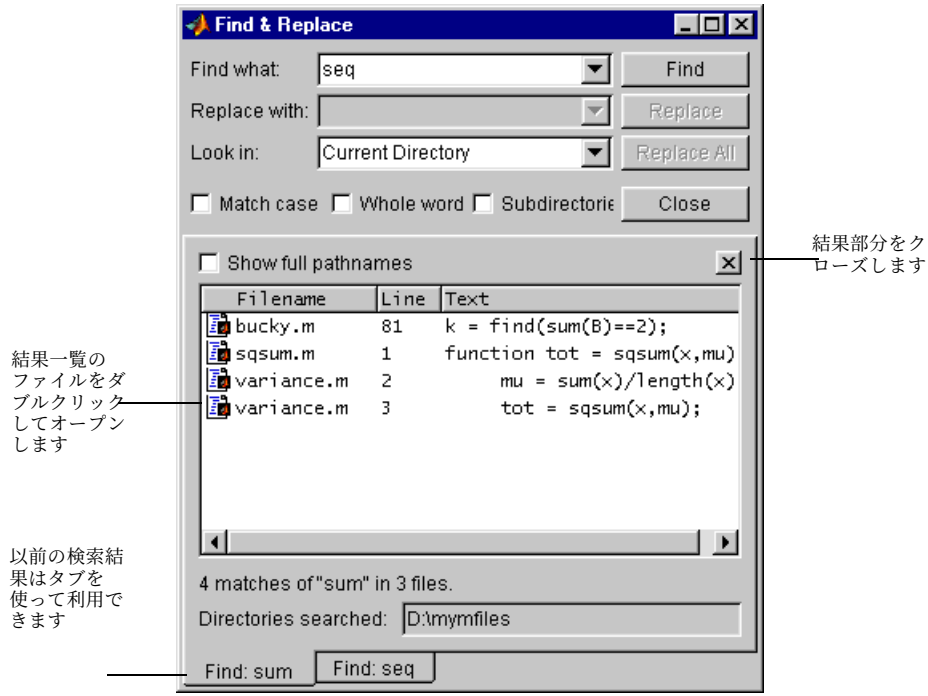


2 指定した文字列が使われている部分をすべて求め、Find & Replace ダイアログボックスの入力欄を埋めていきます。

- 検索するもの (Find what) フィールドに検索文字列を入力します。
- Look in リストボックスで検索対象のディレクトリを選択するか、このフィールドに検索対象ディレクトリの名前を入力してください。
- Match case (大文字小文字の別を検知) または Whole word (完全一致) をチェックして、検索を限定します。
- サブディレクトリも検索したい場合は、Subdirectories をチェックしてください。

3 **検索 (Find)** をクリックします。

検索結果は、**Find & Replace** ダイアログボックスの下欄に表示されます。表示される内容は、ファイル名、検索文字列を含む M- ファイルのラインナンバーとそのラインの内容です。



4 つぎのいずれかの操作により、検索結果リストに表示された任意の M- ファイルを開きます。

- ファイルをダブルクリックする。
- ファイルを選択して、**Enter** または **Return** キーを押す。
- 右クリックで表示されるコンテキストメニューからファイルを選択し、**開く (Open)** を選択します。

この操作により、M- ファイルがエディタ上に開かれ、**Find & Replace** ダイアログボックスの検索結果表示欄に表示された番号のラインにスクロールします。

5 さらに他の検索を行う場合は、それぞれの検索結果に、カレントの結果リストの下に表示されるタブを使ってアクセスすることができるようになります。このタブをクリックすれば、検索条件と検索結果を参照できます。

等価な機能をもつ関数 . lookfor を使うと、サーチパス上の全ての M- ファイルについて、そのファイルを構成するヘルプテキストの最初の行 (H1 ライン) から指定した文字列を検索できます。

ファイル内で指定文字列の置換

ファイル内で文字列の検索を行った後に検索した文字列の置換を実行することができます。

1 MATLAB エディタでファイルをオープンしてください。カレントディレクトリブラウザの **Find & Replace** ダイアログボックス検索結果リストからファイルをオープンすることができます。5-28 ページの " ファイル内で指定文字列の検索 " のステップ 4 を参照してください。文字列を置換したいファイルが、エディタのカレントファイルになっていることを確認してください。

2 **Find & Replace** ダイアログボックスの **Look in** フィールドで、文字列を置換したいファイルの名前を選択します。

Find & Replace ダイアログボックスの**置換 (Replace)** ボタンが選択可能になります。

3 **Replace with** フィールドに、指定した文字列を置換するテキストを入力します。

4 選択したライン上の文字列を置換するには、**置換 (Replace)** をクリックします。また、カレントで開いているファイル全体の置換を行うには、**すべてを置換 (Replace All)** をクリックしてください。

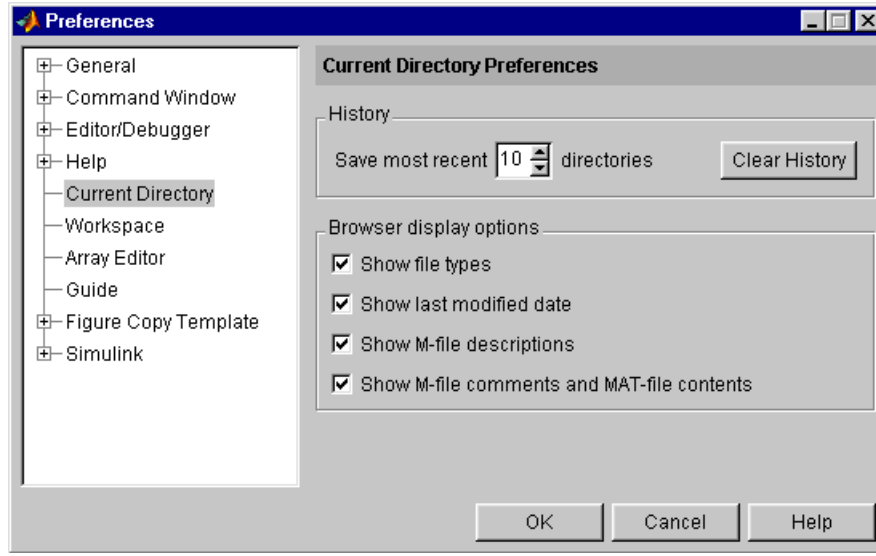
これにより、テキストが置換されます。

5 ユーザが行った変更をセーブするために、エディタの**ファイル(File)**メニューから**保存 (Save)** を選択してください。

カレントディレクトリブラウザの設定

設定 (Preference) を使うと、履歴リストに保存する最近使用したカレントディレクトリの数、カレントディレクトリブラウザに表示される情報のタイプについて設定を行うことができます。

カレントディレクトリブラウザの**ファイル (File)** メニューから**設定 (Preferences)** を選択してください。すると、**カレントディレクトリの設定 (Current Directory Preferences)** パネルが、**設定 (Preferences)** ダイアログボックスに表示されます(下図)。



履歴 (History)

カレントディレクトリブラウザツールバーのドロップダウンリストやMATLABデスクトップのカレントディレクトリフィールドには、最近使用したカレントディレクトリのリストが表示されます。このリストは、カレントの MATLAB セッションで使用されている全てのカレントディレクトリで構成されています。

ディレクトリの削除． リストからディレクトリを削除するには、**履歴の消去 (Clear History)** を選択してください。直ちにリストがクリアされます。

ディレクトリをセーブする． MATLAB セッションが終了すると、ディレクトリのリストは保持されます。**Save most recent directories** フィールドを使うと、つぎのMATLABセッションの起動時にリストに表示するディレクトリの数を指定することができます。

表示オプションブラウザ

カレントディレクトリブラウザでは、ファイルタイプ、直近の修正日、M- ファイルの説明、M- ファイルのコメント、MAT- ファイルの内容を表示することができま

す。このうちのどの情報を表示するかを**表示オプションブラウザ (Browser display options)** から選択してください。

6

データのインポートとエクスポート

テキストデータのインポート 6-3

ASCII データのエクスポート 6-15

バイナリデータのインポート 6-20

バイナリデータのエクスポート 6-26

HDF データの取り扱い 6-30

低水準ファイル I/O 関数の使用 6-52

MATLAB は、ディスクファイルまたはクリップボードからデータをワークスペースにロードしたり (*インポート*) ワークスペース変数をディスクファイルにセーブする (*エクスポート*) ための様々な方法をサポートしています。どの方法を使用するかは、実行する操作がインポートまたはエクスポートであるのか、また、データのフォーマットが、テキストまたはバイナリなのかによって選択してください。

注意 MATLAB にデータをインポートする最も簡単な方法は、インポートウィザードを使用する方法です。インポートウィザードは、データのフォーマットを知っている必要はありません。ユーザは、単にデータを含むファイルを指定するだけで、インポートウイザードが自動的に処理します。詳細は、6-3 ページの " インポートウィザードを使ったテキストデータの取り扱い" と6-20ページの"インポートウィザードを使ったバイナリデータファイルのインポート " を参照してください。

テキストデータ

テキストフォーマットでは、データの値はアルファベットや数字キャラクタを表示する American Standard Code for Information Interchange(ASCII) となります。ASCII テキストデータは、テキストエディタで表示することができます。テキストデータの取り扱いの詳細は、以下の項目を参照してください。

- 6-3 ページの " テキストデータのインポート "
- 6-15 ページの "ASCII データのエクスポート "

バイナリデータ

バイナリフォーマットではデータの値は ASCII コードではなく、テキストエディタで開けません。バイナリファイルは、イメージ、サウンド、他の形式を表すデータを含みます。バイナリデータの取り扱いの詳細は、以下を参照してください。

- 6-20 ページの " バイナリデータのインポート "
- 6-26 ページの " バイナリデータのエクスポート "

他のフォーマット

MATLAB は、Hierarchical Data Format(HDF) を使った科学的データのインポートもサポートしています。詳細は、第 6 章 , "HDF データの取り扱い " を参照してください。

低水準 I/O

MATLAB は、C スタイルの低レベル I/O 関数もサポートしています。この低水準 I/O 関数では、任意のデータフォーマットを処理することができます。詳細は、6-52 ページの " 低水準ファイル I/O 関数の使用 " を参照してください。

テキストデータのインポート

テキストデータをワークスペースにインポートする最も簡単な方法は、MATLAB インポートウィザードを使用する方法です。ユーザは、インポートウィザードを起動し、インポートしたいデータを含むファイルを指定します。インポートウィザードは、数値データファイルを、テキストヘッダを含む否かに依らず、ほぼ自動的に処理することができます。詳細については、" インポートウィザードを使ったテキストデータの取り扱い " を参照してください。

MATLAB コマンドラインから、または、M- ファイルの一部として、インポート操作を実行したい場合、MATLAB インポート関数のいずれかを使用してください。使用する関数の選択は、ファイル内のデータタイプやデータのフォーマットに依存します。MATLAB は数値データを取り扱う関数やアルファベットデータ、数値データの双方を処理することができる関数を用意しています。ユーザのデータに適した関数の選択に関する情報は、6-8 ページの " インポート関数を使ったテキストデータの取り扱い " を参照してください。

注意　データを MATLAB ワークスペースにインポートすると、ワークスペース内と同じ名前をもつ変数に関しては、ワークスペース変数を書き換えます。

インポートウィザードを使ったテキストデータの取り扱い

インポートウィザードを使って、テキストデータをインポートするには、つぎの操作を行ってください。

1 MATLAB の**ファイル (File)** メニューから**データのインポート (Import Data)** を選択し、インポートウィザードをスタートします。MATLAB は、ファイル選択ダイアログボックスを表示します。また、関数 uiimport を使って、インポートウイザードをスタートすることもできます。

  インポートウィザードを使って、データをクリップボードからインポートするには、MATLAB の **Edit** メニューの **Paste Special** オプションを選択します。また、MATLAB コマンドウィンドウで右クリックし、表示されたコンテキストメニューから **Paste Special** を選択する方法でもインポートできます。クリップボードからインポートを行う場合は、ステップ 3 に進んでください。

2 ファイル選択ダイアログボックスで、インポートしたいファイルを指定し、**開く (Open)** をクリックしてください。この操作により、インポートウィザード

は、ファイルをオープンし、その内容を処理しようとします。

3 個々のデータアイテムを区別するために用いるキャラクタを指定します。このキャラクタは、" デリミタ "、または、" 区切り文字 " と呼ばれるものです。インポートウィザードでは、多くの場合に使用するデリミタを設定することができます。ただし、ユーザのテキストファイルで使用するキャラクタを指定しなくてはなりません。詳細は、6-4 ページの " デリミタの指定 " を参照してください。インポートウィザードが正しくデータを処理すると、**つぎへ (Next)** をクリックしてください。

4 インポートしたい変数を選択します。デフォルトでは、インポートウィザードは、ある変数内の全ての数値データ、他の変数の全てのテキストデータを選択します。しかし、ユーザが他のオプションを指定することも可能です。詳細は、6-6 ページの " インポートする変数の選択 " を参照してください。

5 **終了 (Finish)** をクリックして、データのワークスペースへのインポートを完了します。

デリミタの指定

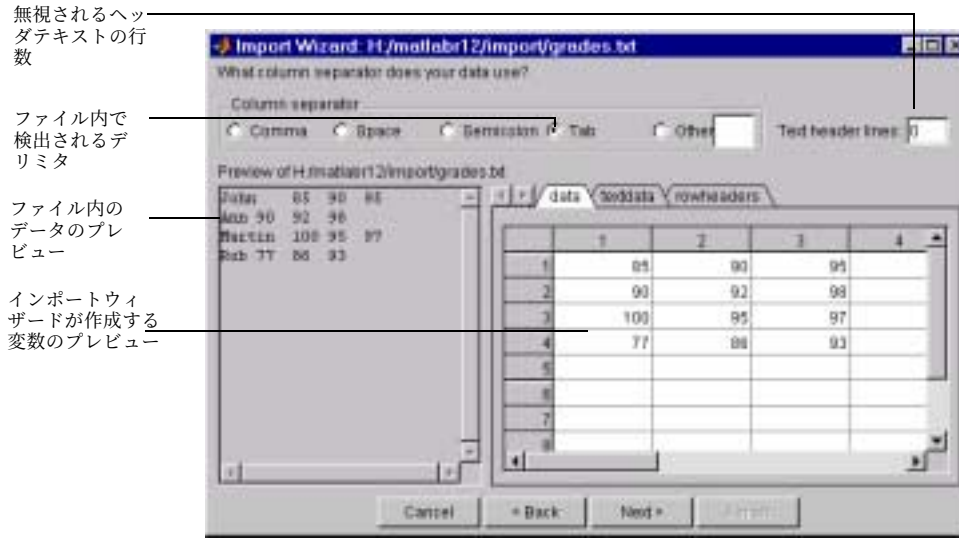
インポートウィザードがテキストファイルを開くか、クリップボードからデータをコピーする場合、ダイアログボックスのプレビューパンに、生データの一部が表示されます。ユーザは、これを使って、対象ファイルにインポートしたいデータが含まれているかどうかを確認することができます。

インポートウィザードは、データで使用されているデリミタを識別して、データを処理しようとします。インポートウィザードは、デリミタを解釈し、それに基づいて作成した変数の表示を行います。複数の変数を表示するには、タブパネルを使用してください。

たとえば、下の図では、インポートウィザードは、つぎのサンプルファイル grades.txt を開いています。

```
John      85      90      95
Ann       90      92      98
Martin   100      95      97
Rob       77      86      93
```

•



上の図では、インポートウィザードがタブキャラクタを、ファイルで使用するデリミタであると解釈し、データから、つぎの 3 つの変数を作成している点に注目してください。

- data -- ファイル内の全ての数値データからなる変数です。
- textdata -- ファイル内の全てのテキストからなる変数です。
- rowheaders -- データの一番左の列に表示される名前からなる変数です。

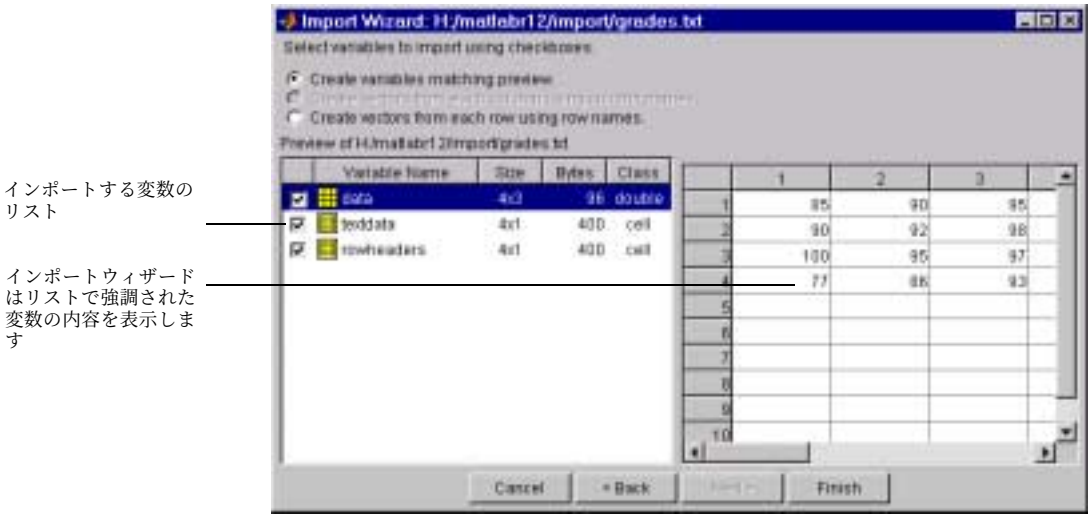
アルファベットデータの取り扱い . インポートウィザードは、行、または、列のヘッダを使用するデータファイルを認識し、別の分離した変数に、これらのヘッダを抽出します。ファイルの中のデータより前に位置するテキストヘッダを全て無視するよう設定することもできます。

他のデリミタの指定 . インポートウィザードが、データで使用されているデリミタを確定できない場合、前と同様に生のデータをプレビュー表示しますが、表示される変数は正しいものでない可能性があります。ユーザのデータが、コンマ以外のキャラクタ (スペース、タブ、セミコロン) をデリミタとして使用している場合、**その他 (Other)** ボタンをクリックし、テキストボックスにデリミタとして使用するキャラクタを入力してください。インポートウィザードは、直ちにデータを再処理し、新しい変数を作成、表示します。

インポートする変数の選択

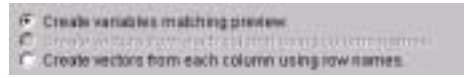
インポートウィザードは、ユーザのデータから作成した変数のリストを表示します。このリストで名前の隣に表示されるチェックボックスをクリックし、インポートしたい変数を選択してください。デフォルトでは、全ての変数が選択されます。

インポートウィザードは、ダイアログボックスの右側に表示されるリストで強調表示されている変数の内容を表示します。それ以外の変数の内容を確認したい場合、内容表示したい変数をクリックしてください。インポートしたい変数を選択したら、**つぎへ (Next)** をクリックしてください。

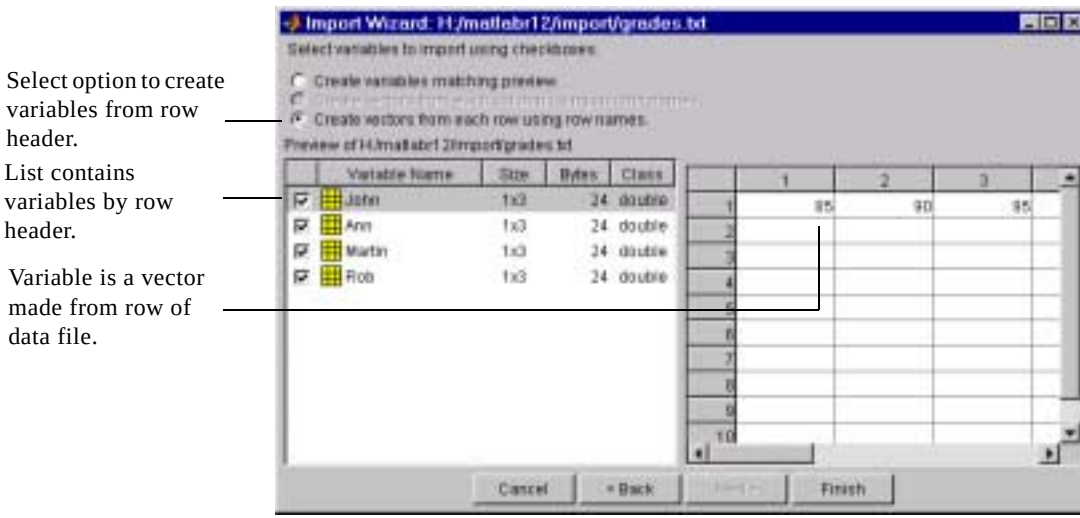


変数の選択を変更． デフォルトでは、インポートウィザードは、ファイル内の全ての数値データを１つの変数に格納します。ファイルがテキストデータを含んでいる場合、インポートウィザードは、それを別々の変数に格納します。また、ファイルが、行または列ヘッダを含んでいる場合、インポートウィザードはそれらを変数 rowheaders、または、変数 colheaders という別個の変数に格納します。

データの各行、または、各列から変数を作成し、作成した変数の名称に、行または列ヘッダのテキストを割り当てるほうがより都合が良い場合もあります。このような操作を行うには、ダイアログボックスの上部に表示されるボタンリストから適切なボタンをクリックしてください。



下記の例のように、各学生毎にその学生の学年で構成される別々の変数を作成すれば、学生の平均を容易に計算することができます。このような変数を作成するには、**Create variables from each column using row names** ボタンをクリックしてください。このオプションをクリックすると、インポートウィザードは、ファイルを再処理し、つぎの新しい変数を作成します。.



インポートを行う変数のリストを確認したら、**Next** をクリックし、データを MATLAB ワークスペースに移動します。このボタンは、インポートウィザードの終了処理も行います。インポートウィザードは、MATLAB コマンドウィンドウに、変数をワークスペースに作成したことを知らせるメッセージを表示します。つぎの例では、各変数を構成する数値テキストデータが、配列 doubles にどのようにインポートされているかに着目してください。

```
Import Wizard created variables in the current workspace.
>> whos
  Name          Size              Bytes  Class

  Ann           1x3                  24  double array
```

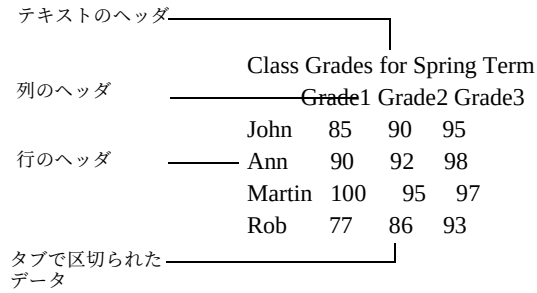
```
  John          1x3              24  double array
  Martin        1x3              24  double array
  Rob           1x3              24  double array

Grand total is 12 elements using 96 bytes
```

インポート関数を使ったテキストデータの取り扱い

コマンドライン、または、M- ファイルからテキストデータをインポートするには、MATLAB インポート関数を使用しなくてはなりません。どの関数を使用するかは、テキストファイルのデータのフォーマットに依存します。

テキストデータは、*デリミタ*、*または*、*列区切り文字*と呼ばれるテキストキャラクタを使って、行と列が統一パターンになるようにフォーマットされていなければなりません。デリミタには、スペース、コンマ、セミコロン、タブ、または、その他のキャラクタを設定できます。個々のデータ項目が、アルファベットキャラクタ、数値キャラクタ、両方の混合のいずれであっても、データを処理することは可能です。

テキストファイルには、*ヘッダライン*と呼ばれる 1 行以上のテキストを含めても構いません。つぎの例は、ヘッダテキスト、行及び列ヘッダを含んだ、タブ区切りテキストのケースを示しています。



データがどのようにフォーマットされたかを確認するには、テキストエディタでデータを表示してください。そして、フォーマットを決定したら、表 6-1 のフォーマットサンプルをスキャンし、ユーザデータのフォーマットに最も近いサンプルを探し出します。より詳しい情報を得るには、表に記載されている関連トピックを参照してください。

表 6-1: ASCII データファイルフォーマットと MATLAB インポートコマンド

データフォーマットのサンプル	ファイル拡張子	詳細
1 2 3 4 5 6 7 8 9 10	.txt .dat or other	詳細は、6-10 ページの " 数値テキストデータのインポート " を参照してください。このデータフォーマットについては、インポートウィザードを使うこともできます。詳細は、6-3 ページの " インポートウィザードを使ったテキストデータの取り扱い " を参照してください。
1; 2; 3; 4; 5 6; 7; 8; 9; 10 or 1, 2, 3, 4, 5 6, 7, 8, 9, 10	.txt .dat .csv or other	詳細は、6-11 ページの " 区切り文字を使った ASCII データファイルのインポート " を参照してください。このデータフォーマットについて、インポートウィザードを使うこともできます。詳細は、6-3 ページの " インポートウィザードを使ったテキストデータの取り扱い " を参照してください。
Ann Type1 12.34 45 Yes Joe Type2 45.67 67 No	.txt .dat,その他	詳細は、6-12 ページの " テキストヘッダを含む数値データのインポート " を参照してください。
Grade1 Grade2 Grade3 91.5 89.2 77.3 88.0 67.8 91.0 67.3 78.1 92.5	.txt .dat or other	詳細は、6-12 ページの " テキストヘッダを含む数値データのインポート " を参照してください。このフォーマットのデータについては、インポートウィザードを使うこともできます。詳細は、6-3 ページの " インポートウィザードを使ったテキストデータの取り扱い " を参照してください。

MATLAB インポート関数を知っているが、いつそれを使ったらいいのかが確かでない、という場合は、表 6-2 を参照してください。この表では、それぞれの関数の特徴が比較対照されています。

Table 6-2: ASCII データインポート関数の機能比較

関数	データタイプ	デリミタ	戻り値の数	注意
csvread	数値データ	コンマのみ	1 個	主に、スプレッドシートのデータに使用します。バイナリフォーマットスプレッドシートインポート関数も参照してください。
dlmread	数値データ	任意のキャラクタ	1 個	フレキシブル、かつ簡単に使用できる関数です。
fscanf	アルファベットや数値データ、ただし、双方のタイプ共に単一の出力変数に出力します。	任意のキャラクタ	1 個	低水準ファイル I/O ルーチンの 1 つです。この関数を使うには、先ず、fopen を使ってファイル識別子を得ることが必要です。また、読み込みが終わった後は、fcloseでファイルをクローズしなくてはなりません。
load	数値データ	スペースのみ	1 個	簡単に使用できる関数です。load の関数型を使用して、出力変数の名前を指定してください。
textread	アルファベット、数値データ	任意のキャラクタ	複数の出力値	簡単に使用できるフレキシブル、かつ高機能の関数です。書式文字列を使って、どのように変換を行うかを指定してください。

数値テキストデータのインポート

数値データのみからなるデータファイルをインポートする場合、データが使用している区切り子に依存して、種々の MATLAB インポート関数を使用することができます (表 6-2 参照)。最も簡単な方法は、コマンド load を使用するものです。つまり、データが長方形の場合は、各行は同数の要素をもち、最も簡単な方法は、コマ

ンドloadを使用するものです(loadコマンドは、ワークスペースの保存用のMATLABバイナリフォーマットであるMAT- ファイルのインポートにも使用できます)。

たとえば、ファイル名my_data.txtが、スペースキャラクタで区切られた2行の数値から構成されているとします。

```
1 2 3 4 5
6 7 8 9 10
```

loadをコマンドとして使用すると、データをインポートし、ファイル名と同じ名前(ファイル拡張子は削除)をもつ変数をワークスペースに作成します。

```
load my_data.txt;
whos
   Name          Size         Bytes   Class

   my_data       2x5             80   double array

my_data

my_data =
    1   2   3   4   5
    6   7   8   9   10
```

ワークスペース変数にファイル名以外の名前を付けたい場合、loadを関数として使用します。つぎの例では、my_data.txtからデータをワークスペース変数Aにインポートする操作を行っています。

```
A = load('my_data.txt');
```

区切り文字を使ったASCIIデータファイルのインポート

スペース以外のキャラクタをデリミタとして使用しているファイルについても、いくつかの使用可能なインポート関数を用意しています(表 6-4 を参照)。その内、最も簡単なものは、関数dlmreadを使うものです。

たとえば、セミコロンをデリミタをして使用しているファイルph.datを考えます。

```
7.2;8.5;6.2;6.6
5.4;9.2;8.1;7.2
```

このファイルの全内容を配列Aに読み込むには、つぎのように入力します。

```
A = dlmread('ph.dat', ';');
```

関数 dlmread では、データファイルで使用しているデリミタを 2 番目の引数として指定します。たとえば、各行の最後のアイテムにデリミタを設定していない場合でも、関数 dlmread は、ファイルを正確に処理する点に注意してください。dlmread は、データ要素間のスペースキャラクタを無視します。そのため、ph.dat の内容が、つぎのような場合でも、dlmread コマンドは、十分に機能します。

```
7.2;   8.5;        6.2;6.6
5.4;   9.2   ;8.1;7.2
```

テキストヘッダを含む数値データのインポート

テキストヘッダを含むASCIIデータファイルをインポートするには、関数textreadを使用し、headerlines パラメータを指定します。関数 textread には、変換を様々な点を制御できるパラメータをあらかじめ定義することができます (これらのパラメータのリストについては、関数 textread のリファレンスページを参照してください)。headerlines パラメータを使用すれば、関数 textread を実行した際にファイルの冒頭を何行無視するか指定することができます。

たとえば、ファイル grades.dat は、1 行のテキストヘッダを含むフォーマット付き数値データで構成されています。

```
Grade1 Grade2 Grade3
78.8     55.9   45.9
99.5     66.8   78.0
89.5     77.0   56.7
```

このデータをインポートするには、つぎのコマンドを使用します。

```
[grade1 grade2 grade3] = textread('grades.dat','%f %f %f',...
'headerlines',1)

grade1 =
   78.8000
   99.5000
   89.5000

grade2 =
   55.9000
   66.8000
```

```
    77.0000

grade3 =
    45.9000
    78.0000
    56.7000
```

アルファベットと数値が混ざったデータのインポート

データファイルにアルファベットと数値ASCIIデータの双方が混在している場合、関数 textread を使って、データをインポートすることができます。関数 textread は、複数の出力変数を戻し、各変数のデータタイプを指定することもできます。

たとえば、つぎのファイル mydata.dat は、アルファベットと数値データの双方で構成しています。

```
Sally    Type1 12.34 45 Yes
Larry    Type2 34.56 54 Yes
Tommy    Type1 67.89 23 No
```

注意　主に数値データで構成されているが、ヘッダがテキストになっているASCIIデータファイルを読み込む場合は、6-12 ページの " テキストヘッダを含む数値データのインポート " を参照してください。

このファイル mydata.dat の全内容をワークスペースに読み込むには、関数 textread の引数として、データファイル名、フォーマット文字列を指定します。フォーマット文字列には、それぞれのデータアイテムをどのように解釈するかを指定する変換設定子も含まれます。たとえば、変換設定子 %s は文字データ、%f は浮動小数点データ、等々を指定します (フォーマット設定子の完全なリストについては、関数 textread のリファレンスページを参照してください)。

ユーザのフォーマット文字列の各変換設定子に関して、別々の出力変数を指定しなければなりません。関数 textread は、ファイル内のそれぞれのデータアイテムを、フォーマット文字列に従って、処理し、出力変数に値を保存します。出力変数の数は、フォーマット文字列で指定した変換設定子の数と一致しなければなりません。

つぎの例題で、関数 textread は、ファイル mydata.dat を読み、ファイルの終端子に出会うまで、ファイルの中の各ラインにフォーマット文字列を適用します。

```
[names,types,x,y,answer] = textread('mydata.dat','%s %s %f ...
   %d %s',1)
names =
    'Sally'
    'Larry'
    'Tommy'

types =
    'Type1'
    'Type2'
    'Type1'

x =
   12.3400
   34.5600
   67.8900

y =
    45
    54
    23

answer =
    'Yes'
    'Yes'
    'No'
```

ユーザのデータが、スペース以外のキャラクタをデリミタとして使用している場合、関数 textread のパラメータ 'delimiter' にデリミタとしてユーザが使用しているキャラクタを指定してください。たとえば、ファイル mydata.dat が、セミコロンをデリミタとして使用しているとしましょう。この場合は、つぎのように使用します。

```
[names,types,x,y,answer]=textread('mydata.dat','%s %s %f ...
   %d %s', 'delimiter',';')
```

オプションパラメータの詳細については、関数 textread のリファレンスページを参照してください。

ASCII データのエクスポート

MATLAB は、様々なフォーマットの ASCII データをエクスポートするための複数の方法をサポートしています。たとえば、行と列がスペースを区切りとして使用している数値データから構成するテキストファイルに、MATLAB 行列をエクスポートすることも可能です。

この節では、複数の共通の ASCII フォーマットを MATLAB 関数を使って、どのようにエクスポートするかを説明します。説明には、つぎの内容も含まれています。

- 6-16 ページの " デリミタを使った ASCII データファイルのエクスポート "
- 6-18 ページの "diary コマンドを使ったデータのエクスポート "

どの関数を使うかは、処理するデータ量とフォーマットによって決定してください。

ユーザが処理しようとしているデータの説明が記載されている節がわからない場合、表 6-3 のデータフォーマットサンプルをスキャンして、作成したいデータフォーマットにほぼマッチするサンプルを検索します。そして、表の中の参照節を読んでください。

MATLAB エクスポート関数について知っているが、いつそれを使えば良いのかわからない場合、各関数の機能比較を行った表 6-4 を参照してください。

注意　他のアプリケーションで必要な形式で、データファイルを書き出すための C または Fortran ルーチンを所有している場合、MEX ファイルを作成してデータを書き込んでください。詳細は、*Application Program Interface Guide* を参照してください。

表 6-3: ASCII データファイルフォーマットと MATLAB エクスポートコマンド

データフォーマットサンプル	MATLAB エクスポート関数
1 2 3 4 5 6 7 8 9 10	これらのオプションに関する詳細は、6-16 ページの " デリミタを使った ASCII データファイルのエクスポート " と 6-18 ページの "diary コマンドを使ったデータのエクスポート " を参照してください。
1; 2; 3; 4; 5; 6; 7; 8; 9; 10;	詳細は、6-16 ページの " デリミタを使った ASCII データファイルのエクスポート " を参照してください。この例題は、セミコロンをデリミタに使ったファイルですが、他のキャラクタをデリミタとして指定することも可能です。

表 6-4: ASCII データエクスポート関数の機能比較

関数	使用対象	デリミタ	注意
diary	数値データ、または、セル配列	スペースのみ	小さい配列に使用します。データファイルをエディットして、余分なテキストを削除する必要があります。
dlmwrite	数値データ	任意のキャラクタ	簡単、かつ、フレキシブルに使用できます。
fprintf	アルファベットや数値データ	任意のキャラクタ	低水準ファイル I/O ルーチンの一部です。ファイル識別子を取得するために、関数 fopen を使用する必要があります。また、書き出しを終了した後は、関数 fclose を実行しなくてはなりません。
save	数値データ	タブまたはスペース	簡単に使用できます。出力値は高精度です。

デリミタを使った ASCII データファイルのエクスポート

デリミタ (区切り文字) を使った ASCII データファイルとして、配列をエクスポートするには、-ASCII オプションを指定して、save コマンドを使用するか、dlmwrite 関数を使用するかのいずれかを行ってください。save コマンドは、使い易い関数です。しかし、dlmwrite 関数は、柔軟性をもち、デリミタを任意のキャラクタと

して指定でき、値の範囲を設定することで、ある配列の一部をエクスポートすることもできます。

save コマンドの使用

つぎの配列 A を

```
A = [ 1 2 3 4 ; 5 6 7 8 ];
```

エクスポートするには、つぎのように、コマンドを使用します。

```
save my_data.out A -ASCII
```

生成されたファイルをテキストエディタで参照すると、つぎのように表示されます。

```
1.0000000e+000   2.0000000e+000   3.0000000e+000   4.0000000e+000
5.0000000e+000   6.0000000e+000   7.0000000e+000   8.0000000e+000
```

デフォルトでは、save はスペースをデリミタとして使用しますが、-tabs オプションを指定すれば、スペースの代わりにタブをデリミタとして使用することもできます。

save を使ってキャラクタ配列を ASCII ファイルに書き出す際、ASCII と等価なキャラクタをファイルに書き出します。キャラクタ文字列 'hello' をファイルに書き出す場合は、save は以下の値を書き出します。

```
104 101 108 108 111
```

dlmwrite 関数の使用

ASCII フォーマットで配列をエクスポートするため、ファイルで使用するデリミタを指定するには、dlmwrite 関数を使います。

たとえば、つぎの配列 A

```
A = [ 1 2 3 4 ; 5 6 7 8 ];
```

を ASCII データファイルでエクスポートするには、つぎのコマンドを使用します。

```
dlmwrite('my_data.out',A, ';')
```

この操作により生成されたファイルをテキストエディタで参照すると、つぎのように表示されます。

```
1;2;3;4 5;6;7;8
```

dlmwrite 関数は、最終行にデリミタを挿入しません。

デフォルトでは、デリミタを特に指定しなければ、dlmwrite 関数は、コンマをデリミタとして使用します。コンマの代わりにスペース (' ') や空の引用符 ('') をデリミタとして指定したり、デリミタを使用せずにエクスポートすることも可能です。

diary コマンドを使ったデータのエクスポート

小さい数列、または、セル配列をエクスポートするには、diary コマンドを使用できます。diary コマンドは、ディスクファイル内にユーザが MATLAB を起動しているときの履歴を作成します (グラフィックスは除く)。

たとえば、ワークスペース内に、つぎの配列 A を保持しているとします。

```
A = [ 1 2 3 4; 5 6 7 8 ];
```

このとき、MATLAB プロンプト上で、つぎのコマンドを実行して、diary コマンドを使って、この配列をエクスポートすることができます。

1 diary関数をアクティブにします。作成する出力ファイルdiaryに別の名前を設定することもオプションとして可能です。

```
diary my_data.out
```

2 エクスポートしたい配列の内容を表示します。つぎの例題は、配列 A を表示しています。この時、セル配列、または、他の MATLAB データタイプを表示することも可能です。

```
A =
    1     2     3     4
    5     6     7     8
```

3 diary 関数を無効にし、セッションの記録を停止します。

```
diary off
```

この操作により、diary は、ファイル my_data.out を作成し、関数を無効にするまでの間に MATLAB セッションで実行した全てのコマンドをこのファイルに記録します。

```
A =

    1     2     3     4
    5     6     7     8
```

```
diary off
```

4 履歴ファイルを、テキストエディタ内でオープンして、関係のない部分を全て削除してください。

バイナリデータのインポート

バイナリデータをインポートする最も簡単な方法は、インポートウィザードを使用する方法です。ユーザは、インポートウィザードを簡単に起動し、インポートしたいデータを含むファイルを指定することができます。詳細は、6-20 ページの "インポートウィザードを使ったバイナリデータファイルのインポート " を参照してください。

MATLAB コマンドラインからインポート操作を実行したい場合や、M- ファイルの一部にインポート操作を組み込みたい場合、MATLAB インポート関数を使用してください。MATLAB は、様々なバイナリフォーマットデータ (イメージファイルやスプレッドシートデータファイル等) のインポートを可能にする多数の関数を用意しています。インポート関数のどの関数を使用するかは、ファイル内のデータタイプとデータのフォーマットによって決定してください。

注意 データを MATLAB ワークスペースにインポートすると、ワークスペース内と同じ名前をもつ変数に関しては、ワークスペース変数を書き換えます。

インポートウィザードを使ったバイナリデータファイルのインポート

インポートウィザードを使って、テキストデータをインポートするには、つぎの操作を行ってください。

1 MATLAB の**ファイル (File)** メニューから**データのインポート (Import Data)** を選択し、インポートウィザードを起動します。MATLAB は、ファイル選択ダイアログボックスをオープンします。この他にも、関数 uiimport を使って、インポートウィザードをスタートすることも可能です。

インポートウィザードを使って、データをクリップボードからインポートするには、MATLAB の**編集 (Edit)** メニューから **Paste Special** を選択します。また、MATLAB コマンドウィンドウで右クリックし、表示されたコンテキストメニューから **Paste Special** を選択する方法でもインポートできます。クリップボードからインポートを行う場合は、ステップ 3 に進んでください。

2 ファイル選択ダイアログボックスで、インポートしたいファイルを指定し、**オープン (Open)** をクリックしてください。この操作により、インポートウィザードは、ファイルをオープンし、その内容を処理しようとします。詳細につ

いては、変数の内容の表示を参照してください。

3 インポートしたい変数を選択してください。デフォルトでは、インポートウィザードはファイルのデータタイプに従って変数を作成します。

4 **終了 (Finish)** をクリックして、データのワークスペースへのインポートを完了します。

変数の内容の表示

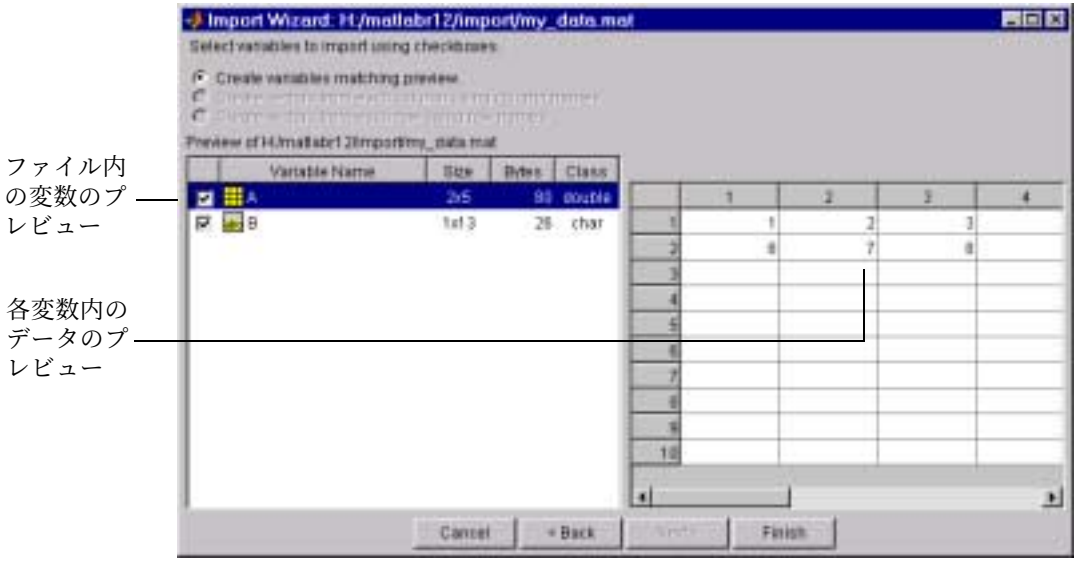
インポートウィザードがバイナリデータファイルをオープンすると、ファイル内部のデータを処理し、ファイル内で検知したデータから変数を作成します。

たとえば、つぎのサンプル MAT- ファイル my_data.mat をインポートウィザードを使って、インポートしたとします。

```
A =
    1 2 3 4 5
    6 7 8 9 10


B =
    a test string
```

これにより、2 変数が生成され、プレビューパンにリスト表示されます。変数名の隣にあるチェックボックスをクリックし、インポートしたい変数をこのリスト

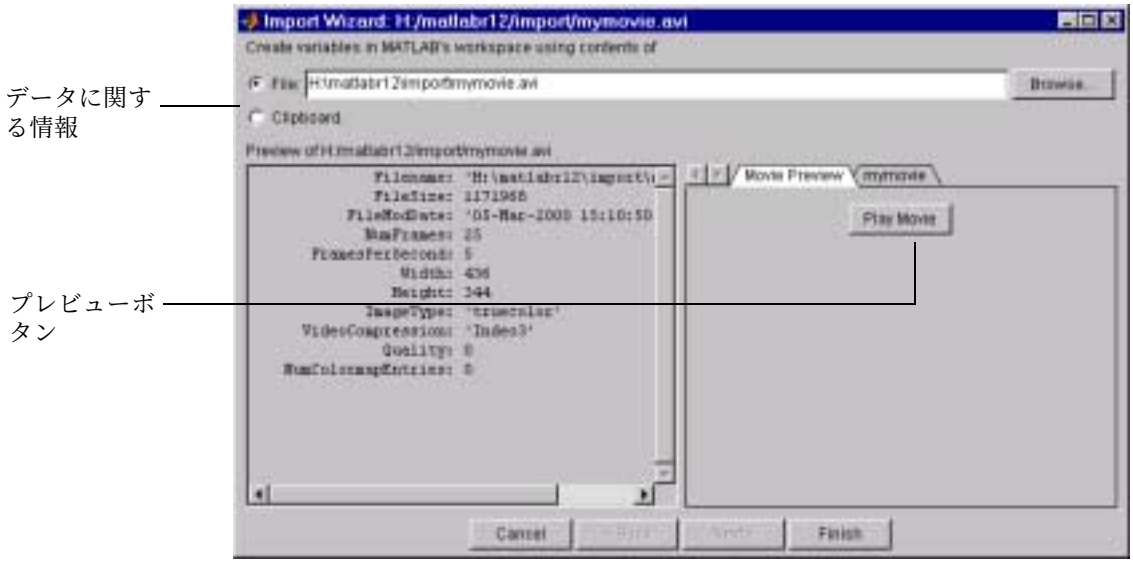
から選択してください。デフォルトでは、生成した変数全てが選択されています。



イメージやサウンドファイル等、他のバイナリデータファイルについては、インポートウィザードは、左側のパンにデータ情報を表示し、右側のパンにプレビューボタンを表示します。このプレビューボタンを押すと、データを見る（聞く）ことができます。

たとえば、インポートウィザードを使って、Audio Video Interleaved(AVI) フォーマットでムービーをインポートすると、インポートウィザードは、つぎのような

ダイアログボックスを表示します。



インポート関数を使ったバイナリデータのインポート

コマンドライン、または、M- ファイルからバイナリデータをインポートするには、MATLAB のインポート関数を使用する必要があります。テキストファイルのデータのフォーマットに依り、関数を選択してください。

特定のバイナリデータフォーマットを取り扱うように設計された関数を検出するには、表 6-5 に表示したデータフォーマットのリストをスキャンして探してください。表には、バイナリフォーマットと、それらのフォーマットをインポートする MATLAB 高水準関数が、追加的情報と共にリストアップされています。

表 6-5: バイナリデータフォーマットと MATLAB インポート関数

データフォーマット	ファイル拡張子	詳細
オーディオファイル	.au .wav	Sun Microsystems プラットフォーム上でオーディオファイルのインポートを行うには、auread 関数を使用します。Microsoft Windows システムで、オーディオファイルのインポートを行うには、wavread 関数を使用します。
Audio-video Interleaved(AVI)	.avi	Audio-video Interleaved(AVI) ファイルのインポートには、aviread 関数を使用します。
Hierarchical Data Format(HDF)	.hdf	HDF イメージファイルのインポートには、関数 imread を使用します。他の全ての HDF ファイル関する詳細情報については、6-30ページの"HDFデータの取り扱い" を参照してください。
イメージファイル	.jpeg .tiff .bmp, .gif .png, .hdf .pcx, .xwd	様々なフォーマットのイメージファイルをインポートするには、関数 imread を使用します。詳細は、グラフィックスイメージファイルの読み、書き、検索を参照してください。
MATLAB 固有フォーマット (MAT- ファイル)	.mat	MATLAB 固有フォーマットのデータをインポートするには、load コマンドを使用します。詳細は、5-7 ページの " セーブされたワークスペースのロード " を参照してください。
スプレッドシート	.xls .wk1	Excel スプレッドシートのデータをインポートするには、xlsread 関数を、Lotus 123 のデータをインポートするには、wk1read 関数を使用してください。

MATLABバイナリデータインポート関数をアルファベット順に整理したリストについては、表 6-6 を参照してください。

データフォーマットを取り扱う高水準関数をサポートしていない場合、バイナリデータのフォーマットがわかっている場合、MATLAB の低水準のファイル I/O 関数を使用することができます。詳細は、6-52 ページの " 低水準ファイル I/O 関数の使用 " を参照してください。

Table 6-6: バイナリデータインポート関数

関数	ファイル拡張子	データフォーマット
auread	.au	Sun Microsystems フォーマットのサウンドデータをインポートします。
aviread	.avi	AVI フォーマットのオーディオビジュアルデータをインポートします。
hdf	.hdf	Hierarchical Data Format(HDF) のデータをインポートします。HDF イメージファイルのインポートには、関数 imread を使用します。他の全ての HDF ファイル関する詳細情報については、6-30 ページの "HDF データの取り扱い " を参照してください。
imread	.jpeg,.tiff,.bmp,.png, .hdf,.pcx, xwd, .gif	種々のフォーマットのイメージファイルをインポートします。
load	.mat	MAT- ファイルフォーマットで、MATLAB ワークスペース変数をインポートします。
wavread	.wav	Microsoft Windows フォーマットで、データをインポートします。
wk1read	.wk1	Lotus 123 スプレッドシートフォーマットで、データをインポートします。
xlsread	.xls	Microsoft Excel フォーマットで、データをインポートします。

バイナリデータのエクスポート

ある特定のバイナリフォーマットでセーブされたバイナリデータをエクスポートするには、そのフォーマットを取り扱うために設計された MATLAB 高水準関数が必要です。

特定のバイナリデータフォーマットを取り扱うために設計された関数を検出するには、表 6-7 にリストされているデータフォーマットをスキャンしてください。表には、バイナリフォーマットと、それらをエクスポートするために使用する MATLAB の高水準関数が、付加情報と共に表示されています。

表 6-7: バイナリデータフォーマットと MATLAB エクスポート関数

データフォーマット	ファイル拡張子	詳細
オーディオファイル	.au .wav	Sun Microsystems プラットフォーム上で、オーディオファイルのエクスポートを行うには、auwrite 関数を使用します。Microsoft Windows フォーマットで、オーディオファイルのエクスポートを行うには、wavwrite 関数を使用します。
Audio Video Interleaved(AVI)	.avi	AVI データを、AVI フォーマットで、MATLAB フィギュアにエクスポートするには、多重定義された avifile、addframe、close 関数を使う必要があります。詳細は、6-28 ページの "MATLAB グラフを AVI フォーマットでエクスポート " を参照してください。
Hierarchical Data Format(HDF)	.hdf	HDF イメージファイルのエクスポートには、関数 imwrite を使用します。他の全ての HDF ファイルの関する詳細情報については、6-30 ページの "HDF データの取り扱い " を参照してください。
イメージファイル	.jpeg .tiff .bmp, .png .hdf, .pcx .xwd	様々なフォーマットのイメージファイルをエクスポートするには、関数 imwrite を使用します。詳細は、グラフィックスイメージファイルの読み、書き、検索を参照してください。
MATLAB 固有フォーマット (MAT- ファイル)	.mat	MATLAB 固有フォーマットのデータをエクスポートするには、save コマンドを使用します。詳細は、5-5 ページの " カレントのワークスペースのセーブ " を参照してください。
スプレッドシート	.xls .wk1	Lotus123 フォーマットのデータをエクスポートするには、wk1write 関数を使用してください。

MATLABバイナリデータエクスポート関数をアルファベット順に整理したリストについては、表 6-6 を参照してください。

特定のデータフォーマットをエクスポートするための高水準関数がMATLABに用意されていない場合、バイナリデータがどのようにフォーマットされているかがわかれば、MATLAB 低水準ファイル I/O 関数を代わりに用いて、エクスポートすることもできます。詳細は、6-52 ページの " 低水準ファイル I/O 関数の使用 " を参照してください。

表 6-8: バイナリデータエクスポート関数

関数	ファイル拡張子	データフォーマット
addframe	.au	カレント軸のスナップショットをとり、AVI ファイルオブジェクトに変換します。avifile も参照してください。
auwrite	.au	Sun Microsystems フォーマットで、サウンドデータをエクスポートします。
avifile	.au	AVI フォーマットでオーディオビジュアルデータをエクスポートします。AVI ファイルオブジェクトを生成します。addframe も参照してください。
hdf	.hdf	Hierarchical Data Format(HDF) で、データをエクスポートします。HDF イメージファイルのエクスポートには、関数 imwrite を使用します。他の全ての HDF ファイル関する詳細情報については、6-30 ページの "HDF データの取り扱い " を参照してください。
imwrite	.jpeg, .tiff, .bmp, .png, .hdf, .pcx, .xwd	いろいろなフォーマットで、イメージファイルをエクスポートします。
save	.mat	MAT- ファイルで、MATLAB 変数をエクスポートします。
wavwrite	.wav	Microsoft Windows プラットフォームで、サウンドデータをエクスポートします。
wk1write	.wk1	Lotus 123 スプレッドシートフォーマットで、データをエクスポートします。

MATLAB グラフを AVI フォーマットでエクスポート

MATLAB では、一連のグラフをムービーとしてセーブすることができます。セーブしたグラフは、movie 関数を使って再生できます。MATLAB ムービーを、他の MATLAB ワークスペース変数同様、MAT- ファイルフォーマットでセーブしてエクスポートすることも可能です（MATLAB ムービーに関する情報については、*Using MATLAB Graphics* の " アニメーション" 節を参照してください）。

一連の MATLAB グラフを、MATLAB を使わずに表示、保存できるフォーマットでエクスポートするには、AVI(Audio Video Interleaved) フォーマットでフィギュアをセーブします。AVI フォーマットは、アニメーションやビデオクリップを PC 上で実行された Windows システム、UNIX システムで再生することを可能にするフォーマットです。

AVI フォーマットのムービーの作成

一連の MATLAB グラフを AVI フォーマットのムービーファイルでエクスポートするには、つぎのように行います。

1 関数 avifile により、AVI ファイルを作成する。

2 一連のグラフをキャプチャし、関数 addframe を使って、AVI ファイルに加えます。

3 AVI ファイルに対して、多重定義された関数 close を使って、AVI ファイルを閉じます。

注意 既存の MATLAB ムービーを AVI ファイルに変換するときは、関数 movie2avi を使用してください。

たとえば、つぎのサンプルコードは、一連の MATLAB グラフを AVI ファイル mymovie.avi としてエクスポートするものです。コメントとして行の最後に番号が付されていますが、この番号はサンプルコードの後に書かれた注釈の番号と対応しています。

```
aviobj = avifile('mymovie.avi','fps',5);

for k=1:25
    h = plot(fft(eye(k+16)));
    set(h,'EraseMode','xor');
```

```
    axis equal;
    frame = getframe(gca);
    aviobj = addframe(aviobj,frame);
end

aviobj = close(aviobj);
```

このリストの中のアイテムは、コードサンプルの中の行番号に対応します。

- 関数 avifile は、AVI ファイルを作成し、AVI ファイルオブジェクトに対するハンドルを出力します。AVI ファイルオブジェクトは、カラーマップ、圧縮、画質などの、AVI ムービーの様々な特性を制御可能にするプロパティをサポートしています。avifile は、ユーザが特に値を指定しない限り、全てのプロパティについてデフォルト値を使用します (詳細は、リファレンスの avifile を参照)。たとえば、avifile へのコールは、秒毎のフレーム数 (frames per second：fps) プロパティを明示的に設定するものです。
- サンプルコードは、ムービーに含まれる一連のグラフをキャプチャーするために、for ループを使います。AVI ムービーで一連のグラフをキャプチャーするには、通例、関数 addframe を使用します。しかし、この特別な MATLAB アニメーションは、XOR グラフックスを使っているので、先ずグラフをキャプチャーする関数 getframe を呼び出し、その後に関数 addframe を使って、キャプチャーしたフレームをムービーに追加する作業を行う必要があります。詳細については、addframe リファレンスページを参照してください。
- このサンプルコードは、関数 close をコールして、フレームをファイルに書き込み、ファイルをクローズします。

HDF データの取り扱い

Hierarchical Data Format(HDF) は、科学的データを保存するために使用される、プラットフォームに依存しない汎用の標準フォーマットです。HDF を最初に開発した National Center for Supercomputing Applications(NCSA) が、科学者が HDF フォーマットでデータの読み書きを行う際に使用するルーチンのライブラリ(C言語、Fortran) を配布しています。

この節では、MATLAB 内部で HDF フォーマットの処理に用いる NCSA ルーチンを呼び出す方法について説明します。説明されるトピックは、つぎの事柄です。

- MATLAB HDF サポートの概要
- MATLAB HDF 関数呼び出し方法
- HDF データの MATLAB ワークスペースへのインポート
- MATLAB データを HDF ファイルでエクスポート
- HDF ファイルにメタデータを挿入する
- MATLAB 固有の HDF 関数を使用する

付加的な情報

この節では、HDF の特徴やルーチンについて全て網羅することは行いません。MATLAB HDF インタフェースを有効に使うためには、このマニュアルに加えて、HDF に関する公式のマニュアルも参照しておく必要があります。公式マニュアルは、NCSA のホームページ(http://hdf.ncsa.uiuc.edu/)にいろいろなフォーマットで用意されています。特に、つぎのマニュアルを参照することをお勧めします。

- *HDF User's Guide*, このマニュアルには、HDF のキーコンセプト、モデルのプログラミングに関する説明、ライブラリルーチンを使用するためのチュートリアル情報が記載されています。
- *HDF Reference Manual*, このマニュアルには、数百の HDF ルーチン、引数、出力値に関する詳しいリファレンス情報が記載されています。

アメリカ航空宇宙局 (NASA) は、地球観測システム (EOS) で使用する標準フォーマットの１つとして、拡張 HDF フォーマットを開発しました。拡張 HDF フォーマットに関する詳細は、EOS ホームページ (http://hdfeos.gsfc.nasa.gov/hdfeos/workshop.html) に掲載されている HDF-EOS 公式マニュアルを参照してください。

最後に、MATLAB HDF 関数の使用法に関する詳細については、MATLAB オンライン関数リファレンスページを参照してください。大部分の HDF ルーチンについて、MATLAB 関数の使用法は、HDF バージョンのものとほぼ同一になっています。ただ

し、ある特定のルーチンについて、MATLAB の HDF 関数の使用法が、HDF バージョンのものと異なる場合があります。

MATLAB HDF サポートの概要

NCSA は、HDF ライブラリに含まれるルーチンを 1 つのコレクション、*Application Programming Interfaces(APIs)* としてまとめあげてきました。各 API は、特別なタイプのデータを取り扱います。たとえば、HDF Scientific Data(SD) API は、数値データの多次元配列を対象として機能します。表 6-9, サポートされている HDF アプリケーションプログラミングインタフェース (APIs) は、MATLAB がサポートしている HDF APIs がアルファベット順にまとめたものです。なお、この表にはユーティリティ関数からなる MATLAB 固有の API も記載されています。

これら標準の HDF APIs に加え、MATLAB は、HDF-EOS 拡張フォーマットもサポートしています。HDF-EOS は、数値データ、ポイントデータ、swath データの多次元配列を保存、制御、抽出する機能を有しています。詳細については、HDF-EOS マニュアルを参照してください。

注意 MATLAB は、HDF データ用の NCSA multifile APIs(バージョン 4.1r3) をサポートしています。multifile APIs は、オリジナルの NCSA APIs に取って代わるものです。MATLAB は、HDF のバージョン 5.0 をサポートしていません。バージョン 5.0 は、全く新しいフォーマットで、バージョン 4.1r3 と互換性がありません。

表 6-9: サポートされている HDF アプリケーションプログラミングインタフェース (APIs)

Application Programming Interface	頭文字	詳細
Annotations	AN	ファイルに含まれる HDF ファイル、または、データ構造体の保存、管理、抽出を行います。
General Raster Images	DF24 DFR8	ラスタイメージ、ラスタイメージの次元、パレットの記憶、制御、抽出を行います。また、連結されていないパレットの操作を行うこともできます。 注意 :HDF ラスタイメージフォーマットには、MATLAB 高水準関数 imread、imwrite を使用してください。

表 6-9: サポートされている HDF アプリケーションプログラミングインタフェース (APIs)

Application Programming Interface	頭文字	詳細
HDF Utilities	H, HD, HE	HDF ファイルのオープンとクローズ、そして、エラー処理を行う関数を提供します。
MATLAB HDF Utilities	ML	MATLAB 環境で、HDF ファイルを取り扱う際に便利なユーティリティ関数を提供します。
Scientific Data	SD	文字データ、数値データの多次元配列を、次元、属性と共に保存、制御、抽出します。
V Groups	V	他の HDF データオブジェクト (ラスタイメージ、または、V データ) のグループを作成、抽出します。
V Data	VS VF VH	テーブルにレコードとして保存された多変量データの保存、制御、抽出を行います。

MATLAB HDF 関数呼び出し方法

それぞれの HDF API は、データを読み書きし、他の関連関数を実行するルーチンから構成されています。たとえば、HDF SD API は、データをオープン (SDopen)、クローズ (SDend)、読み出す (SDreaddata) といった別々の C ルーチンで構成されています。

MATLAB は、個々の API ルーチンに対応する関数をサポートするのではなく、それぞれの HDF API を単一の関数でサポートしています。ユーザは、この単一の関数を使用して、最初の引数に、HDF API を構成する個別ルーチンの名称を指定するだけで、HDF API に含まれる全てのルーチンにアクセスすることが可能となります。

たとえば、C プログラムに含まれる HDF ファイルへのアクセスを終了するために、HDF SD API ルーチンを呼び出すとします。

```
status = SDend(sd_id);
```

MATLAB では、この API に関連する MATLAB 関数を使用するだけで、このルーチンを呼び出すことができます。HDFAPI に関連する MATLAB 関数の名前は、全て API の頭字語を含んでいます。たとえば HDF SD API のルーチンにアクセスする MATLAB 関数には、hdfsd という名前が付けられているのです。

この関数の最初の引数に、API ルーチンの名前の頭字語を取って指定し、残りの引数を必要な順番に設定します。そして、MATLAB から、SDend ルーチンをコールします。

```
status = hdfsd('end',sd_id);
```

注意　この手法を知っていれば、HDF API ルーチンを呼び出すときの MATLAB 関数の使用法をすぐに決定することができます。しかし、HDF API ルーチンの中には、特に、データの出力に出力引数を使うルーチンでは、MATLAB とコール形式 (引数の順番) が異なるものもあります (詳細は、6-33 ページの " 出力引数をもつ HDF ルーチンの取扱い " を参照してください)。正しいシンタックスを確認するには、MATLAB のオンラインヘルプを利用してください。

出力引数をもつ HDF ルーチンの取扱い

データを出力するために出力引数をもつ HDF API ルーチンをコールするとき、すべての出力引数をすべて、戻り値として設定する必要があります。たとえば、C のシンタックスで、SDfileinfo ルーチンは、2 つの出力引数、ndatasets と nglobal_atts に HDF ファイルに関するデータを出力します

```
status = SDfileinfo(sd_id, ndatasets, nglobal_atts);
```

このルーチンを MATLAB からコールするには、出力引数を戻り値に変更する必要があります。

```
[ndatasets, nglobal_atts, status] = hdfsd('fileinfo',sd_id);
```

出力引数の並び順と同じ順番で出力値を指定してください。対象の関数の戻り値 status は、常に最後に設定されます。

HDF ライブラリシンボリックコンテンツの取り扱い

HDF APIs の C バージョンは、ヘッダファイルの中に定義されたシンボリックなコンテンツを使って、モードとデータタイプを設定します。たとえば、SDstart ルーチンは、シンボリックなコンテンツを使って、HDF ファイルを開くモードを指定しています。

```
sd_id = SDstart("my_file.hdf",DFACC_RDONLY);
```

このルーチンを MATLAB からコールするときは、これらのコンテンツをテキスト文字列として指定します。

```
sd_id = hdfsd('start','my_file.hdf','DFACC_RDONLY')
```

MATLAB では、定数を何ら省かずに指定することはもちろんのこと、接頭辞を省いて指定することもできます。たとえば、SDstart を呼び出すときに、文字列、'DFACC_RDONLY'、'dfacc_rdonly'、'rdonly' の 3 種類のいずれかを使用することができます。ここで大文字、小文字のいずれを用いても正確な指定が可能であることに注意してください。

HDF ファイルからデータのインポート

HDF データを、MATLAB にインポートするには、固有の HDF データタイプに関連した HDF API でルーチンを使用する必要があります。各々の API は、固有のプログラミングモデル、すなわち、ルーチンを使って、HDF ファイルをオープンし、ファイルの中のデータにアクセスし、データセット (HDF ファイルの中に保存されている数値配列をデータセットと呼びます) からデータを読むことを前もって設定しています。

この概念を示すために、この節では、一つの特別な HDF API、すなわち、Scientific Data(SD) のプログラミングモデルの詳細を示します。他の HDF API の取り扱いに関する情報は、オフィシャルな NCSA ドキュメントを参照してください。

注意 以下の節で、HDF SD API 固有のルーチンを参照する場合には、MATLAB 関数名ではなく C ライブラリ名を使います。すべての例題において MATLAB シンタックスが用いられます。

HDF SD インポートプログラミングモデル

HDF SD データをインポートするには、API ルーチンを使って、つぎの作業を行なってください。

1 HDF SD データセットを含むファイルをオープンします。

2 インポートしたいファイル内のデータセットを選択します。

3 データセットからデータを読み込みます。

4 データセットと HDF ファイルへのアクセスをクローズします。

必要な操作は以上ですが、HDF ファイルの内容、または、ファイル内のデータセットに関する情報を抽出するといったいくつかの操作をオプションで行うこともできます。つぎの節で、基本操作とオプション操作の双方について、詳しく解説します。

HDF ファイルのオープン

HDF SD データセットをインポートするには、まずデータセットを含むファイルをオープンする必要があります。HDF SD API では、SDstart ルーチンを使ってファイルをオープンし、HDF インタフェースを初期化します。MATLAB では、第一引数として start を指定して関数 hdfsd を利用します。

SDstart は、以下の引数を設定します。

- オープンしたいファイル名を指定する文字列
- ファイルを開くモードを指定する文字列

たとえば、つぎのコードは、読み出し専用で、ファイル mydata.hdf を開きます。

```
sd_id = hdfsd('start','mydata.hdf','read');
```

SDstart が指定したファイルを検出し、オープンできれば、SD ファイルの識別子を戻します。この例題では、sd_id になります。オープンできない場合は、-1 になります。

HDF ファイルは、いくつかのアクセスモードをサポートしています。HDF インタフェースで定義されたシンボリック定数を使って、各モードを指定します。MATLAB では、これらの定数を文字列で指定できます。省略なしの完全なスペルで入力することもできますが、下の表にリスト表示されている省略形を入力しても構いません。

HDF ファイル作成モード	HDF シンボリック定数	MATLAB 文字列
新規ファイルの作成	'DFACC_CREATE'	'create'
読み取り専用	'DFACC_RDONLY'	'read' または 'rdonly'
Read and write access	'DFACC_RDWR'	'rdwr' または 'write'

HDF ファイルに関する情報の抽出

HDF ファイルを開いた後に、関数 SDfileinfo を使って、ファイルの内容に関する情報を得ることができます。この関数は、ファイル内のデータセットの数や、ファ

イルが何らかのグローバル属性を含んでいるかどうかについて出力します (グローバル属性に関する詳細は、6-36 ページの "HDF ファイルから属性の抽出 " を参照してください)。

唯一の引数として、SDfileinfo は、SDstart によって戻される SD 識別子 sd_id を受け入れます。この例題で、HDF ファイルは、3 つのデータセットと 1 つのグローバル属性を含んでいます。

```
[ndatasets, nglobal_atts, stat] = hdfsd('fileinfo',sd_id)

ndatasets =
    3

nglobal_atts =
    1

status =
    0
```

HDF ファイルから属性の抽出

HDF ファイルは、ファイル内部に自身の情報を記録しておくことができます。すなわち、オプションで、属性と呼ばれるもので、含んでいるデータを記述するものです。HDF ファイルに関連した属性は、グローバル属性と呼ばれます (ユーザは、データセット、または、次元と属性を関連付けします。これらの属性に関する詳細は、6-48 ページの " メタデータを HDF ファイルに埋め込む " を参照してください)。

HDF SD API では、SDreadattr ルーチンを使って HDF ファイルからグローバル属性を抽出します。MATLAB では、第一引数として readattr を指定して関数 hdfsd を利用します。その他の引数として以下を指定します。

- SDstart によって出力された SD ファイル識別子 (sd_id)
- 参照したい属性を指定するインデックス値。HDF は、最初のグローバル属性がインデックス値 0, 2 番目がインデックス値 1、という、ゼロベースのインデックスを使用します。

たとえば、つぎのコードは、最初のグローバル属性の内容 (単純なキャラクタ文字列 'my global attribute') を出力します。

```
attr_idx = 0;
[attr, status] = hdfsd('readattr', sd_id, attr_idx)
```

```
attr =
    my global attribute

stat =
    0
```

MATLAB は、自動的に出力値 attr のサイズを、属性内のデータに適するものに設定します。

名前で属性を抽出する． 属性は値と名前をもっています。属性の名前が判明している場合は、関数 SDfindattr を使い、インデックス値を定めて属性を抽出することもできます。MATLAB では、第一引数として findattr を指定して関数 hdfsd を利用します。

その他の引数として以下を指定します。

- SD ファイル識別子 (グローバル属性の場合)
- 属性の名前を指定するテキスト文字列

関数 SDfindattr は、SD オブジェクトに関連した全ての属性を検索します。指定した名前の属性を検知できれば、関数 SDfindattr は、その属性のインデックスを出力します。出力されたインデックス値を使って、関数 SDreadattr で属性を抽出することが可能となります。

つぎの例では、my_att と名付けられた属性のインデックスを、関数 SDfindattr により取得し、そのインデックスを、関数 SDreadattr の引数として、受け渡す処理を行います。

```
attr_idx = hdfsd('findattr',sd_id,'my_att');
[attr, status] = hdfsd('readattr', sd_id, attr_idx);
```

HDF ファイル内のデータセットの選択

HDF ファイルをオープンした後、ファイル内の読みたいデータセットを指定します。1 つの HDF ファイルが、複数のデータセットから構成されていることもあります。HDF SD API では、SDselect ルーチンを使ってデータセットを選択します。MATLAB では、第一引数として select を指定して関数 hdfsd を利用します。

この関数は、引数として以下を指定します。

- SDstart によって出力された SD ファイル識別子 (sd_id)
- 参照したい属性を指定するインデックス値。HDF は、最初のグローバル属性が

インデックス値 0, 2 番目がインデックス値 1、という、ゼロベースのインデックスを使用します。

たとえば、つぎのコードは、sd_id という識別子をもつ HDF ファイルの 3 番目のデータセットを選択しています。関数 SDselect が、ファイル内から指定したデータセットを見つけだすと、SD データセット識別子 (例題では、sds_id) が出力されます。データセットが発見できなかった場合、-1 が出力されます。

注意 例題の sd_id という HDF SD *ファイル*識別子と、sds_id という HDF SD *データセット*識別子を混同しないでください。

```
sds_idx = 2; % HDF は、ゼロベースのインデックスを使用
sds_id = hdfsd('select',sd_id,sds_idx)
```

名前でデータセットを抽出

HDF ファイルにデータセットを作る場合、データセットには名前が割り当てられます。参照したいデータセットの名前を知っていれば、インデックス値を知らなくとも、SDnametoindex ルーチンを使って、インデックスを確定することができます。

データセットに関する情報の取得

HDF ファイル内のデータセットを選択すると、関数 SDgetinfo を使って、データセットに関する情報 (配列次元のサイズ、数等) を得ることができます。関数 SDreaddata を使って、データセットを読み込む時にこれらの情報が必要となります (詳細は、6-40 ページの "HDF ファイルからデータを読み込む " を参照してください)。

HDF SD API では、SDgetinfo ルーチンを使ってこの情報を収集します。MATLAB では、第一引数として getinfo を指定して関数 hdfsd を利用します。さらに、SDselect (sds_id) で出力される HDF SD データセット識別子を指定しなければなりません。

以下の表は、SDgetinfo で出力される情報の一覧です。

出力されるデータセット情報	MATLAB データタイプ
名前	キャラクタ配列
次元の数	スカラ
各次元のサイズ	ベクトル

出力されるデータセット情報	MATLAB データタイプ
配列に保存されたデータのデータタイプ	キャラクタ配列
データセットに関連付けられた属性の数	スカラ

たとえば、つぎのコードは、識別子 sds_id をもつデータセットに関する情報を抽出します。

```
[dsname, dsndims, dsdims, dstype, dsatts, stat] =
             hdfsd('getinfo',sds_id)
dsname =
      A

dsndims =
      2

dsdims =
      5     3

dstype =
      double

dsatts =
      0

stat =
      0
```

データセット属性の抽出

HDF ファイル同様、HDF HDF SD データセットもファイル内部に情報を記録しておくことができます。すなわち、オプションでファイルの中に*属性*と呼ばれる情報を含めておくことができます。データセットに関連付けられた属性を*ローカル*属性と呼びます (属性をデータセットや次元と関連付けることもできます。これらの属性に関する情報は、6-48 ページの " メタデータを HDF ファイルに埋め込む " を参照してください)。

HDF SD API では、SDreadattr ルーチンを使ってローカル属性を抽出します。MATLAB では、第一引数として readattr を指定して関数 hdfsd を利用します。その他の引数として以下を指定します。

- SDselect で出力される HDF SD データセット識別子 (sds_id)
- 参照したい属性を指定するインデックス値。HDF は、最初のグローバル属性がインデックス値 0, 2 番目がインデックス値 1、という、ゼロベースのインデックスを使用します。

つぎのサンプルコードは、識別子 sds_id をもつデータセットに関連付けられた最初の属性の内容を出力します。この場合、属性の値は、キャラクタ文字列 'my local attribute' となります。MATLAB は、自動的に出力値 ds_attr のサイズを属性の値に適した大きさに設定します。

```
attr_idx = 0;
[ds_attr, status] = hdfsd('readattr', sds_id, attr_idx)

ds_attr =
    my local attribute

stat =
    0
```

HDF ファイルからデータを読み込む

HDF ファイルをオープンし、ファイル内のデータセットを選択すると、データセット全体、または、その一部を、関数 SDreaddata を使って、読むことができるようになります。HDF SD API では、SDreaddata ルーチンを使ってデータセットを読み込みます。MATLAB では、第一引数として readdata を指定して関数 hdfsd を利用します。その他の引数として以下を指定します。

- SDselect によって、出力された SD データセット識別子 (sds_id)
- データを読み始めたいデータセットの位置を、インデックス値からなるベクトルで指定します。HDF の語用法では、このベクトルを *start* ベクトルと呼びます。
- それぞれの読み込み操作の間に、それぞれの次元に沿ってスキップする要素数をスカラ値のベクトルで指定します。HDF の語用法では、このベクトルを *stride* ベクトルと呼びます。
- 各次元に沿って読み込む要素数を、スカラ値のベクトルで指定します。HDF の語用法では、このベクトルを *edges* ベクトルと呼びます。

たとえば、3 行 5 列の数値行列からなるデータセットの全内容を読み込みます。

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
```

以下のコードを利用します。

```
[ds_name, ds_ndims, ds_dims, ds_type, ds_atts, stat] =
                              hdfsd('getinfo',sds_id);

ds_start = zeros(1,ds_ndims); % Creates the vector [0 0]
ds_stride = [];
ds_edges = ds_dims;

[ds_data, status] =
          hdfsd('readdata',sds_id,ds_start,ds_stride,ds_edges);

disp(ds_data)
    1    2    3    4    5
    6    7    8    9   10
   11   12   13   14   15
```

この例では、つぎの点に注目してください。

- SDgetinfo の戻り値は、戻り値の次元を指定するため SDreaddata の引数として利用されます。
- データセットを最初から読み込むために、start ベクトル (ds_start) の各要素をゼロに指定しています。また、start ベクトルの長さを決める上で、SDgetinfo を、どのように利用しているかについても注意してください。
- データセットの各要素を読み込むために、stride ベクトルの各要素を 1 に指定するか、空配列に指定しています。
- データセットの各要素を読み込むために、edges ベクトルの各要素を、データセットを構成するそれぞれの次元のサイズに設定しています。

注意　関数 SDgetinfo によって、出力された次元ベクトル dsdims を利用して、edges ベクトルの値を設定してください。これは、SDgetinfo は、行を優先した順番でこれらの値を出力するからです。MATLAB は、列を優先した順番でデータをストアします。MATLAB の 3 行 5 列配列は、HDF では 5 列 3 行配列とみなされます。

データセットの一部分を読み込む

データセットの一部分を読み込むには、start、stride、edges の各ベクトルの値を、読み始めたい部分、読み込みたいデータ数に合わせて設定してください。

たとえば、以下のコードは、SDreaddata を使ってサンプルデータセットの 2 行目すべてを読み込みます。

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
```

start, stride, edges 引数では、列を優先した順番 [columns,rows] で次元を指定しなければなりません。加えて、これらの引数で用いるインデックス値は、ゼロベースでなくてはなりません。

```
ds_start = [0 1] % Start reading at the first column, second row
ds_stride = []; % Read each element
ds_edges = [5 1]; % Read a 1-by-5 vector of data

[ds_data, status] =
           hdfsd('readdata',sds_id,ds_start,ds_stride,ds_edges);

disp(ds_data)
    6    7    8    9    10
```

データセットの範囲をどのように指定するかに関する詳細情報は、6-45 ページの "MATLAB データを HDF ファイルに書き出す " を参照してください。

HDF ファイル、HDF データセットのクローズ

HDF ファイルのデータセットからデータを読み込んだら、データセットと HDF ファイルへのアクセスを終了しなくてはなりません。HDF SD API には、HDF ファイル、HDF データセットをクローズする機能も用意しています。詳細は、6-47 ページの "HDF データセットとファイルのクローズ " を参照してください。

HDF フォーマットでデータのエクスポート

MATLAB から HDF ファイルにデータをエクスポートするには、HDF データタイプに関連付けられた HDF API を使用する必要があります。各々の API は、固有のプログラミングモデル、すなわち、ルーチンを使って、HDF ファイルをオープンし、ファイルの中のデータにアクセスし、データセット (HDF ファイルの中に保存されている数値配列をデータセットと呼びます) からデータを読むことを前もって設定しています。

この概念を示すために、この節では、一つの特別な HDF API、すなわち、Scientific Data(SD) のプログラミングモデルの詳細を示します。他の HDF API の取り扱いに関する情報は、オフィシャルな NCSA ドキュメントを参照してください。

HDF SD エクスポートプログラミングモデル

HDF SD データをエクスポートするプログラミングモデルは、つぎの操作を実行します。

1 HDF ファイルの作成、または、既存の HDF ファイルのオープン

2 ファイル内にデータセットを作成、または、既存のデータセットの選択

3 データをデータセットへ書き込む

4 データセットと HDF ファイルへのアクセスをクローズ

ユーザは、HDF ファイルに自身のデータを説明する情報を埋め込むことができます。詳細は、6-48 ページの " メタデータを HDF ファイルに埋め込む " を参照してください。

HDF ファイルの作成

MATLAB データを HDF フォーマットでエクスポートするには、先ず、HDF ファイルを作成するか、既存の HDF ファイルをオープンする必要があります。HDF SD API では、SDstart ルーチンを利用します。MATLAB では、第一引数として start を指定して関数 hdfsd を利用します。他の引数として、以下を指定します。

- HDF ファイルに割り当てたい名前を示すテキスト文字列 (または、既存の HDF ファイル名)
- HDF SD インタフェースファイルアクセスモードを指定するテキスト文字列

たとえば、つぎのコードは、mydata.hdfと名付けられたHDFファイルを作成します。

```
sd_id = hdfsd('start','mydata.hdf','DFACC_CREATE');
```

ファイルを作成、または、オープンすることができれば、SDstart は、SD ファイルの識別子 (この例では、sd_id) を出力します。作成、または、開くことができなければ、-1 を出力します。

DFRACC_CREATE アクセスモードを指定すると、関数 SDstart は、ファイルを作成し、HDF SD マルチファイルインタフェースを初期化します。DFACC_CREATE モードを指定し、ファイルが既に存在している場合、関数 SDstart は実行できずに、-1 を出力します。既存の HDF ファイルを開くには、HDF read、または、write モードに設定する必要があります。これらのモードで、関数 SDstart をどのように使用するかについては、6-35 ページの "HDF ファイルのオープン " を参照してください。

HDF データセットの作成

HDF ファイルを作成、または、既存の HDF ファイルをオープンした後、エクスポートしたい各 MATLAB 配列に対して、HDF ファイルで、データセットを作成する必要があります。HDF SD API では、SDcreate ルーチンを使ってデータセットを作成します。MATLAB では、第一引数として create を指定して関数 hdfsd を利用します。既存のデータセットを HDF ファイルに書き出すには、関数 SDselect を使用してください。詳細は、6-37 ページの "HDF ファイル内のデータセットの選択 " を参照してください。

関数 SDcreate には、つぎの引数が使用可能です。

引数	MATLAB データタイプ
適切な HDF SD ファイル識別子	SDstart から出力
データセットに割り当てたい名前	テキスト文字列
データセットのデータタイプ	テキスト文字列。データタイプの指定に関する情報は、6-34 ページの "HDF ファイルからデータのインポート " を参照してください。
データセットの次元数。これは、HDF の用語では、rank と呼びます	スカラ数値
各次元のサイズ	ベクトル

これらの引数に割り当てる値は、エクスポートしたい MATLAB 配列によって決まります。たとえば、つぎの 3 行 5 列の MATLAB 配列をエクスポートするには、

```
A = [ 1 2 3 4 5 ; 6 7 8 9 10 ; 11 12 13 14 15 ];
```

この部分的に表しているコードの通りに引数の値を設定します。

```
ds_name = 'A';
ds_type = 'double';
ds_rank = ndims(A);
ds_dims = fliplr(size(A));

sds_id = hdfsd('create',sd_id,ds_name,ds_type,ds_rank,ds_dims);
```

関数 SDcreate が、うまくデータセットを作成できれば、SD データセット識別子 (sds_id) を出力します。失敗すれば、SDcreate は、-1 を出力します。

注意　この例では、コードの一部が、次元引数 (ds_dims) の中の値を逆にどのように並べるかに注意してください。この処理は、MATLAB の size 関数が列優先で、HDF が行優先であるためです。

一度データセットを作成すると、その特性を変更することはできません。しかし、データセットを構成するデータを修正することは可能です。データを修正するには、データセットへのアクセスを初期化し、SDselect を使用し、6-45 ページの "MATLAB データを HDF ファイルに書き出す " で説明している方法で、データセットの書き出し処理を行います。

MATLAB データを HDF ファイルに書き出す

HDF ファイルを作成し、creati ファイル内にデータセットを作成した後、データセット全体または一部を書き出すことが可能となります。HDF SD API では、SDwritedata ルーチンを利用します。MATLAB では、第一引数として writedata をしてして関数 hdfsd を利用します。

以下の表は、SDwritedata のその他の引数の一覧です。

引数	MATLAB データタイプ
正しい SD データセット識別子 (sds_id)	SDcreate によって出力
データを書き出し始めたいデータセットの位置。HDF の語用法では、このベクトルを *start* ベクトルと呼びます。	インデックス値からなるベクトル
それぞれの書き出し操作の間に、それぞれの次元に沿ってスキップする要素数。HDF の語用法では、このベクトルを *stride* ベクトルと呼びます。	スカラ値のベクトル
各次元に沿って書き出す要素数。HDF の語用法では、このベクトルを *edges* ベクトルと呼びます。	スカラ値のベクトル
書き出される MATLAB 配列	double の配列

注意 start, stride, edges 引数の値は、MATLAB で用いられる、列を優先する順位でなく、行を優先する順位でなければなりません。edges 引数の値として割り当てられる前に、関数 size で出力されるベクトルの次元の順番を逆にする関数 fliplr の使い方を注目してください。

これらの引数に割り当てる値は、エクスポートしたい MATLAB 配列によって決まります。たとえば、つぎのコードの一部は、MATLAB の 3 行 5 列の倍数の配列

```
A = [ 1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15 ];
```

を HDF ファイルに書き出したものです。

```
ds_start = zeros(1:ndims(A)); % Start at the beginning
ds_stride = [];               % Write every element.
ds_edges = fliplr(size(A));   % Reverse the dimensions.

stat = hdfsd('writedata',sds_id,
                          ds_start, ds_stride, ds_edges, A)
```

データをデータセットに書き出すことができれば、関数 SDwritedata は、0 を出力します。そうでなければ、-1 を出力します。

注意 SDwritedata は書き込み操作をキューにします。これらのキューになっている書き込み操作の実行を確認するには、SDend ルーチンを使ってファイルをクローズする必要があります。詳細は、6-47 ページの "HDF ファイルのクローズ " を参照してください。便宜上、MATLAB は関数 MLcloseall を提供し、これにより、1 回の呼び出しで、オープンしているすべてのデータセットとファイル識別子をクローズします。詳細は、6-50 ページの "MATLAB HDF ユーティリティ API の使用 " を参照してください。

データをデータセットの一部に書き出す

データセットの一部分を書き出すには、start、stride、edges の各ベクトルの値を使って、書き出したいスタート位置と書き出したいデータ数に合わせて設定してください。

つぎの例題は、サンプルデータセットの2番目の行のすべての要素を、SDwritedata を使って、

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
```

ベクトル B と置き換えるものです。

```
B = [ 9 9 9 9 9]
```

例題では、start ベクトルは、2 行目の最初の列に書き込み操作を始めたい点をベクトルで示しています。HDF は、ゼロベースのインデックス値をどのように使用し、列の次元を最初に指定するかに着目してください。MATLAB では、この位置を (2,1) とします。edges 引数には、書き出すデータの次元を指定します。書き出されるデータ配列のサイズと edge で指定した値は必ず一致しなくてはなりません。

```
ds_start = [0 1] % Start writing at the first column, second row.
ds_stride = []; % Write every element.
ds_edges = [5 1]; % Each row is a 1-by-5 vector.

stat = hdfsd('writedata',sds_id,ds_start,ds_stride,ds_edges,B);
```

HDF データセットとファイルのクローズ

データを HDF ファイルの中のデータセットに書き出した後は、データセットへのアクセスをクローズする必要があります。HDF SD API では、SDendaccess ルーチンを使ってデータセットをクローズします。MATLAB では、第一引数として endaccess を指定して関数 hdfsd を利用します。その他の唯一の引数として、この例題では、有効な HDF SD データセット識別子 sds_id を指定します。

```
stat = hdfsd('endaccess',sds_id);
```

HDF ファイルのクローズ

データをデータセットに書き出し、データセットをクローズした後に、HDF ファイルもクローズする必要があります。HDF SD API では、SDend ルーチンを利用します。MATLAB では、第一引数として end を指定して関数 hdfsd を利用します。その他の唯一の引数として、この例題では、有効な HSD SD ファイル識別子 sd_id を指定します。

```
stat = hdfsd('end',sd_id);
```

HDF ファイルをクローズする前に、それらのファイル内のすべてのデータセットへのアクセスをクローズする必要があります。

注意 HDF ファイルのクローズは、実行は、SDwritedata を使ってキューとなっているすべての書き込み操作を実行します。便利なことに、MATLAB HDF Utility API が提供する関数 MLcloseall は、1 回の呼び出しでオープンしているデータセットとファイル識別子をすべてクローズすることができます。詳細は、6-50 ページの "MATLAB HDF ユーティリティ API の使用 " を参照してください。

メタデータを HDF ファイルに埋め込む

HDF ファイルに、ユーザが作成したデータを解説する情報をオプションで埋め込むことができます。HDF は別々の注釈 API を定義しますが、HDF SD API は注釈機能を含んでいます。ここでは、HDF SD API の注釈機能のみを説明します。Annotation API に関する情報は、オフィシャルな NCSA ドキュメントを参照してください。

属性のタイプ

HDF SD API を使って、属性と 3 タイプの HDF オブジェクトを関連付けることができます。

- An entire HDF file – ファイル属性。この属性は*グローバル*属性とも呼ばれています。グローバル属性は、一般に、ファイル内の全データセットに関係のある情報から構成されています。
- A data set in and HDF file – データセット属性。この属性は、*ローカル*属性とも呼ばれています。ローカル属性は、個々のデータセットの解説で構成されています。
- A dimension of a data set – 次元属性は、個々の SD データセットの次元に適用可能な情報を提供します。

複数の属性

ユーザは、単一の HDF オブジェクトに複数の属性を関連付けることができます。HDF は、各オブジェクトについて属性インデックスを保持しています。属性インデックスは、ゼロベースです。最初の値は、インデックス値 0、2 番目の値はインデックス値 1 等々というようにインデックスが付されています。インデックス値を指定すれば、属性にアクセスすることができます。

各々の属性は、name = value という書式になっています。ここで、name(HDF 用語では、label と呼ばれています) は、256 文字以内の長さで書かれたテキスト文字

列です。value は、同一データタイプの 1 つ、または、それ以上のエントリーで構成されています。単一の属性が複数の value をもつことも可能です。

属性と HDF SD オブジェクトの関連付け

HDF SD API では、SDsetattr ルーチンを使って属性とファイル、データセット、次元を関連付けます。MATLAB では、第一引数として setattr を指定して関数 hdfsd を利用します。その他の引数をして以下を指定します。

- オブジェクトに関連付ける適切な SD 識別子。取り得る値は、ファイル識別子 (sd_id)、データセット識別子 (sds_id)、または、次元識別子 (dim_id) です。
- 属性の名前を定義するテキスト文字列。SD インタフェースは、いくつかの名前や、データタイプをもつ前もって設定された属性をサポートしています。これらの属性に関する情報は、6-49 ページの " 前もって設定する属性の作成 " を参照してください。
- 属性の値

つぎのサンプルコードは、my_global_attr と名付けられたグローバル属性を作成し、作成したグローバル属性と sd_id という識別子の HDF ファイルとの関連付けを行います。

```
status = hdfsd('setattr',sd_id,'my_global_attr','my_attr_val');
```

注意 NCSA のマニュアルでは、SDsetattr ルーチンには、さらに、2 つの引数 (データタイプと属性値の数) を設定できます。しかし、MATLAB からこのルーチンを呼び出した場合、データタイプと属性値の数という 2 つの引数は設定されません。MATLAB HDF 関数は、データタイプと属性のサイズをユーザが指定した値から決定することができるからです。

前もって設定する属性の作成

前もって設定する属性は、SD API によって既に名前とデータタイプを定義済みである、という点を除けば、ユーザ定義属性と同一です。たとえば、SD API は、ユーザがデータセットを使って座標系を指定できる cordsys という名称の属性を定義しています。この属性が取り得る値には、'cartesian'、'polar'、'spherical' というテキスト文字列が含まれています。

プリセット属性は、アプリケーションが依存する規則を作り上げるという点で有用です。SD API は、データセットや次元についてのみ、プリセット属性を用意してい

ます。ファイルについては、前もって設定した属性を用意していません。前もって設定した属性の完全なリストについては、NCSA のマニュアルを参照してください。

HDF SD API では、SDsetattr ルーチンを使って、ユーザ定義属性の作成と同じ保法で前もって設定する属性を作成します。MATLAB では、第一引数として setattr を指定して関数 hdfsd を利用します。

```
attr_name = 'cordsys';
attr_value = 'polar';

status = hdfsd('setattr',sds_id,attr_name,attr_value);
```

SD API は、前もって設定した属性の読み書きに特化した関数も用意しています。SDsetdatastrs のような特殊な関数は、複数の相関関係にある前もって設定した属性を読み書きするいくつかの場合に、汎用の関数よりも簡単に使用することができます。なお、プリセット次元属性の読み書きには、必ず専用関数を使用してください。

MATLAB HDF ユーティリティ API の使用

標準の HDF API に加えて、MATLAB は、6-31 ページの表 6-9, サポートされている HDF アプリケーションプログラミングインタフェース (APIs), に示すように、MATLAB 環境の中で、HDF を使用するように設計されているユーティリティ関数の API(ML) もサポートしています。

たとえば、MATLAB ユーティリティ API は、全てのタイプの HDF 識別子 (例：SD ファイル識別子) を参照することを可能にする関数 MLlistinfo が含まれています。MATLAB は、HDF 識別子が作成、または、終了された場合、常に、この関数によって表示されるリストを更新しているのです。

つぎのサンプルコードは、MLlistinfo 関数を使って、全ての開いている HDF ファイル識別子やデータセット識別子をリスト表示するという操作を行います。この例題では、2 つの識別子のみをオープンします。

```
hdfml('listinfo')
No open RI identifiers
No open GR identifiers
No open grid identifiers
No open grid file identifiers
No open annotation identifiers
No open AN identifiers
Open scientific dataset identifiers:
    262144
```

```
Open scientific data file identifiers:
    393216
No open Vdata identifiers
No open Vgroup identifiers
No open Vfile identifiers
No open point identifiers
No open point file identifiers
No open swath identifiers
No open swath file identifiers
No open access identifiers
No open file identifiers
```

全てのオープンしている HDF 識別子のクローズ

オープンしている HDF 識別子を 1 回の命令で全てクローズするには、MLcloseall 関数を使用します。この命令で、全てのオープンしている HDF 識別子がクローズします。

```
hdfml('closeall')
```

低水準ファイル I/O 関数の使用

MATLAB には、ANSI 標準 C ライブラリの I/O 関数をベースにした低水準 I/O 関数が用意されています。C 言語に精通しているユーザの場合は、これらのルーチンに違和感を持つことはないでしょう。

たとえば、MATLAB ファイル I/O 関数は、C 言語ルーチンと同一のプログラミングモデルを使用しています。このプログラミングモデルでは、データを読み書きするために、つぎの操作を行います。

1. fopen を使って、ファイルをオープンします。fopen は、他の低水準ファイル I/O ルーチンで使用しているファイル識別子を出力します。
2. ファイルを操作します。
   a. バイナリデータの読み込みには、fread を使用します。
   b. バイナリデータを書き出すには、fwrite を使用します。
   c. ライン毎にファイルからテキスト文字列を読み出すには、fgets/fgetl を使用します。
   d. 書式付き ASCII データを読み込むには、fscanf を使用します。
   e. 書式付き ASCII データを書き出すには、fprintf を使用します。
3. ファイルをクローズするには、fclose を使用します。

この節には、これらの関数が読み書き操作が始まるファイルのカレントの位置にどのような影響を与えるか、またファイル内で位置を変更するにはどうするかに関する情報も記載されています。

注意 MATLAB ファイル I/O コマンドは、C 言語 I/O ルーチンをベースにしています。しかし、MATLAB のファイル I/O コマンドは、C 言語 I/O ルーチンと異なる挙動を示す点があります。たとえば、MATLAB の fread 関数は、" ベクトル化 " されており、ファイルの最後まで読み込みを続けます。つぎの節と MATLAB のリファレンスページでは、MATLAB と C 言語間の挙動の相違点を強調しながら説明を行っています。

ファイルのオープン

テキスト、または、バイナリファイルの読み書きを行う前に、fopen コマンドで、

それらのファイルを開く必要があります。

```
fid = fopen('filename','permission')
```

Permission 文字列を指定する

permission 文字列は、ユーザがファイルに対して、どのようなアクセスを要求するかを指定する文字列です。permission 文字列が、取り得る値は、以下のとおりです。

- r ：読み取り専用
- w ：書き込み専用
- a ：内容の追加のみ可能
- r+ ：読み書き可能

注意 Microsoft Windows のように、テキストファイルとバイナリファイルを区別するシステムでは、permission 文字列の中に付加的なキャラクタを必要とします。たとえば、バイナリファイルを読み取り専用でオープンするには、'rb' と設定します。

出力されたファイル識別子 (fid) の使用

ファイルのオープンが成功すると、fopen は、*ファイル識別子* (fid) と云われる非負の整数を出力します。開いたファイルを他の I/O 関数でアクセスする場合、引数にこの値を設定します。たとえば、つぎの fopen ステートメントは、penny.dat という名前のデータファイルを読み取り専用でオープンします。

```
fid = fopen('penny.dat','r')
```

open がうまく実行されない、たとえば、オープンしようとしたファイルが、存在しない場合、fopen は、つぎの操作を行ないます。

- ファイル識別子に、-1 を割り当てます。
- オプションの 2 番目の出力引数に、エラーメッセージが出力されます。エラーメッセージは、システムに依存しています。すべてのエラーが、全システムで現れるわけではないことに注意してください。関数 ferror も、エラーに関する情報を提供します。

ファイルをオープンする度に、ファイル識別子をテストすることをお勧めします。たとえば、つぎのコードは、読み込み可能なファイルの名前をユーザが入力するまでループを繰り返します。

```
fid=0;
while fid < 1
   filename=input('Open file: ', 's');
   [fid,message] = fopen(filename, 'r');
   if fid == -1
     disp(message)
   end
end
```

ここで、nofile.mat が存在せず、goodfile.mat が存在すると仮定します。ある特定のシステム上で、つぎの結果が表れます。

```
Open file: nofile.mat
Cannot open file. Existence? Permissions? Memory? . . .

Open file: goodfile.mat
```

テンポラリファイル、テンポラリディレクトリのオープン

tempdir や tempname コマンドは、ユーザのシステム上にテンポラリデータを格納する際に使用します。

関数	目的
tempdir	テンポラリディレクトリ名の取得
tempname	テンポラリファイル名の取得

ユーザは、テンポラリファイルを作成することができます。システムの中には、再起動する度にテンポラリファイルを削除してしまうものもあります。再起動する度に、テンポラリとしてファイルを設定することは、単に、ファイルをバックアップしない、という意味です。

empdir と名付けられた関数は、ユーザのシステム上で、テンポラリファイルを格納するために設計されたフォルダ、または、ディレクトリの名前を出力します。たとえば、UNIX システム上で tempdir を実行すると、/tmp ディレクトリを出力します。

MATLAB には、テンポラリディレクトリに格納されたファイル名を出力する関数 tempname も用意されています。出力されるファイル名は、テンポラリデータに対して、適切に設定されます。たとえば、テンポラリファイルの中に、あるデータを格納したい場合、つぎのコマンドを先ず使用します。

```
fid = fopen(tempname, 'w');
```

注意　tempname が、作成するファイル名は、ユニークであることが保証されているわけではありませんが、大体のケースにおいて、ほぼ、ユニークと思われます。

バイナリデータの読み込み

関数 fread は、(ファイル識別子により指定される) バイナリファイルの全体、または、一部分を読み込み、1 つの行列に格納する関数です。最もシンプルな形で、この関数はファイル全体を読み込み、それぞれの入力バイトを行列を構成するつぎの要素として解釈します。たとえば、つぎのコードは、ファイル nickel.dat からデータを読み込み、行列 A に格納します。

```
fid = fopen('nickel.dat','r');
A = fread(fid);
```

データを読み込んだ後、それをスクリーンに表示するには、関数 char を使って、A の内容をキャラクタに変換した後、データが水平方向に表示されるように転置して表示します。

```
disp(char(A'))
```

関数 char を使って、行列 A の内容を、MATLAB に数字ではなく文字列として解釈させることができます。A を転置すれば、A は、より自然で水平的なフォーマットで表示されることになります。

読み込む値の数の制御

fread は、2 番目の引数 (オプション) を使って、読み込む値の数を制御することができます (引数を設定していなければ、ファイル全体を読み込みます)。たとえば、つぎのステートメントは、fid により指定されたファイルを構成する最初の 100 個のデータ値を読み込み、列ベクトル A に格納します。

```
A = fread(fid,100);
```

このコードの中の数字 100 を、行列次元 [10 10] にすると、前と同じ 100 個のデータを読み込み、10 行 10 列行列に格納します。

それぞれの値のデータタイプの制御

fread の 3 番目の引数 (オプション) は、入力値のデータタイプを制御します。このデータタイプ引数は、各値の読み込みのバイト数と、バイトをキャラクタ、整数、浮動小数点値のいずれとして解釈するのかに関する制御を行なうことができます。MATLAB は、より広範囲の精度をサポートしているため、MATLAB 固有の文字列、または、C や Fortran と等価な文字列を指定することも可能です。

MATLAB と C、Fortran で共通の精度としては、つぎのものをあげることができます。

- 'char' と 'uchar' は、符号付きキャラクタと符号なしキャラクタです (通常 8 バイト)。
- 'short' と 'long' は、短い (通常 16 バイト) 整数と長い (通常 32 バイト) の整数です。
- 'float' と 'double' は、単精度 (通常 32 バイト) と倍精度 (通常 64 バイト) の浮動小数点値です。

注意 与えられた精度の意味は、ハードウェアプラットフォーム毎に変化します。たとえば、'uchar' は、必ずしも 8 ビットとは限りません。関数 fread は、'int8' や 'float32' のように、より細かく設定した精度をもつ数を提供します。設定をどのようにするか迷っている場合、ハードウェアプラットフォームに依存しないこれらの精度を使うことをお勧めします。精度の設定については、オンラインヘルプの関数 fread のページを参照してください。

たとえば、fid が単精度の浮動小数点値からなるファイルをオープンするように設定している場合、つぎのコマンドは、続く 10 個の浮動小数点値を読み込み、列ベクトル A に格納します。

```
A = fread(fid,10,'float');
```

バイナリデータの書き込み

fwrite 関数は、行列の要素を、指定した数値精度でファイルに書き込み、書き込まれた数値を出力します。たとえば、つぎのラインは、5 行 5 列の魔方陣を構成する 25 個の要素からなる 100 バイトのバイナリファイルを作成します。書き込まれた 25 個の要素は、それぞれ 4 バイトの整数で保存されます。

```
fwriteid = fopen('magic5.bin','w');
count = fwrite(fwriteid,magic(5),'int32');
status = fclose(fwriteid);
```

この例では、関数 fwrite は値が小さく、エラーが生じない時に限り、count 変数を 25 に設定します。

ファイル内の位置の制御

fopen を使ってファイルをオープンすると、MATLAB は、ファイル内の特定の位置を指定するファイルポジションインジケータを保持します。MATLAB は、このファイルポジションインジケータを使って、次にファイル内のどの場所から読み、書きするのかを決定します。以下の節では、ファイル内で操作を開始する位置を制御するためのノウハウを紹介します。

- ファイルポジションインジケータがファイルの終端に位置するかどうかを確認する
- ファイル内の特定の位置に移動する
- ファイルポジションインジケータのカレントの位置の検出
- ファイルポジションインジケータをリセットして、ファイルの冒頭に設定する

ファイルの終端の決定

関数 fseek と ftell は、ファイル内で次に入出力操作を行う位置の設定、検索に使用する関数です。

- 関数 fseek は、データをスキップしたり、ファイルの前の方に移動するといった操作によって、ファイルポジションを再設定する関数です。
- 関数 ftell は、指定したファイルについて、ファイルポジションインジケータのオフセットをバイト単位で出力します。

関数 fseek は、つぎのように使用します。

```
status = fseek(fid,offset,origin)
```

fid は、対象ファイルのファイル識別子です。offset は、正または負のオフセット値で、バイト単位で指定します。origin は、移動量を計算する際の原点となる値で、文字列で指定します。

'bof'	ファイルの最初
'cof'	ファイルの中のカレントの位置
'eof'	ファイルの末尾

ファイルポジションの識別

関数 fseek と ftell が、どのように機能するのかについて、次の簡単な M- ファイル考えてみましょう。

```
A = 1:5;
fid = fopen('five.bin','w');
fwrite(fid, A,'short');
status = fclose(fid);
```

このコードは、1 から 5 までの数字をバイナリファイル five.bin に書き込む操作を行います。関数 fwrite を呼び出すときに、書き込む数値要素は short で格納するように設定しています。その結果、それぞれの数字は、2 ストレージバイトを使用することになります。

ここで、five.bin を読み取り専用で再びオープンします。

```
fid = fopen('five.bin','r');
```

ここで、fseek を使って、ファイルポジションインジケータをファイルの冒頭から前に 6 バイトだけ進めてみましょう。

```
status = fseek(fid,6,'bof');
```

ファイル内の位置	bof	1	2	3	4	5	6	7	8	9	10	eof
ファイルの内容		0	1	0	2	0	3	0	4	0	5	
ファイルポジションインジケータ								↓				

さらに、関数 fread を使って、ファイル内の位置 7 と 8 で読み込んだ内容を変数 four に格納します。

```
four = fread(fid,1,'short');
```

読み込み操作により、ファイルポジションインジケータは前に進みます。カレントのファイルポジションインジケータの位置を知るために、関数 ftell をコールします。

```
position = ftell(fid)

position =
```

8

ファイル内の位置	bof	1	2	3	4	5	6	7	8	9	10	eof
ファイルの内容		0	1	0	2	0	3	0	4	0	5	
ファイルポジションインジケータ										ユ		

つぎに、fseek を使って、ファイルポジションインジケータを 4 バイトだけ後ろに戻します。

```
status = fseek(fid,-4,'cof');
```

ファイル内の位置	bof	1	2	3	4	5	6	7	8	9	10	eof
ファイルの内容		0	1	0	2	0	3	0	4	0	5	
ファイルポジションインジケータ						ユ						

後に、fread を再び呼び出して、つぎの値 (3) を読み込みます。

```
three = fread(fid,1,'short');
```

テキストファイルから文字列を 1 行ずつ読み込む

MATLAB には、書式付きテキストファイルから 1 行ずつ読み込み、文字列ベクトルに格納する操作を行う 2 つの関数 fgetl と fgets を用意しています。この 2 つの関数は、ほぼ同一の機能を提供します。唯一の相違点は、fgets が改行キャラクタを文字列ベクトルにコピーしますが、fgetlはコピーしない、という点だけです。

以下の M- ファイル関数は、fgetl の使用例をデモンストレーションするものです。この関数は、fgetl を使って、ファイル全体を 1 行ずつ読み込みます。読み込んだそれぞれの行について、引数で指定された特定の文字列 (literal) が行の中に現れるかどうかをチェックします。

指定した特定の文字列が現れた場合には、この関数は行の中に literal という文字列が現れた回数を表す数字を先ず印刷し、その後に行全体を印刷します。

```
function y = litcount(filename, literal)
% Search for number of string matches per line.

fid = fopen(filename, 'rt');
```

```
y = 0;
while feof(fid) == 0
   tline = fgetl(fid);
   matches = findstr(tline, literal);
   num = length(matches);
   if num > 0
      y = y + num;
      fprintf(1,'%d:%s\n',num,tline);
   end
end
fclose(fid);
```

この例では、badpoem と名付けた、つぎの入力データファイルをチェックします。

```
Oranges and lemons,
Pineapples and tea.
Orangutans and monkeys,
Dragonflys or fleas.
```

関数 litcount を使って、このファイルに文字列 'an' が何回現れるかを計算し、その結果と文字列を含む行全体を出力します。

```
litcount('badpoem','an')
2: Oranges and lemons,
1: Pineapples and tea.
3: Orangutans and monkeys,
```

書式付き ASCII データを読み込む

関数 fscanf は、標準 C 関数 fscanf に類似する関数です。関数 fscanf と関数 fscanf は、共に、ファイルから一つの行を読み取り、1 つまたはそれ以上の変数に割り当てるという互いによく似た挙動を示します。また、この 2 つの関数は、入力データをどのように解釈するかを制御する同一の変換設定子を使用しています。

関数 fscanf に対する変換設定子は、% キャラクタで始まります。C と共通で、よく利用する変換設定子としては、つぎのものをあげることができます。

- %s ：文字列に揃える
- %d ：10 進法ベースの整数値に揃える t

- %g ：倍精度の浮動小数点値に揃える

変換指定子内のアスタリスクを指定して、fscanf がスキップする値を指定することもできます。たとえば、%*f は、入力データ内の浮動小数点値をスキップし、%*d は入力データ内の整数値をスキップすることを意味します。

MATLAB の fscanf と C の fscanf との違い

MATLAB と C の関数 fscanf は、互いに非常によく似ていますが、わずかに異なる部分も見受けられます。たとえば、つぎの内容をもつファイル moon.dat を使って考えてみましょう。

```
3.654234533
2.71343142314
5.34134135678
```

つぎのコードは、このファイルを構成する 3 要素全てを MyData と名付けられた行列に読み込みます。

```
fid = fopen('moon.dat','r');
MyData = fscanf(fid,'%g');
status = fclose(fid);
```

このコードでは、全くループが使われていない点に注目してください。代わりに、関数 fscanf は、入力の書式が、書式設定子と整合性をもつ限り、テキストの中でデータを読み続けます。

オプションで設定するサイズ引数は、読み込む要素の数を制御します。たとえば、fid が整数文字列を含む開いた状態のファイルを指示しているとしましょう。このとき、つぎのコマンドは、列ベクトル A の中に 100 個の整数値を読み込みます。

```
A = fscanf(fid,'%5d',100);
```

つぎのラインは、100 個の整数値を 10 行 10 列 A に読み込みます。

```
A = fscanf(fid,'%5d',[10 10]);
```

関数 fscanf に関連する関数 sscanf は、ファイルではなく文字列から読み込みを行います。たとえば、つぎのラインは、2 と平方根からなる列ベクトルを出力します。

```
root2 = num2str([2, sqrt(2)]);
rootvalues = sscanf(root2,'%f');
```

書式付きテキストファイルを書き込む

関数 fprintf は、データをキャラクタ文字列に変換し、スクリーン、または、ファイルに出力します。どのような書式で出力するかは、変換設定子とオプションのテキスト文字から構成される書式制御文字列により指定します。変換設定子は、配列要素の出力を制御します。関数 fprintf は、テキストについては、そのままコピーして出力します。

よく利用される変換設定子として、つぎのものがあります。

- %e ：指数表示
- %f ：固定小数点表示
- %g ：指数表示と固定小数点表示のいずれか長さが短い方を自動的に選択、出力します。

書式設定子にオプションで設定されるフィールドを使うと、最小の桁数と精度を制御することができます。たとえば、つぎのコードは、指数関数を短い長さで表示した表からなるテキストファイルを作成します。

```
x = 0:0.1:1;
y = [x; exp(x)];
```

つぎのコードは、xとyを新しく作成されたファイルexptable.txtに書き込みます。

```
fid = fopen('exptable.txt','w');
fprintf(fid,'Exponential Function\n\n');
fprintf(fid,'%6.2f  %12.8f\n',y);
status = fclose(fid);
```

関数 fprintf の 1 回目の実行で、タイトルを出力し、2 度の改行を行います。2 回目の実行では、数値のテーブルを出力します。書式制御文字列は、テーブルの各行の書式を、つぎのように指定します。

- 6 桁の数字で、下 2 桁目に小数点を持つ固定小数点値
- 2 つのスペース
- 12 桁の数字で、下 8 桁に小数点を持つ

関数 fprintf は、配列 y の要素を列順に変換します。この関数は、全ての配列要素を変換するまで繰り返し書式文字列を使用します。

ここで、指数データファイルを読み込むために、関数 fscanf を使用します。

```
fid = fopen('exptable.txt','r');
```

```
title = fgetl(fid);
[table,count] = fscanf(fid,'%f %f',[2 11]);
table = table';
status = fclose(fid);
```

2 行目でファイルタイトルを読み込みます。3 行目では、ファイルの終わりに達するまで、各行に 2 つの浮動小数点値が格納されている数値テーブルを読み込みます。関数 count は、整合する数値の数を出力します。

関数 fprintf に関連した関数 sprintf は、ファイルではなくスクリーン上に文字列を出力します。使用例は、つぎの通りです。

```
root2 = sprintf('The square root of %f is %10.8e.\n',2,sqrt(2));
```

ファイルのクローズ

ファイルの読み書きが終わったら、関数 fclose を使って、ファイルを閉じてください。つぎのラインは、fid に関連付けられたファイルをクローズします。

```
status = fclose(fid);
```

つぎのラインは、全てのオープンしているファイルをクローズします。

```
status = fclose('all');
```

どちらの場合でも、ファイルがうまくクローズする場合は 0、クローズしない場合は -1 を出力します。

ユーザが MATLAB を終了すると、MATLAB は自動的に全てのオープンしているファイルをクローズします。しかし、MATLAB をクローズする前に、関数 fclose を使って、ファイルを確実にクローズしてください。そうしないと、不必要にシステムリソースを浪費することになります。

注意　ファイルをクローズしても、ファイル識別変数 fid はクリアされません。ただし、このファイル識別子を使って、ファイルにアクセスする操作は実行できなくなります。

7

M- ファイルのエディットとデバッグ

エディタ / デバッガの起動 7-3
エディタ / デバッガで新しい M- ファイルを作成 7-4
エディタ / デバッガで既存の M- ファイルを開く 7-4
MATLAB を起動せずにエディタを開く 7-6
エディタ / デバッガのクローズ 7-6

エディタ / デバッガで M- ファイルを作成、エディット . .7-7
M- ファイルの外観 7-7
M- ファイルの中のナビゲーション 7-9
M- ファイルのセーブ 7-14
M- ファイルの印刷 7-14
M- ファイルのクローズ 7-15

M- ファイルのデバッグ 7-16
エラーのタイプ 7-16
エラーの検出 7-16
デバッグの例題 - Collatz 問題 7-17
例題の試行 7-18
デバッグ機能の使用 7-21

エディタ / デバッガの設定 7-34
エディタ / デバッガの一般設定 7-35
エディタ / デバッガで使用するフォント / カラーの設定 . 7-36
エディタ / デバッガの表示設定 7-37
エディタ / デバッガのキーボード / インデント設定 . . 7-39
エディタ / デバッガの印刷設定 7-41

M- ファイルとは、MATLAB コードからなるファイルで、この M- ファイルを作成、編集、デバッグする手段としては、つぎのいくつかの方法があります

タスク	オプション	解説
M- ファイルの作成と編集	MATLAB エディタ	7-3 ページの " エディタ / デバッガの起動 "
	スタンドアロンモードでの MATLAB エディタ (MATLAB を実行せずに起動したケース)	7-6 ページの "MATLAB を起動せずにエディタを開く "
	Emacs または vi のようなテキストエディタ	ユーザ設定を使ってエディタをデフォルトとして指定 - 7-35 ページの " エディタ " を参照
M- ファイルのデバッグ	一般的なデバッグのヒント	" エラーのタイプ " と 7-16 ページの " エラーの検出 "
	MATLAB デバッガ	7-21 ページの " デバッグ機能の使用 "
	MATLAB デバッグ関数	7-21 ページの " デバッグ機能の使用 "

ニーズにもっとも合った編集およびデバッグの環境設定には、ユーザ設定を使います。

M- ファイルの作成に関する詳細は、「プログラミングとデータタイプ」を参照してください。

エディタ / デバッガの起動

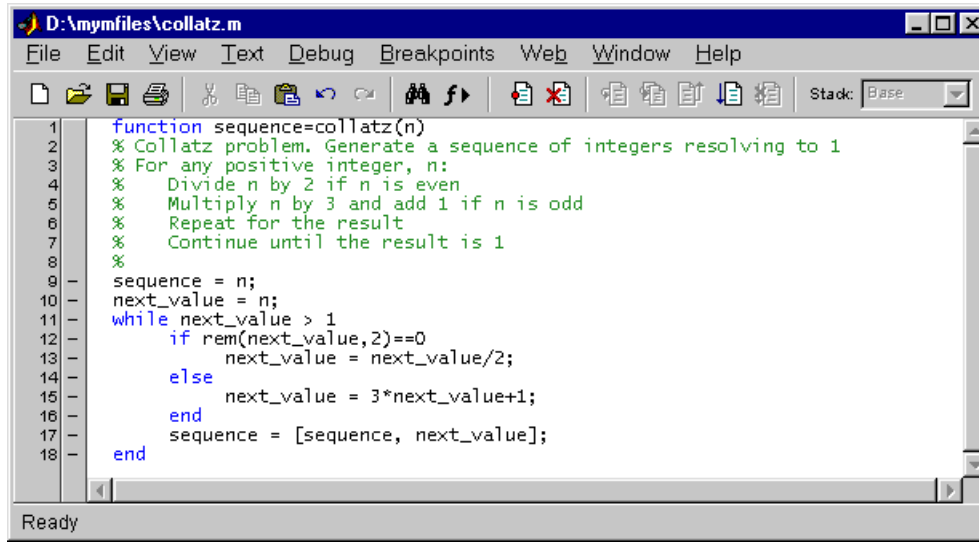
MATLAB エディタ / デバッガは、M- ファイルデバッグ機能同様に、基本的なテキストエディット機能をもつグラフィカルユーザインタフェースを提供するものです。エディタ / デバッガは、エディットとデバッグ、または、その双方の機能に使用できる単一のツールです。エディタ / デバッガを起動させるには、つぎの操作を行います。

- 7-4 ページの " エディタ / デバッガで新しい M- ファイルを作成 "
- 7-4 ページの " エディタ / デバッガで既存の M- ファイルのオープン "
- 7-6 ページの "MATLAB を起動せずにエディタを開く " (デバッガなし)

エディタ / デバッガをオープンした後で、どのような操作を行うかについては、つぎの項目を参照してください。

- 7-8 ページの " エディタ / デバッガで M- ファイルを作成、エディット "
- 7-16 ページの "M- ファイルのデバッグ "
- 7-6 ページの " エディタ / デバッガのクローズ "

下図の例は、既存の M- ファイルをエディタ / デバッガでオープンしたものです。



エディタ / デバッガウィンドウが十分な大きさでない場合、右端のツールバーボタンが表示されませんが、メニューは省略されず、ツールバーの提供する全ての機能について、メニューから同等の機能をもつアイテムにアクセスすることができます。ツールバーボタンを全て表示するには、エディタ / デバッガのウィンドウを広げてください。

エディタ / デバッガを MATLAB デスクトップの中に配置するには、**ビュー (View)** メニューから **Dock M-File** を選択してください。

エディタ / デバッガのデフォルトの外観、挙動を変更する方法については、7-34 ページの " エディタ / デバッガの設定 " を参照してください。

エディタ / デバッガで新しい M- ファイルを作成

エディタ / デバッガで新しい M- ファイルを作成するには、MATLAB ツールバーの新規ファイルボタン🗋をクリックするか、MATLAB デスクトップから **File -> New -> M-file** を選択してください。また、カレントディレクトリブラウザのコンテクストメニューを使って、新しい M- ファイルを作成することもできます。この点については、5-24 ページの " 新規ファイルの作成 " を参照してください。まだエディタ / デバッガをオープンしていない場合は、M- ファイルを作成するブランクファイルでエディタ / デバッガをオープンしてください。

エディタ / デバッガを既にオープンしている場合、ツールバーの新規ファイルボタン🗋をクリックするか、**File -> New -> M-file** を選択すると、さらに新しいファイルを作成することができます。

等価な機能をもつ関数

コマンドウィンドウで、edit とタイプすると、エディタ / デバッガ上で新しい M- ファイルを作成することができます。

エディタ / デバッガで既存の M- ファイルのオープン

エディタ / デバッガで、既存の M- ファイルをオープンするには、MATLAB、または、エディタ / デバッガツールバーのオープンボタン📂をクリックするか、**File -> Open** を選択してください。そして、**Open** ダイアログボックスから、M- ファイルを選択し、**Open** をクリックしてください。カレントディレクトリブラウザからファイルをオープンすることもできます。5-26 ページの " ファイルのオープン " を参照してください。

エディタ / デバッガと同様にデスクトップの中の**ファイル (File)** メニューの下部に表示される最近使用したファイルからオープンするファイルを選択すること

ができます。7-34 ページの " エディタ / デバッガの設定 " により、リスト上に表れるファイルの数を変更することができます。

エディタ / デバッガが、まだオープンしていない場合、表示するファイルと共にオープンします。また、既に、オープンしている場合、ファイルは、それ自身のウインドウの中か、または、7-37 ページの "Opening files in editor" に指定されているように、カレントウインドウの中にタブとして表れます。エディタ / デバッガの中のドキュメントを作成するため、カレントのドキュメントにして、それをクリックするか、**Window** メニュー、または、タブを使います。

ユーザは、MATLAB を最後に使用したときに、オープンしていたファイルを次回、起動した場合に自動的にオープンするようにプリファレンスを設定できます。7-35 ページの " エディタ / デバッガの一般設定 " の**リスタートに関して (On restart)** を参照してください。

等価な機能をもつ関数

エディタ / デバッガ内で存在している M- ファイルをオープンするには、関数 edit や open を使用します。たとえば、つぎのように入力します。

```
edit collatz
```

は、エディタ / デバッガにファイル collatz.m をオープンします。

選択したものをオープン

エディタ / デバッガの中に一つのファイルをもっている場合、その中の関数を選択し、右クリックし、コンテキストメニューから**選択したものをオープン (Open Selection)** を選択します。

関数に関するヘルプの取得

エディタ / デバッガの中に一つのファイルをもっている場合、その中の関数を選択して、右クリックをし、コンテキストメニューから**選択に関するヘルプ (Help on Selection)** を選択します。対象の関数に関するリファレンスページが、エディタ / デバッガにオープンするか、または、リファレンスページに存在していない場合、代わりに、M- ファイルヘルプが表示されます。

ユーザのソースコントロールシステムをアクセス

M- ファイルに関するソースコントロールシステムを使用する場合、エディタ / デバッガをもつものからそれにアクセスでき、ファイルをチェックアウトします。詳細は、第 9 章 , " ソースコントロールシステムとのインタフェース " を参照してください。

MATLAB を起動せずにエディタを開く

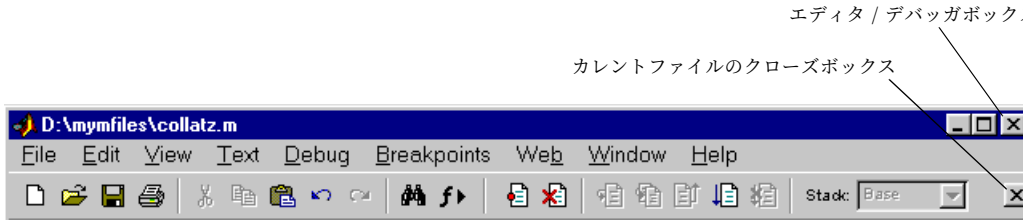
PC プラットフォーム上では、MATLAB を起動せずに MATLAB エディタだけを使用することも可能です。MATLAB エディタだけを使用するには、Windows のエクスプローラで M- ファイルをダブルクリックしてください。この操作で、M- ファイルが MATLAB エディタ上にオープンします。ファイルをオープンせずにエディタだけをオープンするには、$matlabroot/bin/win32/meditor.exe をオープンしてください。MATLAB ライセンスのタイプに関わらず、複数の meditor をオープンすることができます。これは、meditor のみの使用は、MATLAB の使用と考えないからです。

このバージョンの MATLAB エディタは、スタンドアロンのアプリケーションです。ユーザは、エディタから M- ファイルをデバッグしたり、選択対象を分析したり、MATLAB デスクトップにエディタを固定したり、エディタからヘルプにアクセスしたりすることはできません。エディタを独立でオープンした後に、MATLAB をオープンし、そこからエディタ / デバッガをオープンしても、エディタはスタンドアロンのままとなります。これらの制約を除けば、ユーザは、7-8 ページの " エディタ / デバッガで M- ファイルを作成、エディット " で説明したエディタ機能を自由に使うことができます。

PC プラットフォームに対して、MATLAB のインストールの際に、スタンドアロンエディタは拡張子 .m をもつファイルと自動的に関連付けられます。M- ファイルをダブルクリックすると、スタンドアロンエディタがオープンします。拡張子 .m をもつファイルが MATLAB のエディタ / デバッガでオープンするように、Windows Explorer を使ってこの関連を変更することができます。

エディタ / デバッガのクローズ

エディタ / デバッガをクローズするには、エディタ / デバッガのタイトルバーに表示されたクローズボックスをクリックします。このクローズボックスは、エディタ / デバッガのツールバーに表示されるクローズボックスとは異なります。ツールバーのクローズボックスは、一つのウィンドウに複数のファイルを開いているときに、カレントファイルをクローズするものです。



カレントファイルをクローズすると、１つのファイルしか開いていない状態になります。このファイルをクローズすると、エディタ / デバッガも同時にクローズします。

別個のエディタ / デバッガウィンドウに複数のファイルが、それぞれオープンしているような場合では、それぞれのウィンドウを個々にクローズするか、**Window** メニューから **Close All** を選択して、全てのウィンドウをクローズするかの、いずれかを行ってください。

エディタ / デバッガをクローズした場合、変更がセーブされていないファイルがオープンしていた場合、ファイルをセーブするよう注意するメッセージが表示されます。

エディタ / デバッガで M- ファイルを作成、エディット

既存のファイルをオープンするか、新しいファイルを作成した後に、エディタ / デバッガでテキストを入力したり、以下のエディット機能を利用できます。

- シンタックスを強調する等の、7-8 ページの "M- ファイルの外観 "
- ジャンプ、検索機能を含む、7-10 ページの "M- ファイルの中のナビゲーション "
- 7-14 ページの "M- ファイルのセーブ "
- 7-14 ページの "M- ファイルの印刷 "
- 7-15 ページの "M- ファイルのクローズ "

M- ファイルの外観

つぎの外観に関する機能を使うと、M- ファイルを、より読み易くすることができます。

- 7-8 ページの " シンタックスの強調 "
- 7-8 ページの " インデント処理 "
- 7-9 ページの " コメントを付ける "
- 7-10 ページの " バランスデリミタ (Balanced Delimiters) の表示 "

これらの多くのものは、デフォルトの挙動を指定することができます。詳細は、7-34 ページの " エディタ / デバッガの設定 " を参照してください。

シンタックスの強調

入力をいくつかを異なるカラーで表示し、if/else ステートメントのような要素同士の照合をより簡単にすることができます。詳細は、3-5 ページの " シンタックスの強調 " を参照してください。

シンタックスの強調の他にも、右),], または } 、のような最後のデリミタをタイプした時に、そのデリミタに対応する最初のデリミタとの間を強調表示する機能も用意されています。

インデント処理

フローコントロール入力は、ループ (たとえば、while/end ステートメント) を読み易くするために、自動的にインデント処理されます。

カレントのライン、または、選択したラインを、さらに左側に移動するには、**Text** メニューから **Decrease Indent** を選択します。カレントのライン、または、選択

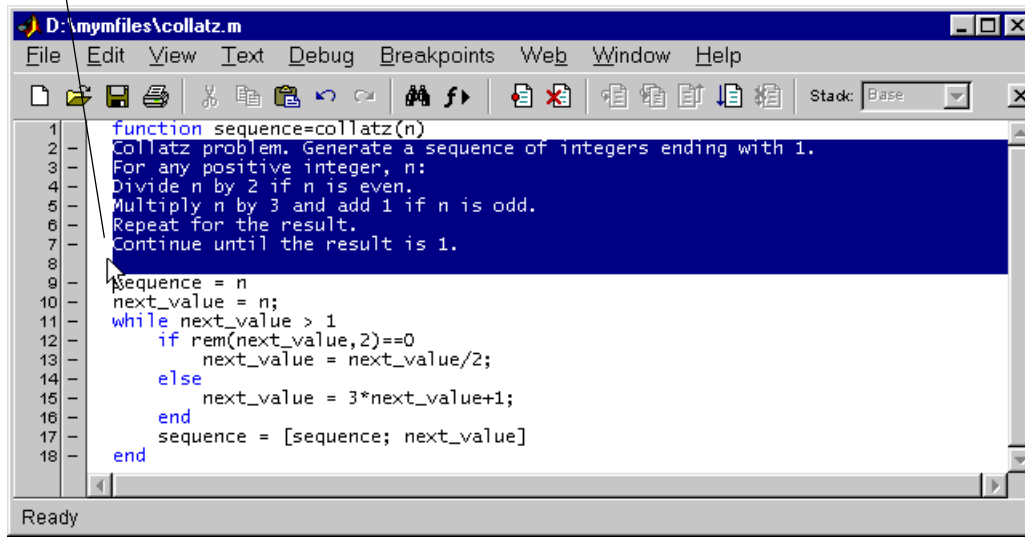
したラインを、さらに右側に移動するには、**Text** メニューから **Increase Indent** を選択します。これらの機能を使用した後に、選択したラインに自動インデント処理を適用したい場合、**Text** メニュー、または、右クリックで表示されるコンテキストメニューから **Smart Indent** を選択します。スマートインデント設定に関する詳細は、スマートインデントの設定を参照してください。

コメントを付ける

カレントライン、または、選択したラインにコメントを付けることができます。ラインを迅速に選択するには、ラインの左をクリックすると、ラインは強調点滅します。複数ラインを選択するには、ドラッグするか、ダブルクリックしてください。そして、**テキスト (Text)** メニューから**コメントを付ける (Comment)** を選択するか、または、右クリックにより表示されるコンテキストメニューから、それを選択します。コメントマーク % は、ラインの最初に加えられ、テキストのカラーは、緑色になります。

選択した複数のラインからコメントを削除するには、**Text** メニューから **Uncomment** を選択するか、右クリックにより表示されるコンテキストメニューから Uncomment を選択してください。

ラインの左側のエリアをクリックしてそのラインを選択します。
ドラッグまたはシフト - クリックして複数のラインを選択します。



バランスデリミタ (Balanced Delimiters) の表示

組になっているデリミタ、たとえば、( )、[ ]、または、{ } の内部にカーソルを置き、**テキスト (Text)** メニューから**バランスデリミタ (Balance Delimiters)** を選択すると、組になっているデリミタの内部にある文字列が強調表示されます。下の例では、カーソルは、/norm の前に置かれているので、つぎのように表示されます。

```
alfa = asin(T*v'./sqrt(diag(T*T'))/norm(v));
```

ここで、**バランスデリミタ (Balance Delimiters)** を選択すると、選択対象が、つぎのように強調表示されます。

```
alfa = asin(T*v'./sqrt(diag(T*T'))/norm(v));
```

M- ファイルの中のナビゲーション

M-ファイルでのナビゲーション用に、いくつかのオプションが用意されています。

- 7-10 ページの " 行番号にジャンプ "
- 7-10 ページの " 関数にジャンプ "
- 7-10 ページの " 選択対象の検索 "
- 7-11 ページの " 文字列の検索と置換 "

行番号にジャンプ

行番号は、エディタ / デバッガウィンドウの左側に表示されます。**Edit** メニューから **Go to Line** を選択し、エディットボックスに行番号を入力後、**OK** をクリックしてください。カレントでオープンしている M- ファイルの指定した行番号へジャンプします。

関数にジャンプ

M- ファイルに含まれる特定の関数 (サブ関数) にジャンプするには、ツールバーの関数ボタン をクリックしてください。そして、その M- ファイルに含まれる全関数のリストからジャンプしたい関数を選択します。M- ファイルに含まれる関数のリストには、M- ファイルから呼び出されている関数は表示されません。関数ステートメントで始まるカレント M- ファイルのラインのみがリスト表示されます。

選択対象の検索

先ず、カレント M- ファイル内部で文字列を選択します。つぎに、**Edit** メニューから **Find Selection** を選択します。そして、つぎに出てくる対象の文字列の箇

所が強調表示されます。続いて、つぎの検索を行う場合、**Edit** メニューから **Find Again** を選択してください。

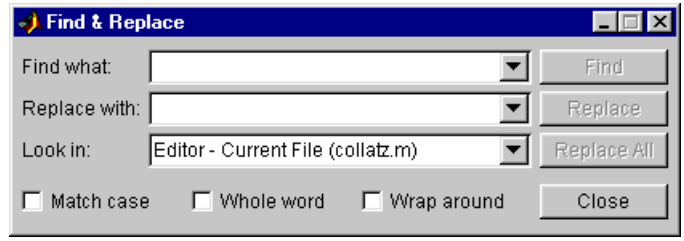
文字列の検索と置換

複数のファイルから、指定した文字列を検索したり、ファイル内の文字列を置換したりすることができます。

文字列の検索 . ファイル内の文字列を検索するには、つぎの操作を行います。

1 エディタ / デバッガツールバーの検索ボタン をクリックするか、または、**Edit** メニューから、**Find and Replace** を選択します。

Find & Replace ダイアログボックスが表示されます。このダイアログボックスは、カレントディレクトリブラウザで利用できる **Find & Replace** ダイアログボックスと同一の機能を備えています。



2 指定した文字列を全て検出するために、**Find & Replace** ダイアログボックスのフィールドを埋めていきます。

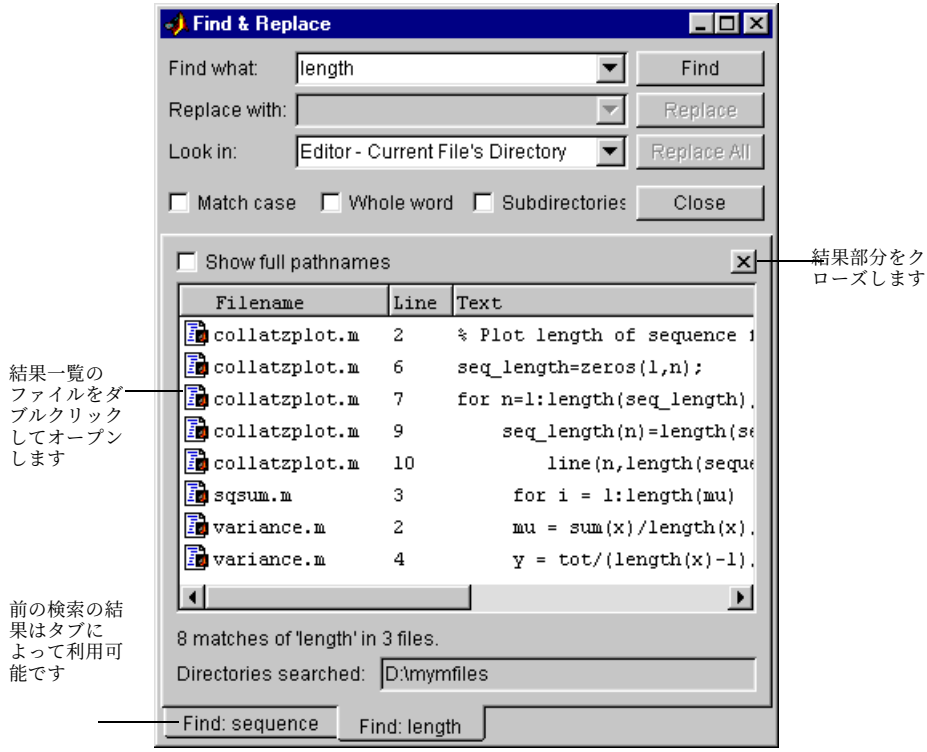
a **Find what** フィールドに文字列をタイプします。

b **Look in** リストボックスから検索対象となるファイルを選択します。

c **Match case**, **Whole word**, **Wrap around** をチェックして、検索条件を決めていきます。

3 **Find** をクリックします。

- カレントファイルについて検索を行った場合、ファイル内で、つぎに検出された文字列は、強調表示されます。
- 検索が複数のファイル間で行われた場合、検索結果は、**Find & Replace** ダイアログボックスのより下の部分に、ファイルネーム、M- ファイルの行番号、文字列を含む、行の内容を伴って表示されます。



4 つぎのいずれかの操作を行うことによって、いずれかの M- ファイルをオープンすることができます。

- オープンしたい M- ファイルをダブルクリックする
- オープンしたい M- ファイルを選択し、**Enter** または **Return** キーを押す
- オープンしたい M- ファイル上で右クリックし、表示されたコンテキストメニューから **Open** を選択する

M- ファイルがオープンされると、**Find & Replace** ダイアログボックスの結果表示セクションに表示された行番号まで自動的にスクロールします。

5 他の検索を行うと、それぞれの検索結果が、結果リストの下に表示されるタブからアクセスできるようになります。このタブをクリックすることにより、検索条件及び検索結果を参照することができます。

等価な機能をもつ関数． lookfor を使用すると、サーチパス上にある全ての M- ファイル内に含まれるヘルプの第 1 ラインから指定した文字列を検索します。

文字列の置き換え． ファイル内の文字列を検索した後、カレントファイルの指定した内容を置換することが可能となります。

1 ファイルがまだオープンされていない場合、ファイルをオープンします。また、この時、**Find & Replace** ダイアログボックスからもファイルをオープンすることができます。7-11 ページの " 文字列の検索 " を参照してください。ここで、ユーザが置換したい文字列を含むファイルが、エディタ上でカレントファイルとなっているかどうか確認してください。

2 **Find & Replace** ダイアログボックスの **Look in** フィールドが、文字列を置換したいファイル名をきちんと表示しているかどうか確認してください。これにより **Find & Replace** ダイアログボックスの **Replace** ボタンが選択できるようになります。

3 **Replace with** フィールドに、指定した文字列を置き換えるテキストをタイプします。

4 そして、**Replace** をクリックすると、選択されたラインの文字列が置換されます。または、**Replace All** をクリックすると、カレントで開かれているファイルに含まれる指定した文字列の全てについて置換が実行されます。

   以上の操作により、テキストの置換は完了します。

5 **File** メニューから、**Save** を選択し、変更をセーブします。

複数のファイルで置換を行う場合には、それぞれのファイルに対し、以上の操作を繰り返し実行してください。

M- ファイルのセーブ

M- ファイルの変更を行うと、エディタ / デバッガのタイトルバーに表示されるファイル名の後にアスタリスク (*) が付加されます。このアスタリスクは、ファイルの変更が未だセーブされていないことを示す記号です。

ファイルの変更をセーブするには、**File** メニューから **Save** コマンドをいずれか１つ選択します。

- **Save** – 既存の名前でファイルをセーブします。ファイルが新しく作成された場合には、**Save file as** ダイアログボックスが開き、ユーザは、ここでファイル名の割り当てとセーブを行います。セーブするもう１つの方法として、ツールバーのセーブボタンを用いる方法も利用できます。
- **Save As** – **Save file as** ダイアログボックスが開き、ここで、ファイル名の割り当てとセーブを行います。ファイル名を付けるときに拡張子 .m をユーザが付ける必要はありません。MATLAB がファイル名に拡張子 .m を付加する操作を、ユーザの代わりに自動で行います。
- **Save All** – 名前の付いた全てのファイルを既存のファイル名でセーブします。新しく作成されたファイルについては、**Save file as** ダイアログボックスが開き、ここで、ユーザはファイル名の割り当てとセーブを行うことになります。

ファイルをセーブし、実行することもできます。詳細は、7-23 ページの "M- ファイルの実行 " を参照してください。

ユーザのソース制御システムにアクセス

M- ファイルについてソース制御システムを使用すると、ユーザはエディタ / デバッガ内からソース制御システムにアクセスし、ファイルをチェックすることができるようになります。詳細は、第 9 章 , " ソースコントロールシステムとのインタフェース " を参照してください。

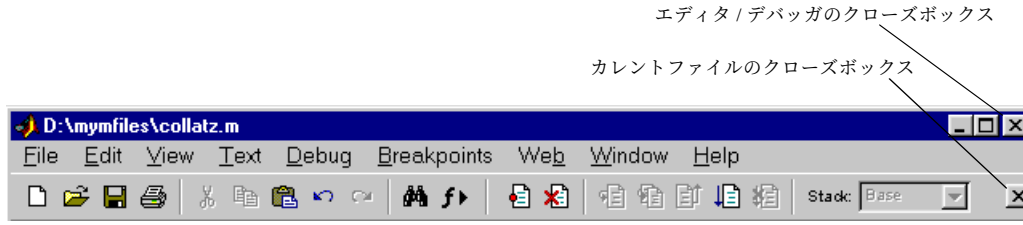
M- ファイルの印刷

M- ファイルの全体を印刷するには、**File** メニューから **Print** を選択するか、ツールバーのプリントボタン をクリックします。カレントで選択した対象を印刷するには、**File** メニューから **Print Selection** を選択します。その後、表示される標準の印刷ダイアログボックスで印刷を完了してください。

カラー印刷、または、白黒印刷、及び他の印刷オプションを指定するには、7-41 ページの " エディタ / デバッガの印刷設定 " を参照してください。

M- ファイルのクローズ

カレントの M- ファイルをクローズするには、**ファイル (File)** メニューから **Close filename** を選択してください。単一のエディタ / デバッガウィンドウ上で複数のファイルがオープンしている場合、エディタツールバーのクローズボックスをクリックすると、カレントの M- ファイルがクローズします。このクローズボックスは、エディタ / デバッガのタイトルバーにあるクローズボックスとは異なります。タイトルバーにあるクローズボックスをクリックすると、エディタ / デバッガをオープンしている全てのファイルと共にクローズしてしまいます。



それぞれのファイルが個々別途のエディタ / デバッガウィンドウでオープンしている場合、**Window** メニューから **Close All** を選択すると全てのファイルをクローズできます。

カレントファイルをクローズしたときに、クローズしたファイルが開かれている唯一のファイルであった場合、エディタ / デバッガも同時にクローズします。

変更をセーブしていないファイルをクローズする場合、ファイルをすぐにセーブしてください。

M- ファイルのデバッグ

この節では、エラーを検出するための一般的なテクニックを紹介します。そして、つぎに、エディタ / デバッガ、及び、デバッガ関数に備わっている MATLAB のデバッガ機能を単純な例題を用いて紹介します。この節は、つぎのトピックスから構成されます。

- 7-16 ページの " エラーのタイプ "
- 7-16 ページの " エラーの検出 "
- 7-17 ページの " デバッグの例題 – Collatz 問題 "
- 7-19 ページの " 例題の試行 "
- 7-21 ページの " デバッグ機能の使用 "

エラーのタイプ

*デバッグ*は、ユーザ自身が作成したコードの中に隠されている問題を検出して、修正するプロセスです。デバッグは、つぎの 2 タイプのエラーの修正で有効です。

- 文法的なエラー – たとえば、関数名のスペルミスや括弧を忘れたというようなエラーです。MATLAB は、文法的なエラーのほとんどを検出し、コマンドウィンドウにエラーメッセージを出力します。このエラーメッセージには、エラーの説明と、エラーが含まれている行番号を表示します。
- 実行時のエラー – これらのエラーは、通常、アルゴリズムに起因しています。たとえば、間違った変数を修正したり、または、正しくない計算を行うようなことです。実行時のエラーは、M- ファイルが予期しないような結果を出力するときに判明します。

エラーの検出

通常、文法的なエラーについては、MATLAB のエラーメッセージをベースに容易に検出できます。しかし、実行時のエラーについては、エラーが生じたときに、関数のローカルワークスペースが失われ、MATLAB ベースワークスペースに出力するため、検出はより困難なものとなります。実行時のエラーを起こしている原因を分離するには、つぎのいずれかのテクニックを使用します。

- ユーザの M- ファイルを構成するステートメントからある部分を選択して、ラインの最後のセミコロンを削除します。これにより、M- ファイルの中の中間的な計算結果が表示されます。セミコロンを除去すれば、M- ファイルの実行に合わせて、中間的な計算結果をスクリーン上に表示することができます。

- M- ファイルに、keyboard ステートメントを追加します。Keyboard ステートメントは、そのステートメントを追加した個所で、M- ファイルの実行を停止し、実行している関数のローカルワークスペースの調査、変更を可能にするステートメントです。このモードに入っているかどうかは、特別なプロンプト “K>>” が表示されているかどうかによって知ることが出来ます。Keyboard ステートメントによる停止状態から関数の実行状態へと復帰するには、return とタイプし、**Return** キーを押してください。
- Function M- ファイルの宣言を削除し、スクリプトとして M- ファイルを実行します。これにより、ベースワークスペース上で中間の計算結果にアクセスできるようになります。
- MATLAB エディタ / デバッガ、または、デバッグ関数を使用します。これらのツールは、関数のワークスペースにアクセスしたり、ワークスペース内部の値を調査、または、変更できるようにすることを通じて、実行時のエラーの修正を容易にします。これらのツールを使って、M- ファイルの中に、途中で実行を停止する*ブレークポイント*を設定、または、解除することができます。また、ワークスペースの内容の変更、関数コールのスタック表示、M- ファイルの内容を一行ずつ実行する、といった操作も可能となります。

この節の残りの部分では、エディタ / デバッガ及びデバッグ関数を、どのように使用するかに関する解説を例題を用いて行います。

デバッグの例題 – Collatz 問題

例題となるデバッグセッションの実行には、2 つの M- ファイル ,collatz.m、及び、collatzplot.m の作成が必要となります。この 2 つの M- ファイルは、collatz 問題に関するデータを生成するファイルです。ある与えられた正の整数 n について、Collatz 関数は、常に 1 に戻る数列を生成します。n が偶数の場合、数列でつぎに来る整数値は、n 番目の整数値を 2 で割った値となります。これに対し、n が奇数の場合、数列でつぎに来る整数値は、n 番目の整数値を 3 倍して、1 を加えた値となります。このような計算プロセスを数列で、つぎの整数値が 1 になるまで繰り返します。数列を構成する整数の数は、数列の開始値 n の値によって決定されます。

Collatz 問題は、この Collatz 関数が全ての正の整数について、1 に還元していくことを証明するものです。このデバッグ例題で用いられる M- ファイルは、この Collatz 問題を研究する上で役立ちます。ファイル collatz.m は、ある与えられた正の整数 n について、整数列を生成します。一方、ファイル collatzplot.m は、ある与えられた正の整数 n について、数列を構成する整数の数を計算し、計算結果をプロットします。出力されるプロット表示は、より一歩進んだ研究の対象となりうるパターンを示すことになります。

n が、1、2、または、3 の場合、計算結果は、つぎのとおりです。

n	数列	数列を構成する整数の数
1	1	1
2	2 1	2
3	3 10 5 16 8 4 2 1	8

Collatz 問題用の M- ファイル

つぎの 2 つの M- ファイルは。デバッグの例題で使用するものです。ユーザのシステムにこれらのファイルを作成するには､2 つの新しい M- ファイルをオープンします。ヘルプブラウザからつぎのコードを選択し、コピーして、M- ファイルにペーストします。そして、ファイル名 collatz.m と collatzplot.m として保存します。それらを保存するとき、ユーザのカレントディレクトリであるか、または、保存したディレクトリを、ユーザのサーチパスに付加していることを確認してください。ファイルの中の一つが、デバッグ用にエラーを組み込んでいます。

collatz.m のコード

```
function sequence=collatz(n)
% Collatz 問題。整数数列は、1 に収束するように作成
% 任意の正の整数 n に対して
%   n が偶数の場合、2 で割り、
%   n が奇数の場合、3 を掛け、1 を加えます。
%   結果が 1 になるまで、繰り返します。
sequence = n;
next_value = n;
while next_value > 1
    if rem(next_value,2)==0
        next_value = next_value/2;
    else
        next_value = 3*next_value+1;
    end
    sequence = [sequence, next_value];
end
```

collatzplot.m のコード

```
function collatzplot(n)
```

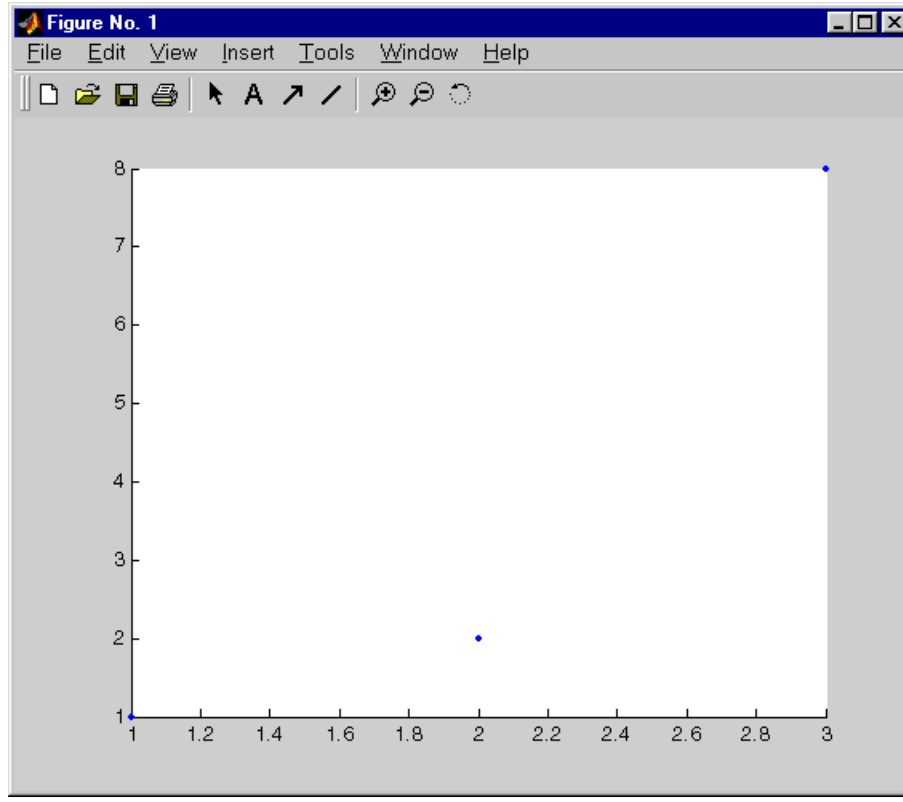
```
% Collatz 問題の数列長をプロット
% フィギュアを用意
clf
set(gcf,'DoubleBuffer','on')
set(gca,'XScale','linear')
%
% 数列長を計算し、プロットします。
for m = 1:n
    plot_seq = collatz(m);
    seq_length(m) = length(plot_seq);
    line(m,plot_seq,'Marker','.','MarkerSize',9,'Color','blue')
    drawnow
end
```

例題の試行

正確に機能するか否かをチェックするために、collatzplot を試してください。単純な入力値 3 を使ってみます。そして、前に記述したテーブルの結果と比較してみましょう。つぎのように入力してください。

```
collatzplot(3)
```

つぎに示す結果が、プロットされます。



n= 1 のとき、Collatz は、1 で、整数は 1 つで、正しい結果を示しているように思えます。n = 2 と n = 3 に対して、各整数に対して、数列の長さを示す 2 と 8 を示す値が 1 つのみしか出力されていないので、間違っています。代わりに、複数の値がプロットされなければなりません。問題と離れて、MATLAB のデバッグを使ってみましょう。

デバッグ機能の使用

ユーザは、エディタ / デバッガ及びデバッグ関数を使って、M- ファイルのデバッグを行うことができます。エディタ / デバッガの使用法、及びデバッグ関数の使用法、これら 2 つの方法を、交互に使用することも可能です。例題は、この 2 つの方法、双方について解説を行っています。

デバッグ機能は、つぎのとおりです。

- 7-21 ページの " デバッグの準備 "
- 7-21 ページの " ブレークポイントの設定 "
- 7-23 ページの "M- ファイルの実行 "
- 7-24 ページの "M- ファイルをライン毎に実行 "
- 7-25 ページの " 変数の値のチェック "
- 7-28 ページの " 問題の修正とデバッグの終了 "

デバッグの準備

デバッグを準備するために、以下の操作を行います。

- ファイルを開く – エディタ / デバッガを使って、デバッグを行うには、実行するファイル、すなわち、この例では、collatzplot.m でエディタ / デバッガを開いてください。
- 変更のセーブ – ファイルをエディットした場合、デバッグを始める前に変更をセーブしてください。変更をセーブしていないファイルをデバッグする場合、それによって得られる結果は、信頼性の低いものとなってしまいます。
- ファイルをサーチパスに付加 – エディタ / デバッガ、または、デバッグ関数のいずれを使用する場合でも、実行予定のファイル、及び、そのファイルが呼び出す全てのファイルがサーチパス上にあることを確認してください。デバッグで使用する全てのファイルが同一フォルダにある場合、サーチパスに付加する代わりに、そのフォルダをカレントディレクトリに移しても構いません。

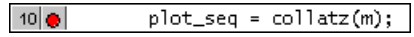
ブレークポイントの設定

問題があると思われる変数値を分析するために、関数の実行を停止する breakpo ブレークポイントを設定します。ブレークポイントは、カレントディレクトリ、または、サーチパス上にあるファイルの実行可能なラインに設定することができます。M- ファイルが実行されているなど、MATLAB がビジー状態にある場合、ブレークポイントを設定することはできません。

例題にブレークポイントを設定．例題では、問題がファイル collatzplot、または、collatz のいずれに存在するのかが明確になっていません。そこで、最初にファイル collatzplot.m のライン 10, 11, 12 にブレークポイントを設定してみましょう。ライン 10 のブレークポイントは、問題がもう一方のファイル collatz にあるのかどうかを確認するために、対象ファイルに割り込むことを可能にするためのものです。なお、ライン 11, 12 のブレークポイントは、計算の中間結果を調査できるように、プログラムの実行を停止する目的で設定しています。

エディタ / デバッガを使ってブレークポイントの設定．エディタ / デバッガを使って、ブレークポイントを設定するには、ライン番号の右側に表示されるブレークポイント欄の - 印をクリックしてください。- 印は、対象ライン上にブレークポイントを設定することが可能であることを示しています。その他にも、設定したいラインにカーソルを置き、ツールバーのブレークポイントの設定 / 消去ボタン をクリックするか、**Breakpoints** メニュー、または、コンテキストメニューから **Set/Clear Breakpoint** を選択しても可能です。

ブレークポイントアイコンが表示されます。下図では、ライン 10 に、ブレークポイントアイコンが設定されています。



等価な機能をもつ関数．デバッグ関数を使って、ブレークポイントを設定するには、関数 dbstop を使用します。たとえば、つぎのようにタイプします。

```
dbstop in collatzplot at 10
dbstop in collatzplot at 11
dbstop in collatzplot at 12
```

また、デバッグに有用である関連関数も用意しています。

- dbtype – コマンドウィンドウに M- ファイルを行番号付きでリスト表示します。
- dbstatus – ブレークポイントをリスト表示します。

実行を停止する条件の設定．Breakpoints メニューから、つぎのいずれかを選択するか、dbstop 関数で同様の設定を行うことにより、ある問題に出会ったときプログラムの実行を停止するよう設定することができます。

- **Stop If Error**, または、dbstop if error
- **Stop If Warning**, または、dbstop if warning
- **Stop If NaN Or Inf**(not-a-number または不定値に対して), または、dbstop if naninf、または、dbstop if infnan

M- ファイルの実行

ブレークポイントを設定した後、コマンドウィンドウ、または、エディタ / デバッガから M- ファイルを実行します。

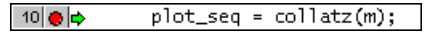
たとえば、単一の入力値 3 について、collatzplot を実行するには、コマンドウィンドウで、つぎのようにタイプします。

```
collatzplot(3)
```

Running an M-File from the Editor/Debugger. ツールバーの実行ボタン をクリックすることにより、入力引数を必要としない M- ファイルをエディタ / デバッガから実行することもできます。**Debug** メニューから、**Run** または **Save and Run**(変更をセーブしていないファイルの場合) を選択することによっても実行できます。例題 collatzplot は、入力引数が必要で、そのためコマンドウィンドウからのみ実行され、エディタ / デバッガからは実行できません。

M- ファイルを実行すると、つぎの結果が表れます。

- コマンドウィンドウのプロンプトが、K>> に変化し、MATLAB がデバッグモードに入ったことを表示します。
- プログラムの実行は最初のブレークポイント (例題ではライン 10 に設定) で停止します。このことは、ユーザが続ければライン 10 が実行されることを意味しています。プログラムの実行が停止状態にあることは、エディタ / デバッガのブレークポイント欄の右側にグリーンの矢印 (下図参照) が表示されることによって示されます。



Command Window debugging の Debugger options preference にチェックを入れた状態で、デバッグ関数を使用すると、実行を停止しているラインがコマンドウィンドウ上に表示されます。たとえば、つぎのように表示されるでしょう。

```
10   plot_seq = collatz(m);
```

- ツールバーの **Stack** フィールドに表示される関数は、カレントで実行している関数によって変化します。デバッグ関数を使用する場合、カレントのコールスタックを表示するには、関数 dbstack を利用して下さい。コールスタックには、呼び出されている関数は、もちろんのこと、サブ関数も表示されます。
- カレントディレクトリを、実行中のファイルを含むディレクトリに変更してください。

M- ファイルをライン毎に実行

デバッグモードでは、M- ファイルをライン単位で実行し、変数値を調べたい位置で停止することができます。

これは、エディタ / デバッガの**デバッグ (Debug)** メニューの中のステップボタン、または、ステップアイテムを使用するか、等価な機能をもつ関数を使用します。

ツールバーボタン	デバッグメニューアイテム	詳細	等価な関数
	Continue	M- ファイルの最後、または、他のブレークポイントに出会うかのいずれかになるまで、M- ファイルの実行を続けます。ファイルがまだ実行されていない場合、メニューアイテムは、Run または Save and Run が表示されます。	dbcont
なし	Go Until Cursor	カーソルが位置するラインまで、M- ファイルの実行を続けます。コンテキストメニューからも選択できます。	なし
	Step	M- ファイルのカレントラインを実行します。	dbstep
	Step In	M- ファイルのカレントラインを実行し、実行したラインで他の関数が呼び出されていれば、その関数に割り込みます。	dbstep in
	Step Out	割り込みをした後、呼び出した関数、または、サブ関数の残りの部分を実行し、その関数の実行から抜けて、実行を停止します。	dbstep out

割り込みを行なう . 例題では、collatzplot は、ライン 10 で実行を停止します。このとき、割り込みボタン、または、コマンドウィンドウで dbstep in とタイプすると、ライン 10 で呼び出されるファイル collatz に割り込みを行い、M- ファイ

ル相互で移動しながら実行することが可能となります。この操作では、ファイル collatz のライン 9 に割り込みを行います。

この時、collatzplot が、ライン 10 で、実行停止状態にあることを示す印が、⇨に変化し、MATLAB の制御が、メインプログラムではなく、メインプログラムから呼び出された関数 (例題では、collatz) に移っていることを表示します。

呼び出された関数においても、ユーザは、メイン関数同様の操作 (ブレークポイントの設定、実行、ライン単位での実行、変数値の調査) を実行することができます。

割り込み状態から抜ける . 例題では、collatz のステップ 9 でプログラムの実行は停止します。問題の解は、n = 1 については正しいので、n = 2 になるまで変数の値を調べる必要がないので、プログラムの実行を継続します。collatz の実行を最も速く終了する方法は、割り込んだ関数の実行から抜ける操作を行うことです。この操作により、collatz の残りのラインを実行し、元のプログラムである collatzplot のライン 11 に復帰します。割り込んだ関数を抜けるには、ステップアウトボタンを使用するか、コマンドウィンドウで、dbstep out とタイプしてください。

変数の値のチェック

プログラムの実行を停止させれば、ワークスペースにカレントで記憶されている変数値を表示させることが可能となります。変数の値をチェックするには、つぎの方法を用いてください。

- 7-25 ページの " 変数の値をどこでチェックするか "
- 7-26 ページの " ワークスペースの選択 "
- 7-26 ページの " エディタ / デバッガ上にデータテップを表示 "
- 7-27 ページの " コマンドウィンドウ上で変数値の表示 "
- 7-27 ページの " 配列エディタ上に変数値の表示 "
- 7-27 ページの " 選択したものの値 "

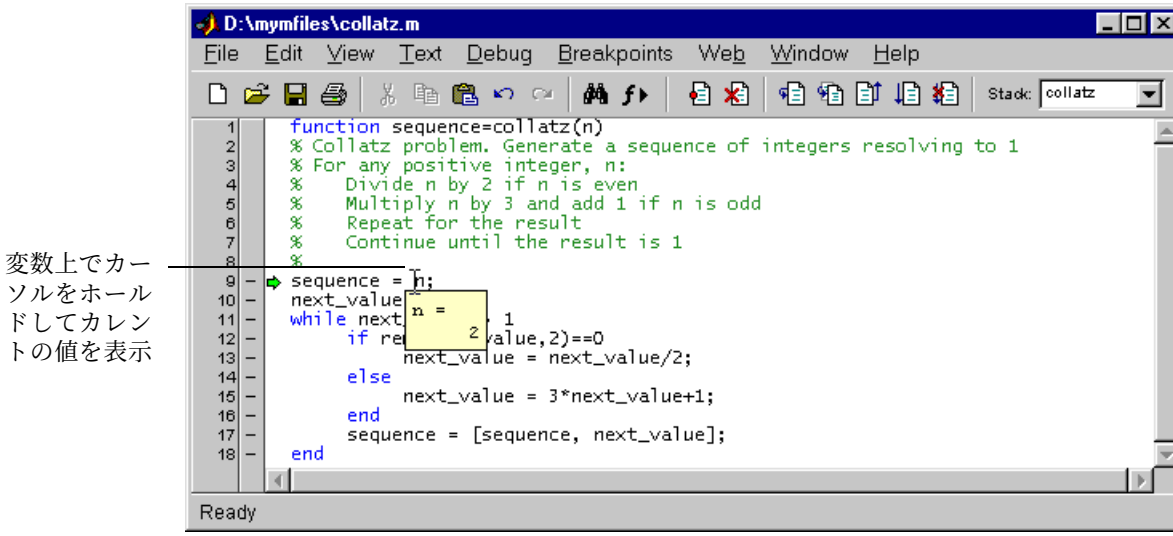
上記の方法のほとんどが、7-28 ページの " 例題の中で変数の値を調べる " で使用されています。

変数の値をどこでチェックするか . プログラムの実行を、ブレークポイント、または、ライン単位で停止させれば、変数の値を分析することが可能となります。コードを構成するある特定のラインが期待した結果を出力しているか確認したい場合、変数の値を調べてください。調べた結果が、期待通りのものとなっていれば、実行を続けるか、または、つぎのラインに実行を進めてください。逆に、調べた結果が期待通りでない場合、対象ライン、または、その前のラインにエラーが隠されているということになります。

たとえば、n = 1 の結果が正しければ、n = 2 まで値をチェックする必要はありません。そのため、m = 1 での collatzplot を構成するループの最初のイタレーションでは、実行を続けてください。ループの2回目の繰り返し (m = 2) で、collatzplot をライン 10 上で停止させれば、変数の値を調べることが可能となります。

ワークスペースの選択． コマンドウィンドウによって割り当てられた変数が、ベースワークスペースとなると考えてください。これに対して、それぞれの関数で生成された変数は、その変数自身のワークスペースをもつことになります。変数の値を分析するために、ユーザは先ずワークスペースを選択しなければなりません。プログラムを実行すると、カレントワークスペースが Stack フィールドに表示されます。カレントで実行されている他の関数のワークスペースを構成する変数値を調べる場合、先ず、Stack フィールドのリストから対象となるワークスペースの選択を行ってください。

エディタ / デバッガ上にデータテップを表示． エディタ / デバッガ上でデータテップを表示させるには、カーソルを変数の左側に置いてください。カーソルを移動させるまで、カレントの変数値が表示されます。- これをデータテップ (datatip) と言います。下図の例では、ファイル collatz の n にカーソルが合わされ、データテップは、n = 2 であると表示しています。この結果から、変数の値が期待した通りの値になっていることが確認できます。なお、Stack フィールドが、カレント関数として、collatz の名を表示していることに注意してください。



コマンドウィンドウ上で変数値の表示．コマンドウィンドウ上で変数の名前をタイプすると、MATLAB は、その変数のカレント値を表示します。ワークスペースにカレントで存在する変数を確認するには、関数 who を使用してください。たとえば、n の値を表示させるには、つぎのようにタイプします。

```
n
```

このとき、MATLAB は予想通り

```
n =
     2
```

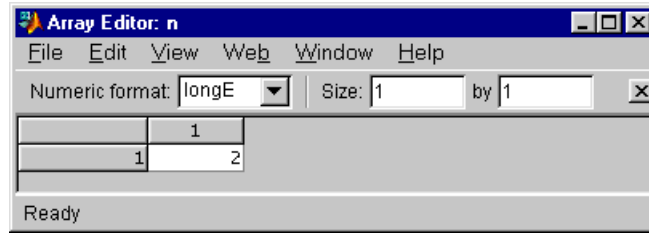
を出力します。

配列エディタ上に変数値の表示．配列エディタ上で変数値を表示することもできます。カレントの変数を表示するには、ワークスペースブラウザを使用します。ワークスペースブラウザ上で、変数をダブルクリックすると、配列エディタがオープンし、選択した変数の値を表示します。ある特定の変数について、配列エディタをオープンするには、関数 openvar を使用します。

たとえば、配列エディタで、n の値を見るには、つぎのようにタイプしてください。

```
openvar n
```

この操作により、配列エディタがオープンし、n = 2 という予想通りの結果となっていることが確認できます。



選択したものの値．選択した変数の値を調査するエディタ / デバッガ上で M- ファイルを構成する変数、または、数式を選択してください。選択したら、右クリックを行い、表示されたコンテキストメニューから**選択したものの値 (Evaluate Selection)** を選択します。これにより、MATLAB は選択した変数、または、数式の値をコマンドウィンドウ上に表示します。ただし、M- ファイル実行時など、MATLABがビジー状態にある場合は、選択した変数の値を調べることはできません。

例題の中で変数の値を調べる． collatz の実行中に、ステップボタン、または、関数 dbstep を使用します。それにより、プログラムはライン 10 まで実行されますが、ここではまだ変数の値をチェックする必要がありません。そのため、プログラムの実行をライン 13 まで進めることにします。

ここで、再びプログラムの実行を進めると、ポーズインジケータは予想通り、if ループ直後のライン 17 までジャンプします。これは、ライン 13 のコードに next_value = 2 という値が受け渡されたからです。ここで、さらにプログラムの実行を進めると、ライン 17 における変数 sequence の値をチェックすることが可能となり、その値が、n = 2 の時に期待された 2 1 という値になっていることが確認できます。そこで、さらにライン 18 からライン 11 へと実行を進めると、ここでは、next_value が 1 となっているため、ループは終了します。この場合、ポーズインジケータはライン 10 にグリーンの下矢印で表示されます。グリーンの下矢印は、呼び出された関数の実行が終了し、プログラムの制御がメインプログラムに復帰したことを示しています。この場合では、collatzplot のライン 10 に復帰します。

collatzplot の実行をライン単位でさらに進めていきましょう。ライン 11、12 まで進めると、変数 sequence_length は要素 1 2 からなるベクトルとなり、正しい値になっていることが確認できます。

最後に、ライン 13 までライン単位で実行を進めていきます。ライン 12 の変数値をチェックすると、m = 2 は予想通りとなっていますが、二番目の変数は 2 つの値をとり、そのうちの 1 つの値しか予想通りになっていないことがわかります。ライン 13 では、二番目の変数である plot_seq は、予想通りの値 2 1 になっていますが、変数 plot_seq はプロットすべき変数ではありません。プロット表示の対象となるのは、変数 seq_length です。

問題の修正とデバッグの終了

問題を修正し、デバッグを終了するには、つぎの操作を行います。

- 7-28 ページの " 変数の変更と結果のチェック "
- 7-29 ページの " デバッグの終了 "
- 7-29 ページの " ブレークポイントの消去 "
- 7-30 ページの "M- ファイルの修正 "

上記の操作のほとんどが、7-30 ページの " 例題を終了 " で実際に使用されています。

変数の変更と結果のチェック． デバッグ中に、新しい値が予想通りの結果を生成するかどうか確認するために、カレントのワークスペースの中で、変数の値を変更することも可能です。プログラムが停止している間、コマンドウィンドウ、または、配列エディタ上で変数に新しい値を割り当ててください。その後、プログラ

ムの実行、または、ライン単位での実行を続ければ、予想通りの結果が生成されるかどうか確認できます。また、変更後の値が予想通りの結果を生成しない場合、プログラムの中にそれ以外の問題、または、さらなる新しい問題が隠されていると考えてください。

デバッグの終了． 問題を特定したら、デバッグセッションを終了してください。問題を修正するために、M- ファイルを変更したい場合、MATLAB で他の関数を実行したい場合、必ずデバッグセッションを終了しなくてはなりません。

注意 M- ファイルをエディットする前に常にデバッグモードを終了してください。デバッグモードで M- ファイルをエディットする場合は、ファイルの実行時に予期しない結果になる場合があります。

デバッグを終了するには、exit debug mode アイコン をクリックするか、**Debug** メニューから **Exit Debug Mode** を選択してください。

また、デバッグを終了する関数は、dbquit です。

デバッグを終了すると、エディタ / デバッガのポーズインジケータが表示されなくなります。そして、コマンドウィンドウ上のプロンプトも、デバッグプロンプト K>> ではなく、通常のプロンプト >> に戻ります。また、この状態からコールスタックにアクセスすることはできません。

ブレークポイントの消去． ブレークポイントは、ユーザが消去するか自動的に消去されるまで、ファイルの中に保持されます。ブレークポイントは、つぎの操作によって自動的に消去されます。

- MATLAB セッションの終了
- 関数 clear を使って、M- ファイルの消去
- ライン番号を変更するエディットをファイルに加える
- デバッグモード中にファイルをエディットする (この場合、常にブレークポイントが消去されるわけではありません)。

問題を特定し、修正した後のように、プログラムを中断せずに実行したい場合、ブレークポイントを消去します。エディタ / デバッガ上でブレークポイントを消去するには、ラインに表示されるブレークポイントアイコンをクリックするか、**Breakpoints** または、コンテキストメニューから **Set/Clear Breakpoint** を選択します。これにより、当該ラインのブレークポイントは消去されます。

ファイルに含まれる全てのブレークポイントを消去するには、**Breakpoints** メニューから、**Clear All Breakpoints** を選択するか、ツールバーのボタン をクリックしてください。

また、ブレークポイントを消去する関数は、dbclear です。たとえば、ファイル collatzplot に含まれる全てのブレークポイントを消去するには、つぎのようにタイプします。

```
dbclear all in collatzplot
```

M- ファイルの修正． M- ファイルに隠されている問題を修正するには、

1 デバッグを終了します。

DMATLAB がデバッグモードにある間は、M- ファイルを変更しないでください。デバッグモード中に M- ファイルを変更すると、ファイルに含まれている全てのブレークポイントが消去され、予期せぬ結果になります。

2 ファイル内のすべてのブレークポイントを消去します。

M- ファイルが変更されるとブレークポイントは適切なものではなくなってしまいます。ブレークポイントは、ファイルの実行時に予期しない結果になることがあります。

3 M- ファイルを変更します。

4 M- ファイルを保存します。

5 希望する場合がブレークポイントを設定します。

6 予想する結果を出力するように M- ファイルを再度実行します。

例題を終了． 例題の問題を修正するには、以下を行います。

1 デバッグセッションを終了します。これを行う 1 つの方法は、**Debug** メニューから **Exit Debug Mode** を選択することです。

2 collatzplot.m のブレークポイントを消去します。これを行うためには、コマンドウィンドウで

```
dbclear all in collatzplot
```

と入力します。

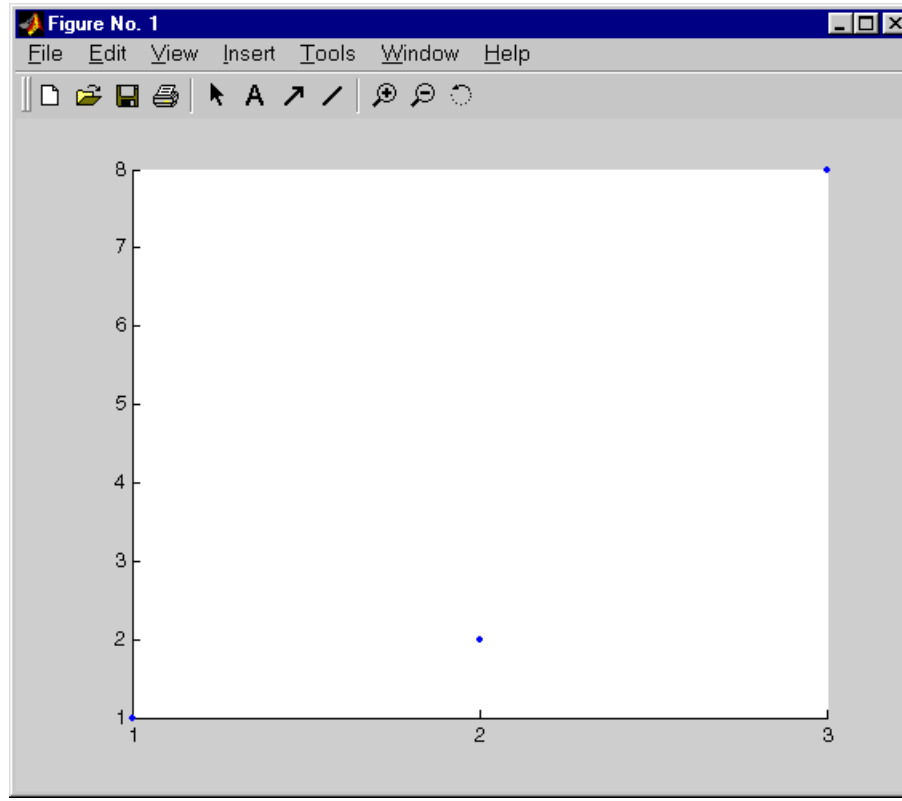
3 collatzplot.mのライン11で、文字列plot_seqをseq_length(m)に変更し、ファイルを保存します。

4 以下のように

```
collatzplot(3)
```

コマンドウィンドウに入力して n = 3 に collatzplot ついてを実行します。

5 結果を検証します。下図は、Collatz 数列の長さが n = 1 に対しては 1、n = 2 に対しては 2、n = 3 に対しては 8 と予想通りであることを示します。



6 n を 3 よりも、やや大きい値 (たとえば、6) で、もう一度関数のテストを行い、正しい出力結果が出ていることを確認します。n = 6 に対する collatzplot の結

果を、collatz の結果と共に検証する作業を、より簡単なものにするために、collatz.m の最終ラインに、つぎの 1 行を加えてください。

```
sequence
```

これにより、コマンドウィンドウに数列が表示されます。

その後、つぎのようにタイプして、n = 6 について、collatzplot を実行します。

```
collatzplot(6)
```

7 デバッグを簡単にするために、小さい n の値で、collatzplot を実行してきました。ここでは、このファイルが正しい計算結果を出力することがわかっているので、より大きな値で、collatzplot を実行し、より興味深い結果を出力してみましょう。この操作を行う前に、ステップ 7 で追加したライン (collatz.m

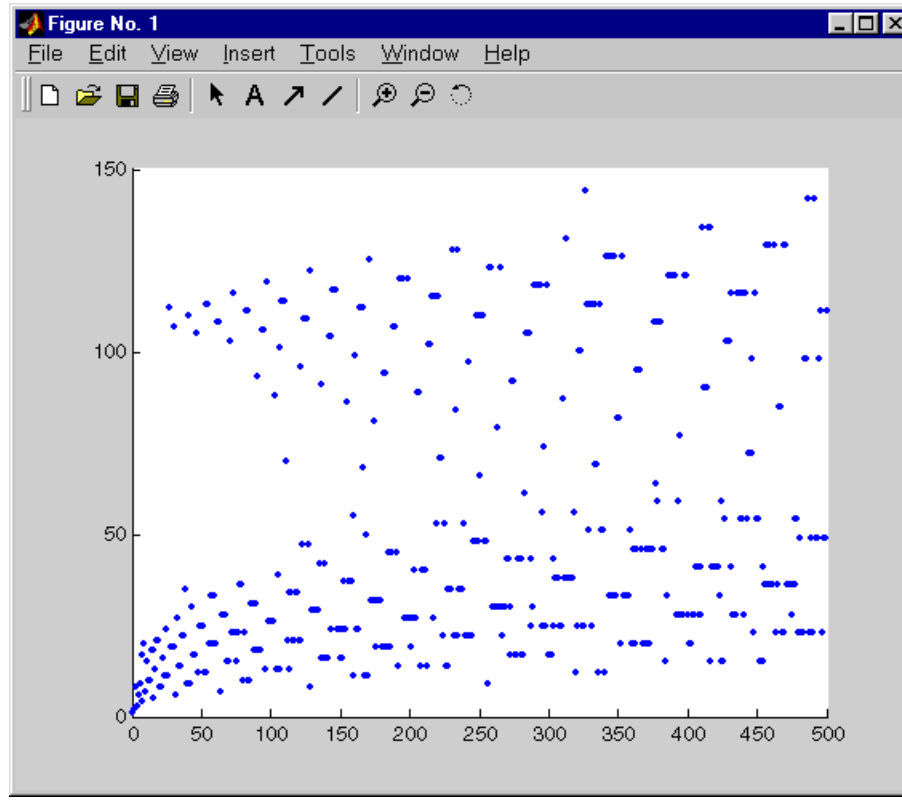
のライン 19) に対する出力結果を省略します。その場合、つぎのように、ラインの最後にセミコロンを追加してください。

```
sequence;
```

その後、ファイルを実行します。

```
collatzplot(500)
```

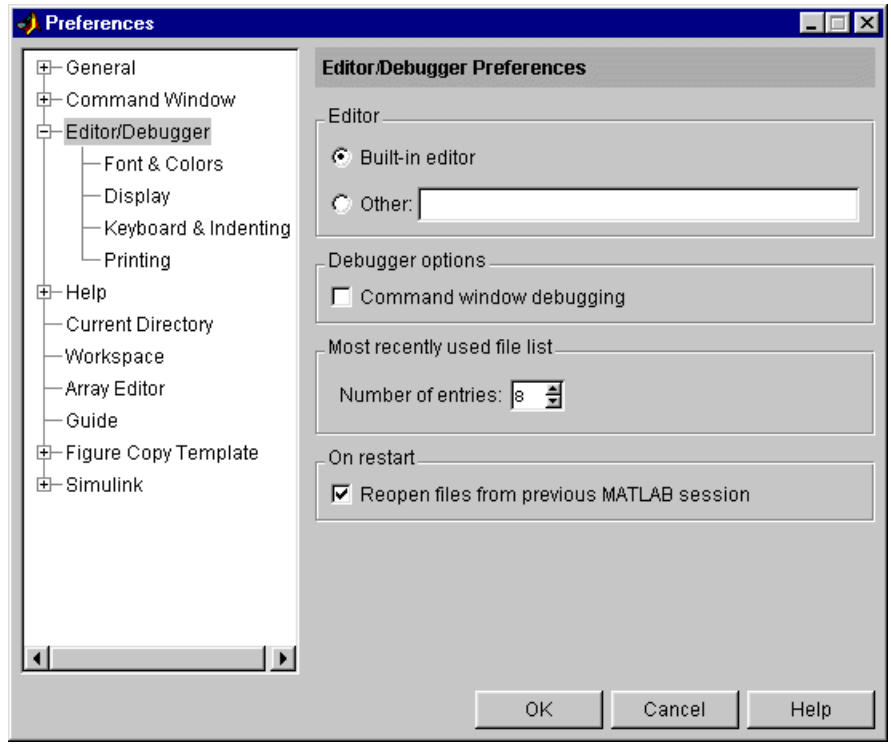
つぎの図は、n = 1 から = 500 における Collatz 数列の長さを示しています。
n = 500



エディタ / デバッガの設定

エディタ / デバッガの設定 (ユーザ設定) を使って、エディタ / デバッガの様々な状況におけるデフォルト挙動を指定できます。

エディタ / デバッガの設定は、エディタ / デバッガの**ファイル (File)** メニューから、**設定 (Preferences)** を選択します。すると**設定 (Preferences)** ダイアログボックスが開かれ、**エディタ / デバッガ設定 (Editor/Debugger Preferences)** が表示されます。



ユーザは、つぎのエディタ / デバッガに関する設定を変更できます。

- 7-35ページの"エディタ/デバッガの一般設定"(最初のパネルに表示される設定で、**Editor** もここに組み込まれています)
- 7-36 ページの " エディタ / デバッガで使用するフォントやカラーの設定 "
- 7-37 ページの " エディタ / デバッガの表示設定 "
- 7-39 ページの " エディタ / デバッガのキーボード / インデント設定 "
- 7-41 ページの " エディタ / デバッガの印刷設定 "

エディタ / デバッガの一般設定

エディタ / デバッガの設定にアクセスすると、最初に先ずこの節で説明される一般設定 (General Preference) を指定することができます。

エディタ

MATLAB は、デフォルトでは、**組み込みエディタ (Built-in editor)** オプションが選択され、MATLAB に組み込まれたエディタを使用するよう設定されています。

MATLAB 内部から M- ファイルを開いた場合、MATLAB にデフォルトで用意されたエディタではなく、他の (Emacs、または、vi のような) 外部エディタを使用するよう設定するには、**Other** を選択します。そして、**Other** フィールドに、使用したいエディタアプリケーションのタイプ、及び、パスを入力します。

たとえば、C:\Applications\Emacs.exe と **Other** フィールドに入力し、その後、MATLAB デスクトップの**ファイル (File)** メニューから、**Open** を選択し、ファイルを開きます。この時、ファイルは、MATLAB に組み込まれたエディタ / デバッガではなく、Emacs 上で開かれます。

デバッガオプション

デフォルトでは、アイテム **Command Window debugging** にチェックは入っていません。この状態では、ブレークポイントを含む M- ファイルを実行すると、ブレークポイントが設定されたラインが実行されると同時に、MATLAB エディタ / デバッガが起動します。

デバッグ関数を使用する場合には、アイテム **Command Window debugging** にチェックを入れて、ブレークポイントが設定されたラインが実行されても、MATLAB エディタ / デバッガが起動しないよう設定しておく必要があります。

最近使用されたファイルリスト

この設定によって、**ファイル (File)** メニューに表示される最近使用したファイルのリストに記載されるファイルの数を自由に指定することができます。

リスタートに関して

MATLAB を起動し、前回、MATLAB をシャットダウンしたときに開いていたファイルを自動的にオープンする、という操作を行うには、**Reopen files from previous MATLAB session** をチェックしてください。このアイテムをチェックせずに、MATLAB を終了すると、次回、MATLAB を起動したときに、エディタ / デバッガは、スタートアップ時に起動しません。

エディタ / デバッガで使用するフォントやカラーの設定

Font & Colors 設定を用いると、エディタ / デバッガで使用するフォントやカラーを指定することができます。

フォント

エディタ / デバッガのフォント設定では、エディタ / デバッガ上でオープンしたファイルで使用するフォントの特性を指定します。**General - Font & Colors** で指定したフォントと同一のフォントをファイルでも使用したい場合、**Use Desktop Font** を選択してください。デスクトップと同一のフォントを使用したくない場合、**Use Custom Font** を選択し、使用するフォントの特性を指定してください。

- タイプ、例、Lucida Console
- スタイル、例、Plain
- ポイント単位のサイズ、例、12 ポイント

選択すると、**Sample** エリアで、どのようにフォントが表示されるかを確認することができます。

カラー

エディタ / デバッガ上で開いたファイルで、使用するカラーを指定します。

- **テキストカラー** – 強調しない通常のテキストを表示するカラーです。強調表示するテキストのカラーは、**Syntax highlighting** で指定します。
- **バックグラウンドカラー** – ウィンドウ内のバックグラウンドのカラー
- **強調表示シンタックス** – シンタックスを強調する際に使用するカラーです。ここに、チェックを入れた場合、**Set Colors** で強調色を指定してください。詳細

は、3-5 ページの " シンタックスの強調 " を参照してください。

エディタ / デバッガの表示設定

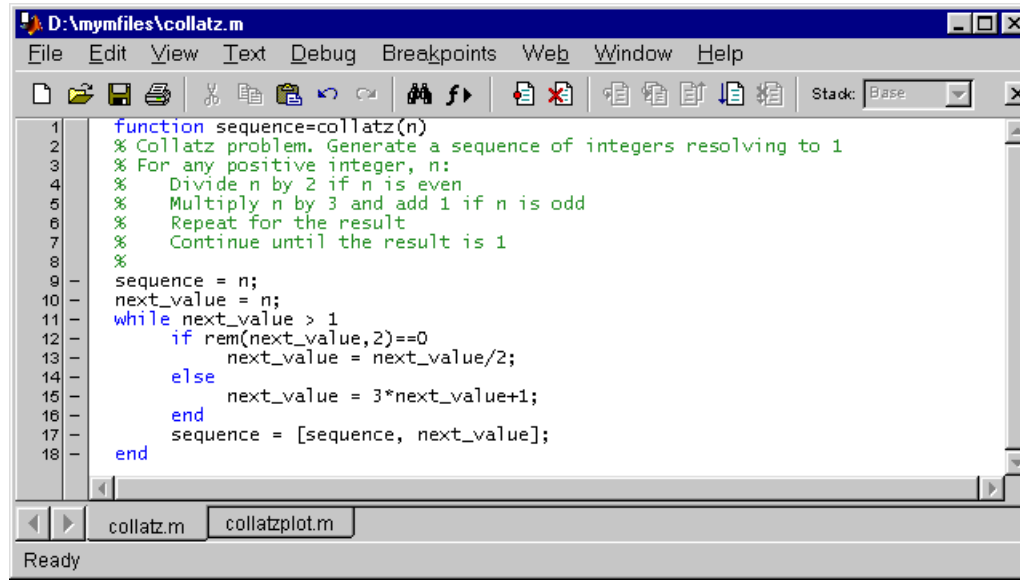
表示 (Display) 設定を使用して、エディタ / デバッガウィンドウを、どのように表示するかを指定することができます。

Opening files in editor

この設定は、エディタ / デバッガでファイルをオープンした場合、オープンしたファイルをどのように配置するかを制御します。この設定を変更すると、設定の変更は、変更を行った後にオープンしたファイルから適用されます。カレントでオープンしているファイルについては、設定を変更しても再配置されません。

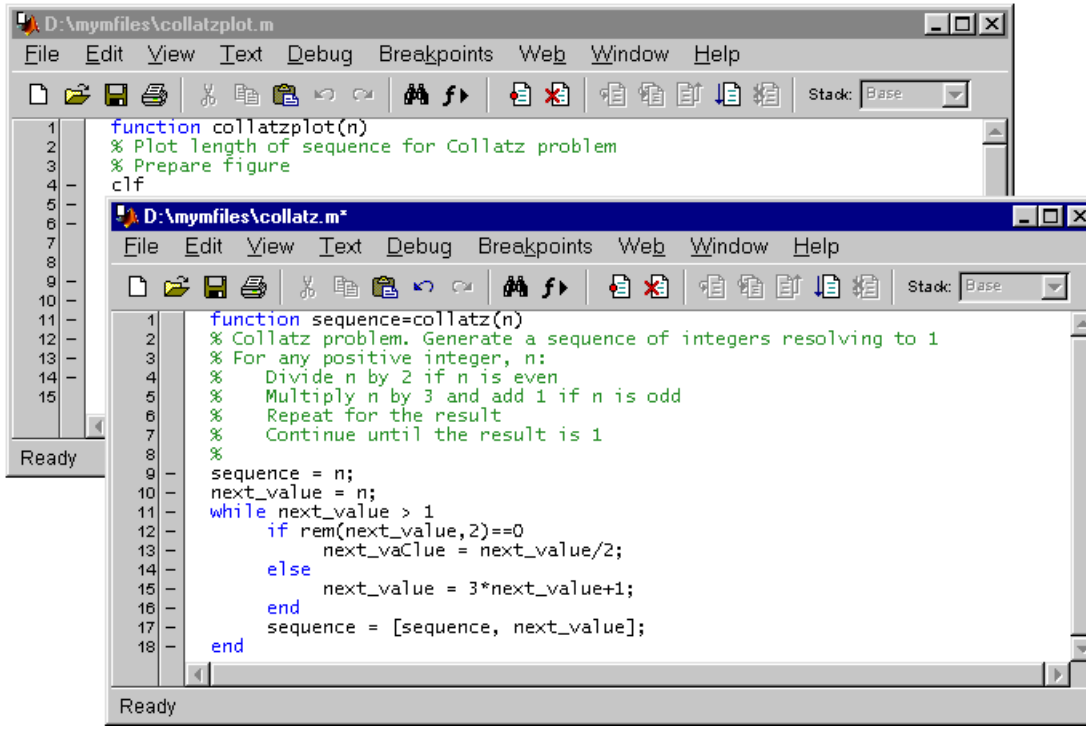
Single window contains all files(tabbed style) をチェックすると、つぎの図のように、オープンした全てのファイルが、単一のエディタ / デバッガウィンドウ上に表示されます。オープンしているファイルの中のいずれかのファイルをカレントファイルに選択するには、そのファイルのタブをクリックしてください。

Files are tabbed within one



Each file is displayed in its own window をチェックすると、オープンしたそれぞれのファイルが、別個のエディタ / デバッガウィンドウに表示されます。この状態を例示したのが、下の図です。

Each file is in its own



Display

Display オプションを使用すると、エディタ / デバッガ上に何が表示され、何が非表示となるのかについての指定を行うことができます。

Show toolbar. このアイテムをチェックするとツールバーが表示されます。チェックをはずすと、ツールバーは非表示となります。

Show line numbers. このアイテムをチェックすると行番号が表示されます。行番号はウィンドウの左側に表示されます。このアイテムのチェックをはずすと、行番号は表示されなくなります。

Enable datatips in edit mode. このアイテムをチェックすると、データテップがエディットモード中に表示できるようになります。データテップは、デバッグモード中は常に表示されます。

エディタ / デバッガのキーボード / インデント設定

キーボード設定によって、MATLAB が従うキーバインディングを指定することができます。さらに、インデント設定を使って、エディタ / デバッガ上で、ラインをどのようにインデントするかを指定することもできます。

キーバインディング

ここでは、エディタ/デバッガがアクセラレータやショートカットの際に従うキーバインディングを **Windows** または **Emacs** のいずれかに選択します。このオプションを変更した後に、メニューに表示されるアクセラレータも変更されます。

たとえば、Windows のキー入力規則を選択すると、ペーストのショートカットは、**Ctrl+V** になります。一方、Emacs のキー入力規則を選択すると、ペーストのショートカットは **Ctrl+Y** となります。アクセラレータについては、**Edit** メニューの **Paste** アイテムで確認できます。

M-file indenting for Enter key

ここでは、エディタ / デバッガ上で、**Enter** キーを押したときに適用されるインデントのスタイルを指定します。インデントの例は、つぎのとおりです。

- **No indent** – ラインに字下げ処理を行いません。ラインを左揃えにしたいとき、または、字下げ処理を手動で行いたいときに、このスタイルを選択します。
- **Block indent** – ラインを上のラインと同じ幅だけ字下げします。
- **Smart indent** – キーワード関数で始まるライン、または、あるキーワード関数に続くラインを自動的に字下げします。このようなスマートインデント処理を施すことにより、コード配列を目で追っていく作業が簡単なものになります。

インデント処理のスタイルは、設定を変更した後に入力したラインに対してのみ有効となります。既存のラインは、インデント設定の変更の影響を受けません。既存のラインに対するインデント処理を変更するには、**テキスト (Text)** メニューの 7-8 ページの " インデント処理 " を使用してください。

インデント処理のいずれかのスタイルを適用した場合、ユーザは、ラインの冒頭にタブを手動で挿入することができます。

例：タブを挿入していない No Indent の例題 .

```
sequence = n
next_value = n;
while next_value > 1
if rem(next_value,2)==0
next_value = next_value/2;
else
next_value = 3*next_value+1;
end
sequence = [sequence, next_value]
end
```

Created using **No indent** preference.

例：タブを用いた No Indent の例題 .

```
sequence = n
next_value = n;
while next_value > 1
    if rem(next_value,2)==0
    next_value = next_value/2;
    else
    next_value = 3*next_value+1;
    end
    sequence = [sequence, next_value]
end
```

Created using **No indent** preference.
Created indentation by manually inserting a tab before each indented

例：Block Indent の例題 .

```
sequence = n
next_value = n;
while next_value > 1
    if rem(next_value,2)==0
    next_value = next_value/2;
    else
    next_value = 3*next_value+1;
    end
    sequence = [sequence, next_value]
    end
```

Created using **Block indent** preference.
Inserted a tab before the if statement.
Subsequent lines automatically

例：Smart Indent　の例題 .

```
sequence = n
next_value = n;
while next_value > 1
    if rem(next_value,2)==0
        next_value = next_value/2;
    else
        next_value = 3*next_value+1;
    end
    sequence = [sequence, next_value]
end
```

Created using **Smart indent** preference.
Did not manually insert any tabs.

Indent

Indent size. スマートインデント処理に適用するインデントサイズ (字下げ幅) を指定します。

Emacs-style tab key smart indenting. このインデント規則は、Emacs エディタで用いられているスタイルに基づいています。このスタイルを選択すると、どのラインにも自動的なインデント処理が施されなくなります。スマートインデント処理によって、ラインをインデント処理するには、当該ラインにカーソルを合わせるか、または、インデント処理したい複数のラインを選択し、**Tab** キーを押してください。

Tab

Tab size. Tab キーを押したときに挿入されるスペースの数を指定します。**タブサイズ (Tab size)** を変更すると、対象ファイルを構成する既存のラインについても、スペースの数が変更されます。

Tab key inserts spaces. Tab キーを押したときに一連のスペースを挿入したい場合、このアイテムにチェックを入れてください。このアイテムにチェックが入ってない場合、タブによって、**タブサイズ (Tab size)** で選択した 1 つのスペースのみ挿入されます。

エディタ / デバッガの印刷設定

印刷設定を使って、M- ファイルをどのように印刷するかを指定することができます。

強調表示されたシンタックス

この設定により、強調表示されたシンタックスをどのように印刷するかを指定できます。指定できるオプションは、つぎのとおりです。

- **Print as black and white text**
- **Print as colored text**
- **Print as styled text** – 白黒で印刷します。コメントはイタリックで、キーワードはボールドで印刷されます。

ヘッダの設定 (Header preference)

印刷したページのヘッダに、ファイルのフルパス名、ページ数、印刷を行った日付と時間をリスト表示したい場合、**Print header** にチェックしてください。

8

M- ファイルのパフォーマンスを改善 – プロファイラ

プロファイリングとは？ 8-3

プロファイラの機能 8-4
profile コマンド 8-4
例題：プロファイラの使用 8-6

プロファイラの結果表示 8-7
プロファイルレポートの表示 8-7
プロファイル結果のプロット 8-11
プロファイルレポートの保存 8-12

ユーザが作成したM- ファイルのパフォーマンスを改善する1つの方法は、M- ファイルをプロファイルすることです。MATLAB は、M- ファイルを構成する各々のラインの計算にどれだけの時間が必要となるのかを計測するM-ファイルプロファイラを用意しています。

この節では、以下のトピックスを説明します。

- 8-3 ページの " プロファイリングとは ?"
- "profile コマンド " と " 例題:プロファイラの使用 " を含む 8-4 ページの " プロファイラの機能 "
- 8-7 ページの " プロファイラの結果表示 "

プロファイリングとは？

*プロファイリング*は、プログラムが、どのポイントで時間を費やしているのかを知る手段です。計測することは、実行時間を最も消費している場所はどこなのかを推定する、より良い方法です。設計時にスピードを優先してプログラムを組んでいても、実際に計測すると予期しない効果が発見されることがあるからです。効率良くコードを作成するための１つのポイントは、出来る限りシンプルに原型となるインプリメンテーションを作成し、その後で、プロファイラを使って高速化で障害となる部分を見つけだすという手順でコードを作成することです。早すぎる段階で最適化を行うと、コードが不必要に複雑になるにも関わらず、性能はほとんど改善されていない結果になることがしばしば生じます。

プロファイラを使って、最も多くの時間を消費する関数を識別し、その後、その関数をコールしている理由を考え、その使用を最小化する方法を考えます。特別な関数をコールする回数が適切か否かを決定する手助けになります。プログラムが、複雑な構造に分かれているために、最も時間を消費する関数を明示的にコールしているかを見ることは困難です。むしろ、ユーザのコードの中に含まれている関数が、他の関数を呼び、コードの中のいくつかの層で、時間を消費する関数をコールしている可能性もあります。この場合、ユーザが使っているどの関数が時間を消費する関数をコールしているのかを確定することが、重要なことになります。

プロファイラは、つぎの事柄を解明することで、問題を解く手助けになります。

- 見落としから生ずる不必要な計算の回避
- ユーザのプログラムを変更して、実行に時間が必要な関数を回避
- 将来の使用のために、結果を保存することで、再計算の回避

時間の大部分を消費する部分が、数個の組み込み関数からなっている場合、期待したとおりのコードの最適化が既に行われていると考えて良いでしょう。

プロファイラの機能

profile コマンドを使って、統計量を作成し、参照することができます。

1 コマンドウィンドウで profile on と入力してプロファイラを起動します。利用したいオプションを指定してください。

2 M- ファイルを実行します。

プロファイラは、M- ファイルを構成する各ラインの実行に何秒消費しているのかをカウントします。統計量をクリアするまで、ユーザが実行する各 M- ファイルに関する量を累積します。

3 profile report を使って、デフォルトの Web ブラウザに、HTML 形式のレポートとして、統計結果を表示します。

profile コマンド

つぎに、profile コマンドの主な使用法をまとめてみます。これらに関する、また、他のオプションに関する詳細は、help profile、または、doc profile を参照してください。

シンタックス	オプション	詳細
profile **on**		以前に記録した統計量をクリアして、プロファイラをスタートします。
	-**detail** *level*	プロファイル対象となる関数のレベルを指定します。
	-**history**	関数の実行を正確な順序で、記録するように指定します。
profile **off**		プロファイラを停止します。

シンタックス	オプション	詳細 (Continued)
profile **report**		プロファイラを停止して、プロファイルレポートを HTML で作成、Web ブラウザに、そのレポートを表示します。
	basename	カレントディレクトリのファイル basename にレポートを保存します。
profile **plot**		プロファイラを停止して、フィギュアウィンドウに、実行時間を最も多く費やす関数を棒グラフで表示します。
profile **resume**		以前記録したカウント結果をクリアせずに、プロファイラをスタートします。
profile **clear**		プロファイラによって記録された統計量をクリアします。
s = profile('**status**')		カレントのプロファイラステータスを含む構造体を表示します。
stats = profile('**info**')		プロファイラを停止して、プロファイラ結果からなる構造体を表示します。

例題：プロファイラの使用

つぎの例題では、プロファイラを実行する方法をデモンストレートしています。

1 プロファイラを起動するため、コマンドウィンドウで以下を入力します。

```
profile on -detail builtin -history
```

-detail builtin オプションは、デフォルトの M- 関数、M- サブ関数、MEX- 関数のみならず、組み込み関数についても統計量を収集します。

-history オプションは、関数の呼び出し、終了を正確な順序で記録します。

2 M- ファイルを実行します。この例題では、Lotka-Volterra predator-prey population モデルを考えています。このモデルに関する詳細については、lotkademo と入力し、デモンストレーションを実行して見てください。

```
[t,y] = ode23('lotka',[0 2],[20;20]);
```

3 プロファイルレポートを作成し、その結果を、ファイル lotkaprof に保存します。

```
profile report lotkaprof
```

これによりプロファイラ作業を中断し、プロファイルレポートを Web ブラウザウィンドウに表示し、結果を保存します。詳細は、8-7 ページの " プロファイラの結果表示 " を参照してください。

4 前回のカウント結果をクリアせずに、プロファイラを再スタートします。

```
profile resume
```

この場合、プロファイラは、今後実行する M- ファイルについても統計を収集し続けます。そして、その結果作られた新しい統計結果は、前のステップで作成された統計結果に追加されます。

5 統計量の収集が完了したら、つぎのコマンドでプロファイラを停止します。

```
profile off
```

プロファイラの結果表示

プロファイラの結果を表示する主な方法は、2 つあります。

- 8-7 ページの " プロファイルレポートの表示 "
- 8-11 ページの " プロファイル結果のプロット "

結果の保存については、8-12 ページの " プロファイルレポートの保存 " を参照してください。

プロファイルレポートの表示

プロファイラの結果を表示するには、以下を入力します。

```
profile report
```

これは、プロファイラを停止し、3 つのレポートを生成します。

- 8-7 ページの " サマリープロファイルレポート "
- 8-9 ページの " 関数詳細プロファイルレポート "
- 8-10 ページの " 関数実行履歴プロファイルレポート "

サマリーレポートは、システムのデフォルト Web ブラウザに表れます。レポートページの一番上のリンクを使ってその他のレポートを参照します

サマリープロファイルレポート

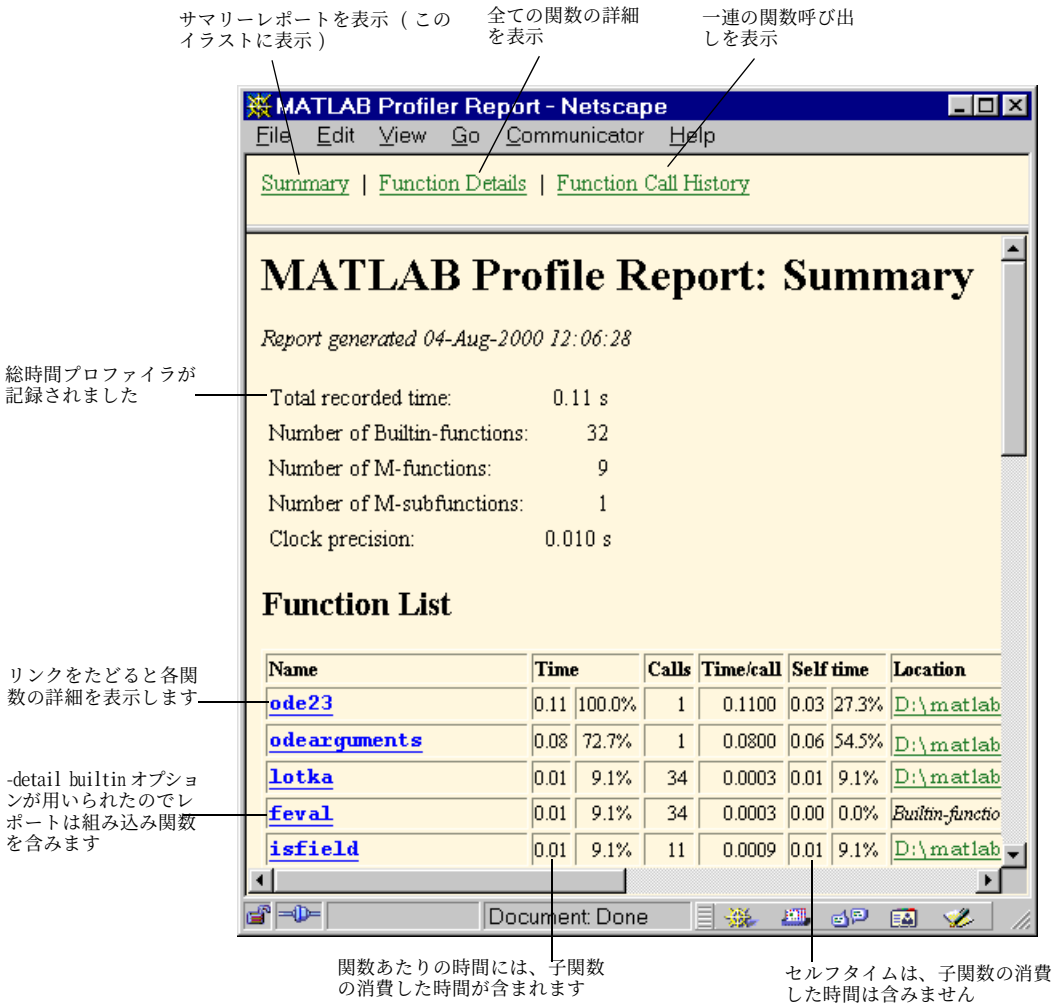
サマリーレポートは、M- ファイルの実行に関する総合的な結果と、コールした各関数の実行に関する統計量を提供します。これらには、つぎの値が含まれています。

- **関数の数** – 組み込み関数、M- ファンクション、M- サブファンクションの数がレポートされます。
- **時計の精度** – プロファイラが行う時間計測の精度です。関数に対する時間 (Time) が 0 の場合、実際の時間は正の値になりますが、設定した時間精度で検出できる値より小さいものになります。
- **時間**列 – 1 つの関数が消費する総時間です。この中には、コールされる全ての子の関数を実行するのに要する時間も含まれています。この値は、子の関数で消費する時間も含んでいるので、総時間を合計した**総記録時間 (Total recorded time)** の値と一致せず、割合としても 100% を超えるものになっています。
- **セルフタイム**列 – 関数の実行にかかった総時間で、子の関数の実行にかかった時間を含まずに表示します。すべての関数の**自身の消費時間 (セルフタイム)** の値を加算すると、**総記録時間 (Total recorded time)** になります。すべての

関数に対する**セルフタイム**の割合を加算したものは、ほぼ 100% になります。

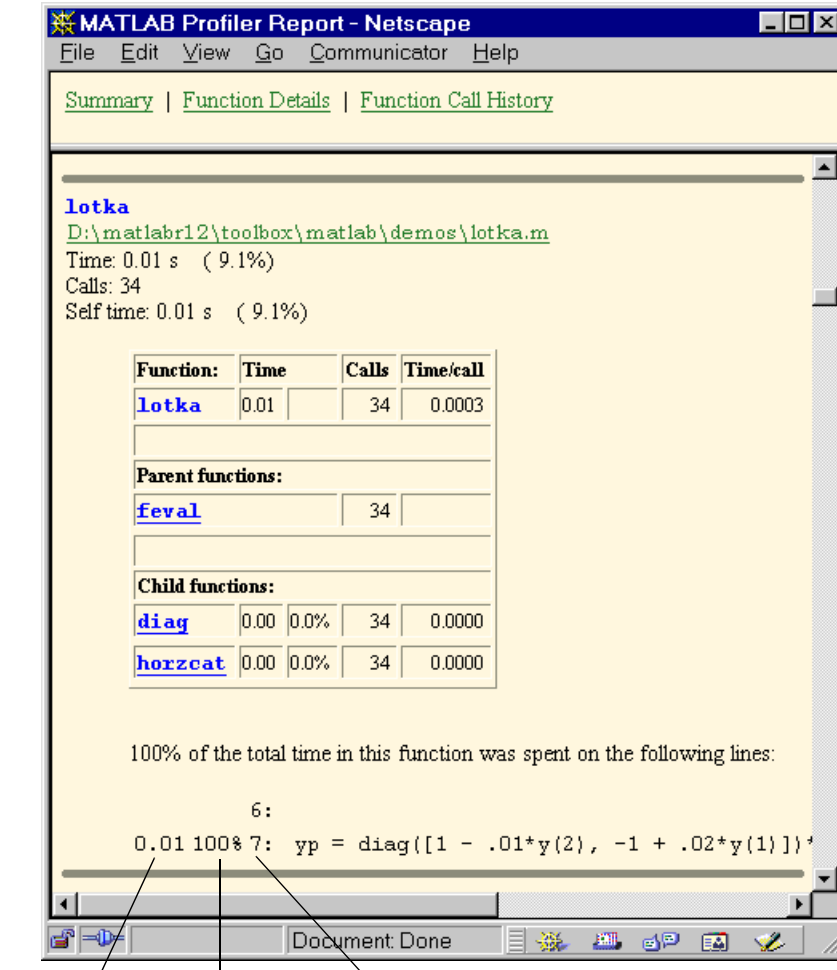
プロファイラ自身は、プロファイラの結果表示にのために時間を消費します。

以下は、8-6 ページの " 例題：プロファイラの使用 " で説明されている Lotka-Volterra モデルのサマリーレポートです。



関数詳細プロファイルレポート

関数詳細レポートは、関数の親関数と子関数に関する統計結果を提供し、最も多くの時間を費やしたラインの番号を表示します。下図は、8-6 ページの " 例題：プロファイラの使用 " で実行された関数の 1 つである関数 lotka に関する詳細レポートを示しています。



時間 (秒)

そのラインで関数が費やした時間の割合

ライン番号

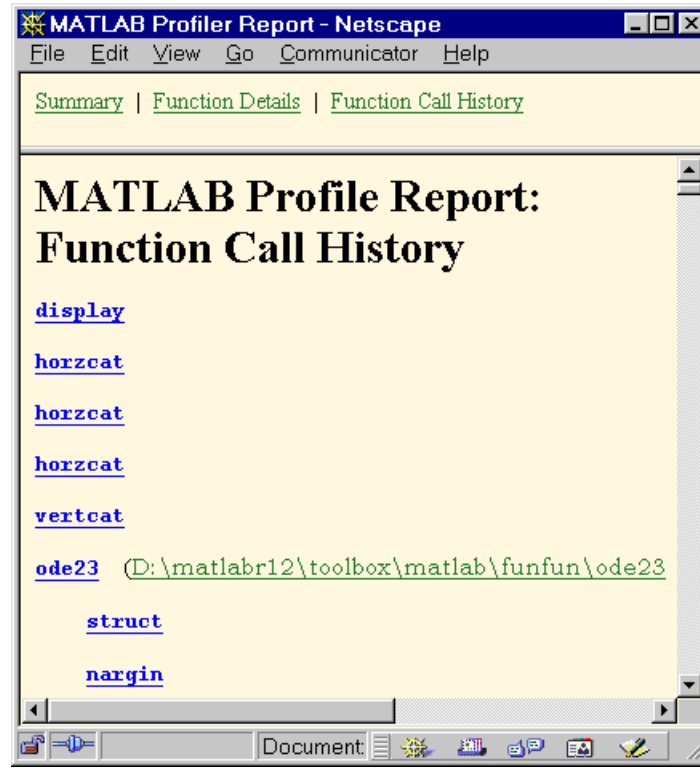
関数実行履歴プロファイルレポート

関数実行履歴レポートは、コールされた関数を正確な順序で表示します。このレポートを見るには、つぎのように、-history オプションを使って、プロファイルをスタートします。

```
profile on -history
```

プロファイラは、10,000 個まで、関数の実行、終了イベントを記録できます。イベントの数が 10,000 以上に達すると、プロファイラは他のプロファイル統計量に記録を続けますが、連続の順番は保たれません。下図は、8-6 ページの " 例題：プロファイラの使用 " で作成された履歴レポートです。

正確な実行順序

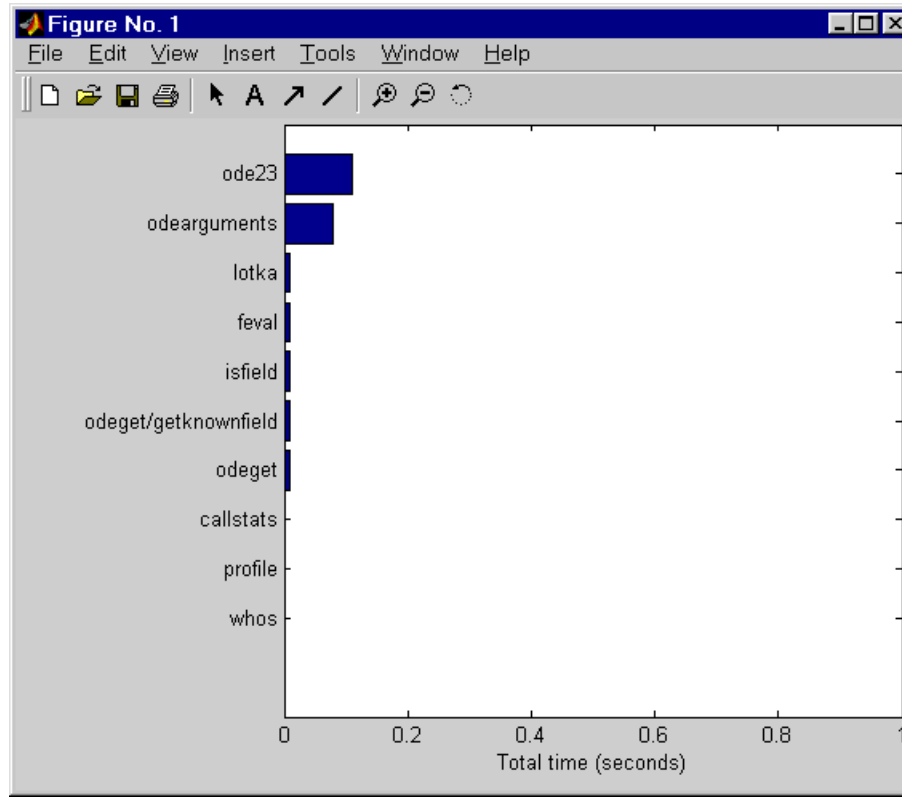


プロファイル結果のプロット

実行に最も多くの時間を消費した関数に対する棒グラフを見るには、つぎのように入力します。

```
profile plot
```

これは、プロファイラを停止します。figure ウィンドウに棒グラフを表示します。下図は、8-6 ページの " 例題 : プロファイラの使用 " で作成された棒グラフです。



プロファイルレポートの保存

プロファイルレポートの作成時に、オプションを使ってレポートを保存します。たとえば、

```
profile report basename
```

は、プロファイルレポートをカレントディレクトリのファイル basename に保存します。後で Web ブラウザを使って保存された結果を見ることができます。

結果を保存するもう一つの方法は、コマンド info = profile を使って、プロファイル結果を含む構造体を表示するものです。この構造体を保存して、関数 profreport(info) を使って、後で、見ることができます。

例題：プロファイラ結果の構造体の使用

ここでは、プロファイラ結果を後々参照、または、アクセスすることができる構造体に格納します。つぎの例題は、どのようにプロファイル結果を見るかを示しています。

1 Lotka-Volterra predator-prey populationモデルを計算するコードに対して、プロファイラを実行します。

```
profile on -detail builtin -history
[t,y] = ode23('lotka',[0 2],[20;20]);
```

2 プロファイラ結果を含む構造体を表示するために、つぎのようにタイプします。

```
stats = profile('info')
```

これにより、MATLAB は、つぎの結果を出力します。

```
stats =
      FunctionTable: [41x1 struct]
    FunctionHistory: [2x826 double]
     ClockPrecision: 0.0100
               Name: 'MATLAB'
```

3 ユーザは構造体の内容を参照、アクセスすることが可能になります。たとえば、つぎのようにタイプしてください。

```
stats.FunctionTable
```

これにより、MATLAB は、FunctionTable 構造体を表示します。

```
ans =
41x1 struct array with fields:
    FunctionName
    FileName
    Type
    NumCalls
    TotalTime
    TotalRecursiveTime
    Children
    Parents
    ExecutedLines
```

4 FunctionTable 構造体に含まれる要素の内容を参照するには、たとえば、つぎのようにタイプします。

```
stats.FunctionTable(2)
```

MATLAB は、構造体の 2 番目の要素を出力します。

```
ans =
          FunctionName: 'horzcat'
              FileName: ''
                  Type: 'Builtin-function'
              NumCalls: 43
             TotalTime: 0.0100
    TotalRecursiveTime: 0.0100
              Children: [0x1 struct]
               Parents: [2x1 struct]
         ExecutedLines: [0x3 double]
```

5 結果を保存してください。

```
save profstats
```

6 より後のセッションで、保存した結果を使って、プロファイラレポートを作成するには、つぎのようにタイプします。

```
load profstats
profreport(stats)
```

これにより、MATLAB はプロファイルレポートを表示します。

9

ソースコントロールシステムとのインタフェース

SCS とのインタフェースプロセス 9-3

ソースコントロールシステムの参照と選択 9-5
SCS の参照と等価な機能をもつ関数 9-5

ソースコントロールシステムの設定 9-6
SourceSafe のみ – MATLAB 階層のミラー化 9-6
UNIX での ClearCase のみ – ビューの設定とディレクトリのチェックアウト 9-6

プロジェクト設定ファイルの指定 – PVCS のみ 9-7

ソースコントロールシステムへのファイルのチェックイン . . 9-8
ファイルのチェックインと等価な機能をもつ関数 9-9

SCS からのファイルのチェックアウト 9-10
ファイルのチェックアウトと等価な機能をもつ関数 . . 9-11
チェックアウトを元に戻す 9-11

ソースコントロールシステム (SCS) を使ってファイルを管理したい時に、M- ファイル、Simulink、Stateflow のファイルを MATLAB からソースコントロールシステムにチェックインしたり、ソースコントロールシステムから MATLAB にチェックアウトしたりすることができます。

MATLAB, Simulink, tateflow はソースコントロール関数を実行しませんが、ソースコントロールシステムへのインタフェースのみを提供します。これは、たとえば、MATLAB エディタでファイルをオープンし、チェックアウトせずに修正することを意味します。しかし、ファイルは、リードオンリーのままなので、ファイルのソースコントロールバージョンを誤って上書きすることはありません。

4つのよく利用されるソースコントロールシステムがサポートされていますが、他のソースコントロールシステムをオプションで指定することも可能です。

- ClearCase from Rational Software
- PVCS from Merant
- RCS
- Visual SourceSafe from Microsoft
- Custom option – 他のソースコントロールシステムとのインタフェースの構築

SCS とのインタフェースプロセス

GUI で操作を行いたい時はメニューを使って、コマンドウィンドウでの操作を行いたい時は関数を使って、ソースコントロールシステムに接続できます。メニューによる操作では利用できないいくつかのオプション機能を関数からの操作で利用することができます。このようなオプション機能の詳細については、メニューを使った操作の説明を行っている箇所に記載されています。

M- ファイル、Simulink ファイルまたは Stateflow ファイルを通常の方法で作成し、ファイルを保存します。その後次のステップに従って、MATLAB、Simulink、Stateflow からユーザのソースコントロールシステムとのインタフェースをとります。

	ステップ	手順
1	使用するソースコントロールシステムを選択します。	9-5 ページの " ソースコントロールシステムの参照と選択 " を参照
2	ユーザのソースコントロールシステムがファイルを正しく取り込めるように、設定を行います。これは、いくつかのソースコントロールシステムで必要な操作です。	9-6 ページの "SourceSafe のみ – MATLAB 階層のミラー化 " を参照 9-6ページの"UNIX上のClearCaseのみ – ビューの設定とディレクトリのチェックアウト " を参照
3	PVCS の場合のみ、プロジェクト設定ファイルを、ファイル cmopts.m の中で指定します。	9-7ページの"プロジェクト設定ファイルの指定 – PVCS のみ " を参照
4	ファイルをソースコントロールシステムにチェックインします。一部のソースコントロールシステムでは、チェックイン操作の前にファイルをチェックアウトしておかなければなりません。	9-8ページの"ソースコントロールシステムへのファイルのチェックイン " を参照
5	次回ファイルを修正したい時には、MATLAB エディタ、Simulink または Stateflow でファイルを開き、そのファイルをチェックアウトします。	9-10ページの"SCSからのファイルのチェックアウト " を参照

	ステップ	手順（続き）
6	チェックアウト後の変更を反映させずにチェックイン状態に戻すためには、そのファイルのチェックアウトを元に戻す (Undo checkout) を選択します。	9-11 ページの " チェックアウトを元に戻す " を参照

ソースコントロールシステムの参照と選択

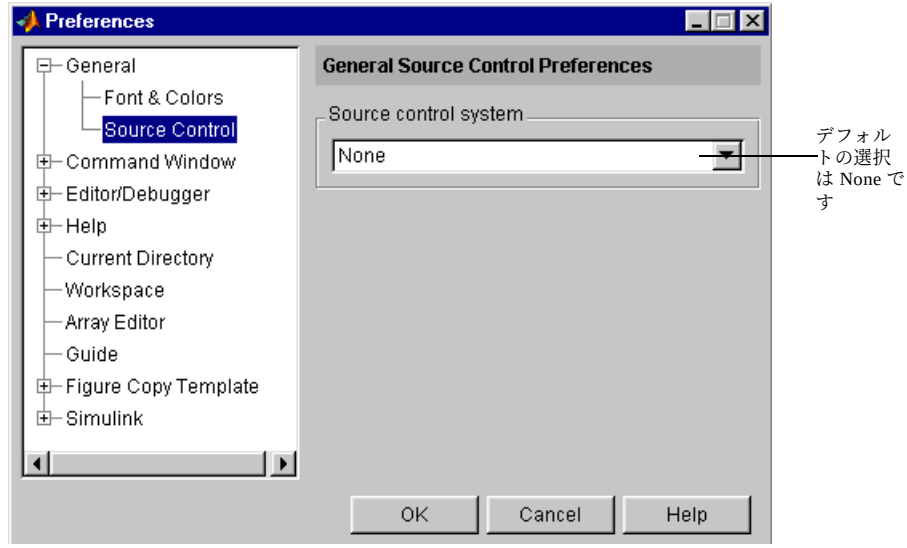
MATLAB とのインタフェースをとるソースコントロールシステムを指定します。

1 MATLAB デスクトップにおいて、**File** メニューから **Preferences** を選択します。Simulink または Stateflow モデル、あるいはライブラリウィンドウからも選択できます。

Preferences ダイアログボックスがオープンします。

2 **General** の + をクリックして、**Source Control** を選択します。

現在選択されているシステムを以下に示します。デフォルトの選択は None です。



3 **Source control system** リストから利用したいシステムを選択します。

4 **OK** をクリックします。

SCS の参照と等価な機能をもつ関数

現在選択されているシステムを参照するには、コマンドウィンドウで cmopts と入力します。MATLAB はカレントのソースコントロールシステムを表示します。

ソースコントロールシステムの設定

UNIX の ClearCase および SourceSafe に対して、ソースコントロールシステムを以下に示すように設定してください。

SourceSafe のみ – MATLAB 階層のミラー化

Visual SourceSafe を利用している場合は、MATLAB, Simulink, Stateflow ファイルのディレクトリの階層をミラーするように設定する必要があります。たとえば、MATLAB エディタを使って、D:\matlabr12\mymfiles に格納されたファイルについて SourceSafe とインタフェースをとるためには、SourceSafe プロジェクトを $:\matlabr12\mymfiles として設定します。

UNIX 上の ClearCase のみ – ビューの設定とディレクトリのチェックアウト

UNIX プラットフォーム上で ClearCase を利用している場合は、ClearCase を使って以下を行ってください。

1 ビューを設定します。

2 ファイルのセーブ、チェックイン、チェックアウトをしたいディレクトリをチェックアウトします。

MATLAB, Simulink, Stateflow と ClearCase とのインタフェースがとれ、ステップ 2 でチェックアウトしたディレクトリにファイルをチェックインしたり、そのディレクトリからファイルをチェックアウトすることができます。

プロジェクト設定ファイルの指定 – PVCS のみ

PVCS を使用している場合は、プロジェクト設定ファイル名を cmopts.m の中で指定しておく必要があります。プロジェクト設定ファイルは、PVCS 上でユーザが作成、利用するファイルですが、MATLAB もそのファイル名を知っておく必要があるからです。cmopts.m ファイルは、$matlabroot\toolbox\local に格納されています。ここで、$matlabroot は、MATLAB がインストールされているディレクトリを指しています。

MATLAB エディタまたは他のテキストエディタで、cmopts.m を開いてください。そして、% BEGIN CUSTOMIZATION SECTION で始まるセクションにプロジェクト設定ファイルの名前を指定します。ユーザのプロジェクトファイルの名前は、絶対パスも含む形で、変数 'DefaultConfigFile' に割り当てて下さい。以上の操作が終わったら、cmopts.m をセーブします。

たとえば、プロジェクト設定ファイルが Projmgr.cfg であったとします。このとき、cmopts.m につぎのようなラインを書き加えます。

```
DefaultConfigFile = 'c:\\PVCS\\PVCSPROJ\\Projmgr.prj\\Proj.cfg'
```

cmopts 関数に、'DefaultConfigFile' を引数として設定して実行すると、カレントのプロジェクト設定ファイルを参照することができます。

```
cmopts('DefaultConfigFile')
```

とタイプすると、MATLAB はつぎのように出力します。

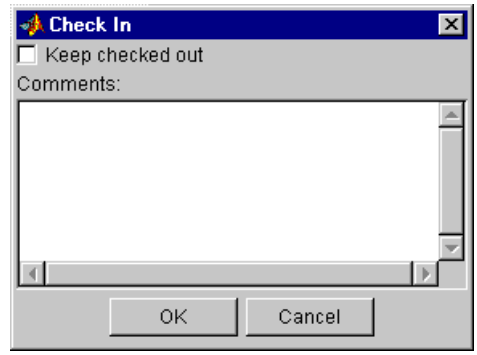
```
'c:\\PVCS\\PVCSPROJ\\Projmgr.prj\\Proj.cfg'
```

ソースコントロールシステムへのファイルのチェックイン

MATLAB エディタ、Simulink、Stateflow でファイルを作成、エディットし、セーブした後、そのファイルをチェックインするには、つぎの操作を行います。

1 MATLABエディタから、**ファイル(File) ->ソースコントロール(Source Control) -> チェックイン (Check In)** を選択します。Simulink または Stateflow のモデルウィンドウ、ライブラリウィンドウからもこの操作を行うことは可能です。

   Check in ダイアログボックスがオープンします。



2 変更作業を続行するために、ファイルをチェックアウトしたままでチェックインするためには、**チェックアウトを維持 (Keep checked out)** を選択します。

3 コメントを付ける時は、そのコメントを**コメント (Comments)** エリアに入力します。

   チェックアウトを維持 (Keep checked out) を選択したとしても、ユーザのコメントは削除されません。

4 OK をクリックします。

ファイルがソースコントロールシステムにチェックインされます。チェックインの前にファイルを保存しなかった場合は、チェックイン時に自動的に保存されます。

ファイルのチェックアウトを維持せずにファイルをオープンしたままの状態では、そのファイルは読み込み専用になっていることに注意してください。

PVCS を使用していて、'DefaultConfigFile' が定義されていないというエラーメッセージが出た場合、ユーザはプロジェクトファイルを定義できません。詳細は、9-7 ページの " プロジェクト設定ファイルの指定 – PVCS のみ " を参照してください。

ファイルのチェックインと等価な機能を持つ関数

ファイルをソースコントロールシステムにチェックインするには、checkin を利用します。ファイルは、checkin の実行時にオープンあるいはクローズできます。関数 checkin は以下の形式です。

```
checkin({'file1',...'filen'},'comments','string','option',...
'value')
```

ファイルに対しては絶対パスを使ってください。**comments** 引数およびコメント文字列には checkin を含めておかなければなりません。

option 引数は、以下のように使用します。

- ファイルをチェックアウト状態のままチェックインするためには、lock オプションを on に設定します。
- 前回チェックインしてから何ら変更されていない場合でも、そのファイルをチェックインするためには、force オプションを on に設定します。

comments 引数と、lock および force オプションは、チェックインされるすべてのファイルに適用されます。

チェックアウトの状態及び開いた状態にしていなかったファイルをチェックインする場合には、そのファイルが読み取り専用であることに注意してください。

例題 – コメント付きでのファイルのチェックイン

ファイル clock.m にコメント Adjustment for Y2K を付けてチェックインするには、つぎのようにタイプします。

```
checkin('\matlabr12\mymfiles\clock.m','comments','Adjustment ... for
Y2K')
```

他の例については、関数 checkin のリファレンスページを参照してください。

SCS からのファイルのチェックアウト

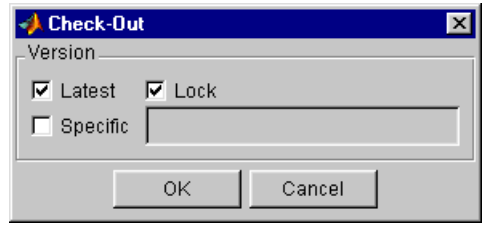
MATLABを使ってソースコントロールシステムからファイルをチェックアウトするには、以下のステップを行います。

1 チェックアウトしたい M- ファイル、Simulink ファイル、Stateflow ファイルをオープンします。

ファイルがオープンし、タイトルバーが読み込み専用であることを示します。

2 MATLAB エディタから**ファイル (File) -> ソースコントロール (Source Control) -> チェックアウト (Check Out)** を選択します。Simulink または Stateflow モデルあるいはライブラリウィンドウから選択することもできます。

Check-Out ダイアログボックスがオープンします。



3 チェックインされた最新のバージョンをチェックアウトするには、**Latest** を選択します。ファイルの固有のバージョンをチェックアウトするには、**Specific** を選択し、バージョン番号をフィールドに入力します。

最新バージョンをチェックアウトするときに、Lock オプションを利用することができます。チェックアウト中に他のファイルのチェックアウトを回避するには、**Lock** を選択します。読み込み専用のファイルをチェックアウトするには **Lock** のチェックを外します。

4 **OK** をクリックします。

ファイルはソースコントロールシステムからチェックアウトされ、編集が可能です。

PVCS を使用していて、'DefaultConfigFile' が未定義というエラーメッセージが出た場合は、プロジェクトファイルを定義できません。詳細は、9-7 ページの " プロジェクト設定ファイルの指定 – PVCS のみ " を参照してください。

ファイルのチェックアウトと等価な機能をもつ関数

ファイルをソースコントロールシステムからチェックアウトするには、checkout を使います。複数のファイルを一度にチェックアウトすることができ、チェックアウトオプションを指定することができます。関数はcheckoutは以下の形式です。

```
checkout({'file1',...'filen'},'option','value')
```

ファイルに対しては絶対パスを使ってください。

option 引数は以下のように使用します。

- 読み込み専用のファイルをチェックアウトするには、lock オプションを off に設定します。
- 既にチェックアウトしたファイルをチェックアウトするには、force オプションを on に設定します。
- ファイルの特定のバージョンをチェックアウトするには、revision オプションを使い、バージョン番号を value 引数に指定します。

オプションは、チェックアウトするすべてのファイルに適用されます。ファイルは、checkout の実行時にオープンあるいはクローズされます。

例 – 特定のバージョンのファイルをチェックアウト

バージョン1.1のclock.mをチェックアウトするにはつぎのようにタイプします。

```
checkout('\matlab\mymfiles\clock.m', 'revision', '1.1')
```

他の例については、関数 checkout のレファレンスページを参照してください。

チェックアウトを元に戻す

ファイルについてチェックアウトを元に戻すことができます。ファイルは、チェックインされた状態になり、ファイルをチェックアウトした後に加えた変更は全て反映されなくなります。変更のローカルコピーを保持したい場合は、**ファイル (File)** メニューから**名前を付けて保存 (Save As)** を利用します。

MATLAB エディタから、**ファイル (File) -> ソースコントロール (Source Control) -> チェックアウトを元に戻す (Undo Check Out)** を選択します。Simulink または Stateflow モデル、あるいはライブラリウィンドウからも選択できます。

チェックアウトを元に戻すことと等価な機能をもつ関数

関数 undocheckout は以下の形式です。

```
undocheckout({'file1',...'filen'})
```

ファイルに対しては絶対パスを使ってください。

例題 – 2つのファイルのチェックアウトを元に戻す． clock.m と calendar.m のチェックアウトを元に戻すには、以下のように入力します。

```
undocheckout({'\matlab\mymfiles\clock.m', '\matlab\mymfiles\calendar.m'})
```

10

ノートブックの使用

ノートブックの基本機能 10-3
M- ブックの作成 10-3
M- ブックに MATLAB コマンドを入力 10-6
ユーザのワークスペースを完全な状態で保護 10-6
データの一貫性の確保 10-7

MATLAB コマンドを入力セルとして定義 10-8
セルグループの定義 10-8
自動実行型セルの定義 10-10
Calc Zones の定義 10-10
入力セルをテキストに変換 10-11

MATLAB コマンドの実行 10-12
セルグループの計算 10-13
ある範囲の入力セルの計算 10-14
Calc Zone の計算の実行 10-14
M- ブック全体の計算の実行 10-15
ループを使って入力セルの繰り返し計算の実行 . . . 10-15
出力セルをテキストに変換 10-16
出力セルを削除 10-17

M- ブックの印刷とフォーマット 10-18
M- ブックの印刷 10-18
M- ブックテンプレートでのスタイルの変更 10-18
Loose、または、Compact フォーマットの選択 10-19
数値出力フォーマットの制御 10-20
グラフィック出力の制御 10-20

ノートブックの設定 10-24

ノートブックコマンドリファレンス 10-26

ノートブックを 使って、MATLAB の数値計算や可視化ソフトを文書作成環境 (Microsoft Word) からアクセスすることができます。ノートブックでは、テキスト、MATLAB コマンド、及び、その MATLAB コマンドからの出力を含む *M- ブック*と呼ばれる文書の作成を行うことができます。

M- ブックは、テキストによって注釈を付けた対話式の MATLAB セッションの記録、または、MATLAB コマンドとそれによる出力を含んだドキュメントのいずれかと考えることができます。MATLAB セッションの記録を電子的または印刷された形で作成するのに、または、クラスノート、テキストブック、テクニカルレポートを作成する上で、ノートブックは大変便利なものです。

この節では、つぎの情報を提供します。

- 10-3 ページの " ノートブックの基本機能 "
- 10-8 ページの "MATLAB コマンドを入力セルとして定義 "
- 10-12 ページの "MATLAB コマンドの実行 "
- 10-18 ページの "M- ブックの印刷とフォーマット "
- 10-24 ページの " ノートブックの設定 "
- 10-26 ページの " ノートブックコマンドリファレンス "

ノートブックの基本機能

この節では、ノートブックの基本機能を紹介します。説明される内容は、つぎのとおりです。

- 10-3 ページの "M- ブックの作成 "
- 10-6 ページの "M- ブックに MATLAB コマンドを入力 "
- 10-6 ページの " ユーザのワークスペースを完全な状態で保護 "
- 10-7 ページの " データの一貫性の確保 "

M- ブックの作成

この節では、つぎの点について説明を行います。

- MATLAB から M- ブックの作成法
- ノートブック実行中での M- ブックの作成法
- 既存の M- ブックのオープン方法
- Word ドキュメントから M- ブックへの変換

MATLAB から M- ブックの作成法

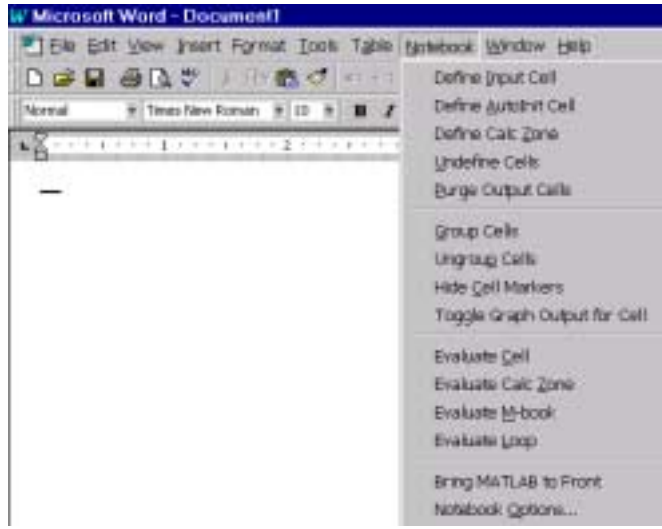
MATLAB で新しい M- ブックを作成するには、プロンプトで、つぎのようにタイプします。

```
notebook
```

最初にノートブックを実行する場合、ノートブックの設定を行う必要があります。詳細は、10-24 ページの " ノートブックの設定 " を参照してください。

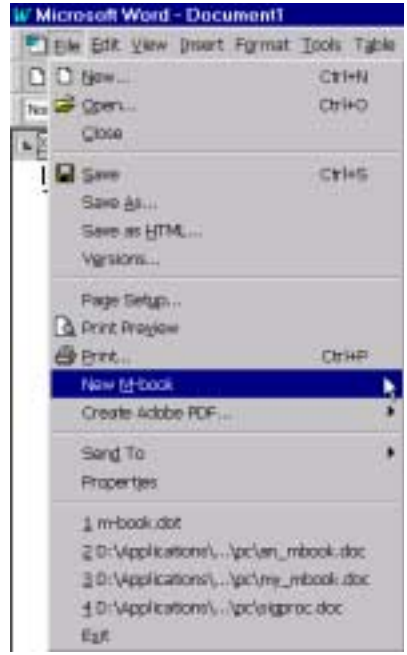
ノートブックは、Microsoft Word をユーザのシステムで起動し、Document1 という新しい文書を作成します。ノートブックは、Word のメニューバーに、**ノートブック (Notebook)** メニューを追加します。このメニューから、ノートブックのコマンドにアクセスすることができます。

注意 Notebook は、Microsoft Word マクロを定義し、MATLAB は、MATLAB コマンドとその出力が定義された様々なタイプのセルを解釈することが可能となります。



ノートブック実行中での M- ブックの作成法

ノートブックを実行しながら、Word の**ファイル (File)** メニューから**新規 M ブックの作成 (New M-book)** を選択すれば、新しいM-ブックを作成することができます。



既存の M- ブックのオープン方法

notebook コマンドを使えば、既存の M- ブックをオープンすることもできます。

```
notebook filename
```

ここで、filename は、オープンする M- ブックです。または、別の方法として、M- ブックファイルをダブルクリックする方法もあります。

M- ブックをダブルクリックすると、Microsoft Word は、M- ブックを開き、MATLAB がまだ起動されていなければ、MATLAB を起動します。ノートブックは、Word のメニューバーに**ノートブック (Notebook)** メニューを付加します。

Word 文書を M- ブックに変換する方法

Word 文書を M- ブックに変換するには、つぎの操作を行います。

1 新しい M- ブックを作成する。

2 **Insert** メニューから、**File** コマンドを選択

3 変換したいファイルを選択

4 OK ボタンをクリック

M- ブックに MATLAB コマンドを入力

注意 ノートブックの使用法を学ぶには、サンプル M- ブック Readme.doc を開いて、この節で紹介される様々なテクニックを実際に試してみることです。このサンプルファイルは、MATLAB をインストールしたディレクトリを $MATLAB とすると、$MATLAB\notebook\pc ディレクトリに置かれています。

他の Word 文書にテキストを入力するのと同様の方法で、MATLAB コマンドを M- ブックに入力することができます。たとえば、つぎのテキストを Word 文書に入力することもできます。この例では、フォントに Courier フォントを使用していますが、他の任意のフォントを使用することも可能です。

```
Here is a sample M-book.

a = magic(3)
```

この文書に入力された MATLAB magic コマンドを実行するには、つぎの操作を行う必要があります。

- コマンドを入力セルとして定義
- 入力セルの計算を実行

これらの操作により、MATLAB は、Word 文書のコマンド出力を出力セルに表示します。

ユーザのワークスペースを完全な状態で保護

2 つ以上の M- ブックを単一の Word 処理セッションで使用する場合、つぎのことに注意する必要があります。

- それぞれの M- ブックが、MATLAB の同一 " コピー " を使用します。
- 全ての M- ブックが、同一のワークスペースを共通します。

2 つ以上の M- ブックで同一の変数名を使用すると、1 つの M- ブックで使用されているデータが、他の M- ブックによって影響を受けてしまうことになります。ここ

で、M- ブックで最初に登場する自動実行型セルとして、clear コマンドを指定しておくと、ユーザのワークスペースを完全な状態で保護します。

データの一貫性の確保

M- ブックは、MATLAB セッションを逐次記録したものと考えます。最初の MATLAB コマンドから最後のコマンドまで、順序正しく実行すれば、M- ブックは、これらコマンド間の関係を正確に反映することができます。

ただし、ユーザが、入力セル、または、出力セルを、ユーザの M- ブックを良くするように変更すると、ノートブックは、変更したセルの内容、または、結果に影響を受ける入力セルの再計算を自動的に行いません。その結果、M- ブックには矛盾したデータが含まれてしまうことになります。

M- ブックで作業するときは、ユーザの M- ブックが妥当性をもつことを確認するため、定期的に、**M- ブックの計算 (Evaluate M-book)** を選択することが効果的になります。calc zone を使って、M- ブックの１つのセッションに、互いに関連付けられたコマンドを他の節から切り離して配置することもできます。この場合、**Evaluate Calc Zone** コマンドを使って、calc zone に含まれる入力セルのみ単独で実行することも可能です。

MATLAB コマンドを入力セルとして定義

Word 文書の MATLAB コマンドを入力として定義するには、つぎの操作を行います。

1 つぎのコマンドを M- ブックにテキストで入力します。たとえば、

```
This is a sample M-book.

a = magic(3)
```

2 このコマンドの任意の場所にカーソルを置き、Notebook メニューから **Define Input Cell** コマンドを選択するか、**Alt+D** キーを押します。テキストを構成する行にコマンドが埋め込まれている場合、コマンドをマウスで選択してください。この操作により、ノートブックは、MATLAB コマンドを入力セルとして定義します。

```
This is a sample M-book.

[a = magic(3)]
```

ノートブックは、入力セル内のテキストに適用されるキャラクタフォントを、ボルド、ダークグリーンに変更し、*セルマーカ*で括ります。セルマーカは、ボルドのグレーの括弧です。これらは、サイズや重みで行列を設定する括弧とは異なります。これらのデフォルトフォーマットの変更に関する情報は、10-18 ページの "M- ブックテンプレートで M- スタイルを修正する " を参照してください。

他のタイプの入力セルを定義する方法については、以下を参照してください。

- 10-8 ページの " セルグループの定義 "
- 10-10 ページの " 自動実行型セルの定義 "
- 10-10 ページの "Calc zone の定義 "
- 10-11 ページの " 入力セルをテキストに変換 "

定義した入力セルの評価に関する情報は、10-12 ページの "MATLAB コマンドの実行 " を参照してください。

セルグループの定義

ユーザは、複数の入力セルを単一の入力セルにまとめることができます。複数の入力セルから構成される単一の入力セルを*セルグループ*と呼びます。1 つのセルグループからの出力は全て、対象セルグループの直後にノートブックが挿入した単一の出力セルに出力されます。そのため、グラフィックを定義するために複数の

MATLAB コマンドを使用している場合、セルグループは、大変便利な手段となります。

たとえば、グラフィックを生成するのに必要な MATLAB コマンドを全て 1 つのセルグループとして定義し、そのセルグループを実行するようにすれば、ノートブックはコマンドに定義された全てのグラフィックコマンドを含む単一のグラフィックを生成します。このような方法ではなく、グラフィックを生成する全ての MATLAB コマンドをそれぞれ別々の入力セルに定義して計算を行うと、その結果、複数のグラフィック出力セルが生成されてしまうことになります。

セルグループの評価に関する情報は、10-13 ページの " セルグループの計算 " を参照してください。セルグループの定義を解除する方法については、10-32 ページの "Ungroup Cells コマンド " を参照してください。

セルグループの作成

セルグループを作成するには、つぎの処理を行います。

1 グループにまとめたい複数の入力セルをマウスで選択する。

2 Notebook メニューから **Group Cells** コマンドを選択するか、**Alt+G** キーを押します。

ノートブックは、選択したセルをセルグループに変換し、複数のセルマーカをグループ全体を括る単一のセルマーカに置き換えます。

```
This is a sample cell group.

[date
a = magic(3) ]
```

つぎの点には、注意してください。

- セルグループにテキスト、出力セルを含むことはできません。選択に出力セルが含まれている場合、ノートブックは、このセルを削除します。
- 選択にテキストが含まれている場合、ノートブックはテキストをセルグループの後に移動します。しかし、選択範囲の最初の入力セルより前にテキストが存在する場合には、ノートブックはテキストの位置を移動させずにそのままにしておきます。
- 入力セルではなく、出力セルの全て、または、一部を選択すると、ノートブックは、セルグループに、その出力セルと関連付けられた入力セルを格納します。

セルグループを作成すると、セルグループの最初の行が自動実行型セルでない限

り、ノートブックはそのセルグループを入力セルとして定義します。最初の行が自動実行型の時は、ノートブックは、そのセルグループを自動実行型セルとして定義します。

自動実行型セルの定義

自動実行型セル (autoinit cell) を定義すると、M- ブックが開かれる度に自動実行される MATLAB コマンドを指定することができます。これにより、ワークスペースを迅速かつ容易に初期化することができます。*自動実行型セル*は、通常の入力セルにはないつぎの追加的特性をもつ単一の入力セルです。

- ノートブックは、M- ブックを開いたときに自動実行型セルの計算を実行します。
- ノートブックは、ダークブルーのキャラクタを使って、自動実行型セル内のコマンドを表示します。

自動実行型セルは、以上の点以外では、入力セルと同一の機能をもっています。

自動実行型セルの作成

自動実行型セルは、つぎの 2 つの方法で作成することができます。

- MATLAB コマンドをテキストで入力し、Notebook メニューから **Define AutoInit Cell** コマンドを選択して、入力したコマンドを自動実行型セルに変換します。
- 既に、MATLAB コマンドが入力セルとして入力されている場合、その入力セルを自動実行型セルに変換することができます。変換したい入力セルを選択するか、または、カーソルをそのセルの中に置いて、Notebook メニューから **Define AutoInit Cell** コマンドを選択してください。

自動実行型セルの実行に関する詳細については、10-12 ページの "MATLAB コマンドの実行 " を参照してください。

Calc zone の定義

ユーザは、M- ブックを calc zone と呼ばれる独立したセクションに分けることができます。calc zone は、テキスト、入力セル、出力セルが隣り合わせに並んで配置された 1 つのブロックです。ノートブックは、calc zone を定義したセクションの前後に、Microsoft Word セクション区切り (section break) を挿入します。セクションブロックインジケータは、通常のワードセクション区切りと異なることを示す太字のグレーの括弧で括られます。

それぞれの問題を、単独で作成、テストが可能な個々の calc zone に分割することにより、calc zone を使って問題セットを作成することもできます。M-book に

は、任意の数の calc zone を設定することができます。

注意 calc zone は、M- ブック内の変数領域には影響を及ぼしません。calc zone 内で使用される変数は、全ての calc zone からアクセスできます。

---

calc zone の作成

calc zone を設定したいテキストとセルを作成した後、calc zone を、つぎの操作によって定義してください。

1 calc zone に含める入力セルとテキストを選択します。

2 **Define Calc Zone** コマンドを Notebook メニューから選択します。

注意 calc zone に含めるには、入力セルと出力セルの全体を選択しなくてはなりません。

---

calc zone の計算法に関しては、10-14 ページの "Calc Zone の計算の実行 " を参照してください。

入力セルをテキストに変換

入力セル (自動実行型セル、または、セルグループ) をテキストに変換するには、つぎの操作を行います。

1 入力セルをマウスで選択するか、カーソルを入力セル内に置く。

2 Notebook メニューから、**Undefine Cells** コマンドを選択するか、または、**Alt+U** と押す。

ノートブックが、セルをテキストに変換すると、Microsoft Word 標準スタイルに従って、セルの内容が再フォーマットされます。M- ブックのスタイルに関する詳細は、10-18 ページの "M- ブックテンプレートで M- スタイルを修正する " を参照してください。入力セルをテキストに変換すると、ノートブックは、その入力セルに対応する出力セルもテキストに変換します。

MATLAB コマンドの実行

MATLABコマンドを入力セルとして、あるいは自動実行型セルとして定義した後で、コマンドとして M- ブックで実行することができます。以下のステップにより、MATLAB コマンドを定義し、実行します。

1 つぎのコマンドを M- ブックにテキストで入力します。たとえば、

```
This is a sample M-book

a = magic(3)
```

2 カーソルをコマンド上の任意の場所に置いて、Notebook メニューから **Define Input Cell** コマンドを選択するか、または、**Alt+D** を押します。コマンドがテキスト文の中に埋め込まれている場合には、コマンド部分をマウスで選択します。この操作により、ノートブックは MATLAB コマンドを入力セルとして定義します。たとえば、

```
This is a sample M-book

[a = magic(3)]
```

3 Notebook メニューから **Evaluate Cell** コマンドを選択するか、**Ctrl+Enter** を押します。実行する入力セルを指定するには、マウス、または、カーソルをセル内に置くことによってセルを選択してください。

ノートブックは、入力セルの計算を実行し、入力セルの直後に位置する出力セルに計算結果を表示します。既に出力セルが存在する場合、ノートブックは、その出力セルが M- ブック内のどこにあったとしても、セルの内容を新しい計算結果に置き換えます。たとえば

```
This is a sample M-book.

[a = magic(3) ]

[a =
     8     1     6
     3     5     7
     4     9     2  ]
```

出力セル内のテキストはブルーで、セルマーカで括られています。セルマーカは太字のグレーの括弧です。セルマーカは、行列に用いられるブラケットとは、サ

イズと活字の大きさの 2 つの点で異なります。エラーメッセージは、赤で表記されます。これらのデフォルトフォーマットの変更に関する情報は、10-18 ページの "M- ブックテンプレートで M- スタイルを修正する " を参照してください。

M- ブック内で MATLAB コマンドを実行する方法に関する詳細は、以下を参照してください。

- 10-13 ページの " セルグループの計算 "
- 10-14 ページの " ある範囲の入力セルの計算 "
- 10-14 ページの "Calc Zone の計算の実行 "
- 10-15 ページの "M- ブック全体の計算の実行 "
- 10-15 ページの " ループを使って入力セルの繰り返し計算 "
- 10-16 ページの " 出力セルをテキストに変換 "
- 10-17 ページの " 出力セルの削除 "

セルグループの計算

セルグループの計算については、入力セルの計算と同じ方法で行うことができます (セルグループは入力セルだからです)。

1 セル内、または、対応する出力セル内のどこかにカーソルを置く

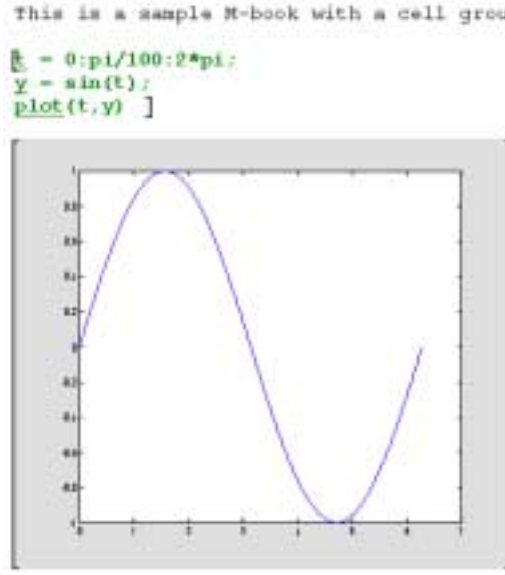
2 Notebook メニューから **Evaluate Cell** コマンドを選択するか、または、**Ctrl+Enter** キーを押す。

セルグループの作成に関する情報は、10-8 ページの " セルグループの定義 " を参照してください。

MATLAB がセルグループの計算を行なうと、グループに含まれる全てのコマンドに対する出力が、単一の出力セルに表示されます。デフォルトでは、ノートブックは、セルグループの計算が最初に実行された時に、セルグループのすぐ後に出力セルを表示します。既に、出力セルを有するセルグループの計算を実行した場合、ノートブックは、既存の出力セルが、M- ブック内のどこにある場合でも、既存のセルに計算結果を出力します。

注意 グループの中のコマンドの順番に関わらず、テキストまたは数値出力のどちらかが、最初に表れます。

図は、セルグループを示し、ユーザがセルグループを計算している場合に、作成されるフィギュアを指名します。



ある範囲の入力セルの計算

互いに異なるが隣り合わせに位置する入力セルに定義されたMATLABコマンド群を実行するには、つぎのように操作します。

1 計算を実行したい入力セルを含むセルの範囲を選択します。入力セルの周囲に書かれたテキストを含んで選択してしまっても構いません。

2 Notebook メニューから **Evaluate Cell** コマンドを選択するか、または、**Ctrl+Enter** キーを押します。

この操作により、ノートブックは、選択した各入力セルの計算を実行し、新しい出力セルを挿入するか、既存の出力セルを置き換えます。

Calc Zone の計算の実行

calc zone の計算を実行するには、つぎのように行います。

1 calc zone 内のどこかにカーソルを置きます。

2 Notebook メニューから **Evaluate Calc Zone** コマンドを選択するか、**Alt+Enter** を押します。

calc zone を作成する方法については、10-10 ページの "Calc zone の定義 " を参照してください。

デフォルトでは、ノートブックは、calc zone の計算が最初に実行されると、calc zone の直後に出力セルを挿入します。既に出力セルが存在する場合、ノートブックは、その出力セルが M- ブック内のどこにあったとしても、セルの内容を新しい計算結果に置き換えます。

M- ブック全体の計算の実行

M- ブック全体の計算を実行するには、**Evaluate M-book** コマンドか、**Alt+R** を押すかのいずれかを選択します。

ノートブックはカーソル位置に関わらず、M- ブックのトップからブックに含まれる各セルの計算を行っていきます。M- ブックの計算が終わると、ノートブックは新しい出力セルを挿入するか、既存の出力セルを置き換えます。

複数コマンドの実行の制御

M- ブック全体の計算を行っているとき、エラーが発生したとしても、計算は中止されずにそのまま続行されます。エラーが発生したときに計算を停止させるには、つぎの操作を行います。

1 Notebook メニューから **Notebook Options** を選択する。

2 **Stop evaluating on error** チェックボックスをクリックする

ループを使って入力セルの繰り返し計算

一連の MATLAB コマンドを繰り返し実行するには、つぎの操作を行ってください。

1 マウスを使って、入力セルを選択します。選択した入力セルの間に位置する任意のテキストまたは出力セルも含めて選択してしまって構いません。

2 **Evaluate Loop** コマンドを選択するか、または **Alt+L** を押してください。これにより、ノートブックは、**Evaluate Loop** ダイアログボックスを開きます。

3 MATLAB が選択したコマンドの計算を実行する回数を **Stop After** フィールドで選択し、**Start** ボタンを押します。このボタンの名前が **Stop** に変わります。この操作を行うと、ノートブックはコマンドの実行を開始し、今までに実行を完了した反復計算の回数を **Loop Count** フィールドに表示します。

Slower または **Faster** ボタンを使って、反復実行終了時の遅れを調節することができます。Slower ボタンは遅れ値を大きく、Faster ボタンは遅れを小さくします。

コマンドの実行を中断するには、**Pause** ボタンを押します。ボタンを押すとボタンの名前が **Resume** に変化します。ここで、**Resume** ボタンを押すと、コマンドの実行に復帰できます。

コマンドの処理を停止するには、**Stop** ボタンを押してください。**Evaluate Loop** ダイアログボックスをクローズするには、**Close** ボタンを押してください。

出力セルをテキストに変換

ユーザは、**Undefine Cells** コマンドを使って、出力セルをテキストに変換することができます。出力が数値、または、テキストの場合、ノートブックはセルマーカを削除し、セルの内容を Microsoft Word 標準スタイルに従って、テキストに変換します。出力がグラフィックスの場合、ノートブックは、セルマーカを削除し、入力セルからグラフィックスを分離しますが、その内容に変更を加えることはありません。

注意 未定義の出力セルは、関連した入力セルに影響を与えません。

出力セルを未定義にするには、

1 未定義のセルにしたい出力セルを選択します。

2 Notebook メニューから、**Undefine Cells** コマンドを選択するか、**Alt+U** キーを押します。

出力セルの削除

出力セルを削除するには、つぎの操作を行います。

1 出力セルをマウス、または、カーソルで選択します。

2 Notebook メニューから **Purge Output Cells** コマンドを選択するか、**Alt+P** キーを押します。

それぞれに出力セルが関連付けられた範囲の入力セルを選択している場合、関連付けられた出力セルの全てを削除します。

M- ブックの印刷とフォーマット

この節では、つぎの内容を解説します。

- 10-18 ページの "M- ブックの印刷 "
- 10-18 ページの "M- ブックテンプレートで M- スタイルを修正する "
- 10-19 ページの "Loose、または、Compact フォーマットの選択 "
- 10-20 ページの " 数値出力フォーマットの制御 "
- 10-20 ページの " グラフィック出力の制御 "

M- ブックの印刷

File メニューから **Print** コマンドを選択すれば、M- ブックの全部、または、一部を印刷できます。M- ブックのセルやグラフィックスを印刷する場合、Word は、つぎの規則に従って印刷を行います。

- セルマーカは、印刷しません。
- 入力セル、自動実行型セル、出力セル (エラーメッセージを含む) が、それぞれを定義したスタイルで印刷します。これらのセルを、カラー、または、グレーの影付きで印刷せずに、黒で印刷するには、スタイルを変更してください。

M- ブックテンプレートで M- スタイルを修正する

M-ブックテンプレートに前もって設定されているスタイルを変更することによって、M- ブックにテキストがどのように表示されるかを制御することができます。ここで、指定したスタイルは、テキストやセルの表示を制御します。デフォルトでは、M- ブックは、他の全てのテキストに対して、Word の標準スタイルを適用します。

たとえば、M- ブックをカラープリンタに印刷する場合、入力セルがダークグリーン、出力や自動実行型セルはダークブルー、エラーメッセージは赤で印刷されます。また、グレースケールのプリンタで印刷する場合、これらのセルは薄いグレーで印刷されます。これらのセルを白黒で印刷するには、M- ブックテンプレートで入力、出力、自動実行型、エラーにそれぞれ適用されたスタイルを変更しなければなりません。

下の表は、Notebook で使用するデフォルトのスタイルをまとめたものです。スタイルを変更した場合、下の表の情報を使えば、最初のセッティングに戻すことが容易に出来るようになります。Word の中でのスタイルの使用に関する一般的な情報については、Word のマニュアルを参照してください。

スタイル	フォント	サイズ	太さ	カラー
標準	Times New Roman	10 points		黒
自動実行型	Courier New	10 points	Bold	ダークブルー
エラー	Courier New	10 points	Bold	赤
入力セル	Courier New	10 points	Bold	ダークグリーン
出力セル	Courier New	10 points		青

スタイルを変更すると、Word は、そのスタイルを使用する M- ブックの全てのキャラクタに変更内容が適用され、テンプレートを変更するオプションが提供されます。テンプレートに変更を加える際は十分に注意してください。テンプレートに変更を適用すると、そのテンプレートを使って作成する新しい M- ブックの全てがその変更の影響を受けることになります。詳細は、Word のマニュアルを参照してください。

Loose、または、Compact フォーマットの選択

Loose、または、Compact のいずれかのフォーマットを選択することによって、入力セルと出力セルの間に黒い線を表示するかどうかを指定できます。

1 Notebook メニューをクリックして、**Notebook Options** を選択します。

2 **Notebook Options** ダイアログボックスで、Loose、または、Compact チェックボックスのいずれかを選択します。Loose フォーマットは、空のラインを付加します。Compact フォーマットは、空のラインを付加しません。

3 **OK** をクリックします。

注意 Notebook Options ダイアログボックスを使って行った変更は、OK ボタンを押した*後に*生成された出力に対して有効となります。既存の入力セル、または、出力セルに影響を及ぼすには、セルの再実行を行う必要があります。

数値出力フォーマットの制御

Notebook が、数値出力表示法の変更を行なうには、つぎの操作を行います。

1 Notebook メニューをクリックして、**Notebook Options** を選択します。

2 **Notebook Options** ダイアログボックスで、**Numeric Format** メニューをクリックし、利用可能なフォーマットの一覧を参照します。ここには、MATLAB の format コマンドで利用可能なフォーマットが表示されます。下の図は、利用可能な数値フォーマットを表示しています。

3 **OK** をクリックしてください。

注意 Notebook Options ダイアログボックスを使って行った変更は、OK ボタンを押した*後に*、生成された出力に対して有効となります。既存の入力セル、または、出力セルに影響を及ぼすには、セルの再実行を行う必要があります。

グラフィック出力の制御

この節は、M- ブックの中の MATLAB コマンドによって生成されたグラフィック出力をいろいろな側面から制御するためのノウハウを解説します。ここで、説明される内容はつぎのものです。

• 10-21 ページの "M- ブックにグラフィック出力を埋め込む "

- 10-21 ページの " 個々の入力セルに対するグラフィック出力を省略する "
- 10-22 ページの " グラフィック出力のサイズを調整する "
- 10-23 ページの " 出力されたグラフィックを切り取る "
- 10-23 ページの " グラフィック出力の周りに白のスペースを付加する "
- 10-23 ページの " カラーモードの指定 "

M- ブックにグラフィック出力を埋め込む

デフォルトでは、グラフィック出力は、M- ブックに埋め込まれます。グラフィック出力を別個のフィギュアウィンドウに表示するには、つぎの操作を行ってください。

1 Notebook メニューをクリックして、**Notebook Options** を選択します。

2 **Notebook Options** ダイアログボックスで、**Embed Figures in M-book** チェックボックスのチェックを外します。

3 **OK** をクリックします。

注意 埋め込まれたフィギュアは、関数 uicontrol や関数 uimenu によって生成された Handle Graphics® オブジェクトを含みません。

ノートブックは、フィギュアオブジェクトの Visible プロパティの値をチェックして、M- ブックにフィギュアを埋め込むかどうかを決定します。このプロパティの値が、off になっていれば、Notebook はフィギュアを埋め込みます。プロパティの値が、on になっていれば、全てのグラフィック出力をカレントフィギュアウィンドウに表示します。

個々の入力セルに対するグラフィック出力を省略する

入力セル、または、自動実行型セルが、生成するフィギュア出力を省略したい場合には、

1 カーソルを入力セルの中に置く

2 Notebook メニューをクリックして、Toggle Graph Output for Cell コマンドを選択します。

ノートブックは、セルからのグラフィック出力を、入力セルの後に文字列 (図なし) を挿入することによって省略します。

セルに対するグラフィック出力を可能にするには、同じ手続きを繰り返します。ノートブックは、セルからのグラフィック出力を可能にし、(図なし) マーカを削除します。

注意 Toggle Graph Output for Cell コマンドは、Embed Figures in M-book オプションが設定されていても優先されます。

グラフィック出力のサイズを調整する

M- ブックに埋め込まれたグラフィクスのデフォルトサイズを設定するには、つぎの操作を行います。

1 Notebook メニューをクリックして、Notebook Options を選択します。

2 Notebook Options ダイアログボックスで、Height や Width フィールドを使って、M- ブックによって生成されるグラフィックスのサイズを設定します。

3 OK をクリックします。

注意 Notebook Options ダイアログボックスを使って行った変更は、OK ボタンを押した*後に*生成された出力に対して有効となります。既存の入力セル、または、出力セルに影響を及ぼすには、セルの再実行を行う必要があります。

既に埋め込まれたフィギュアのサイズを変更するには、フィギュアを選択して、フィギュア上の任意の場所で左マウスボタンをクリックし、フィギュアのリサイズハンドルをドラッグします。フィギュアのハンドルを使って埋め込まれたフィギュアのサイズを変更すると、フィギュアが再び生成され、サイズが元のサイズに逆戻りしてしまいます。

出力されたグラフィックを切り取る

表示したくない部分を削除するために、埋め込まれたフィギュアを切り取るには、つぎの操作を行います。

1 フィギュアの任意の場所で左マウスボタンを押して、グラフィックを選択する。

2 **Shift** を押す。

3 グラフィックの中心に向かってサイズ調整ハンドルをドラッグする。

グラフィック出力の周りに白のスペースを付加する

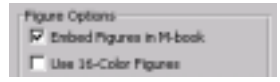
グラフィックの外側境界を動かすことによって、白のスペースをグラフィックのまわりに付加することができます。グラフィックを選択し、**Shift** キーを押して、サイズ調整ハンドルをグラフィックから離れたところにドラッグしてください。

カラーモードの指定

面、または、パッチを含むグラフィック出力を印刷する場合、デフォルトでは、出力は 16 色モードを使用します。256 色モードを使用するには、つぎの操作をしてください。

1 Notebook メニューをクリックして、**Notebook Options** を選択します。

2 **Notebook Options**ダイアログボックスの**Use 16-Color Figures**チェックボックスのチェックを外してください。



3 **OK** をクリックします。

注意　**Notebook Options** ダイアログボックスを使って行った変更は、**OK** ボタンを押した*後に*生成された出力に対して有効となります。既存の入力セル、または、出力セルに影響を及ぼすには、セルの再実行を行う必要があります。

ノートブックの設定

ノートブックをインストールすると、その設定を行わなくてはなりません (ノートブックは、MATLAB をインストールしたときに一緒にインストールされます。詳細は、*MATLAB Installation Guide for PC* を参照してください)。

ノートブックの設定を行うには、MATLAB コマンドウィンドウで、つぎのようにタイプします。

```
notebook -setup
```

ノートブックは、ユーザがどのバージョンの Microsoft Word を使用しているのかについて訊ねてきます。

```
Choose your version of Microsoft Word:
[1] Microsoft Word for Windows 95 (Version 7.0)
[2] Microsoft Word 97
[3] Microsoft Word 2000
[4] Exit, making no changes

Microsoft Word Version: 2
```

つぎに、ノートブックは、Microsoft Word 実行可能ファイル (winword.exe) が、ユーザのシステムのどこにインストールされているのかについて訊ねます。

```
You will be presented with a dialog box.  Please use it to select
your copy of the Microsoft Word 97 executable (winword.exe).
Press any key to continue...
```

つぎのダイアログボックスを使って、Microsoft Word プログラムのロケーションを指定してください。

Open をクリックすると、ノートブックは、Microsoft Word テンプレートファイルの名前を指定するよう要求します。

```
You will be presented with a dialog box.  Please use it to
select a Microsoft Word template (.dot) file in one of your
Microsoft Word template directories.  We suggest that you specify
your normal.dot file.
Press any key to continue...
```

つぎのダイアログボックスを使って、ユーザの Microsoft Word テンプレートファイルの名前を指定してください。

以上の設定が完了すると、ノートブックは、つぎのメッセージを出力します。

```
Notebook setup is complete.
```

ノートブックコマンドリファレンス

この節では、ノートブックコマンドの各々に関するリファレンス情報をアルファベット順に提供します。これらのコマンドを使用するには、Notebook メニューから選択してください。

- 10-26 ページの "Bring MATLAB to Front コマンド "
- 10-26 ページの "Define Autoinit Cell コマンド "
- 10-27 ページの "Define Calc Zone コマンド "
- 10-27 ページの "Define Input Cell コマンド "
- 10-28 ページの "Evaluate Calc Zone コマンド "
- 10-28 ページの "Evaluate Cell コマンド "
- 10-29 ページの "Evaluate Loop コマンド "
- 10-30 ページの "Evaluate M-Book コマンド "
- 10-30 ページの "Group Cells コマンド "
- 10-31 ページの "Hide Cell Markers コマンド "
- 10-31 ページの "Notebook Options コマンド "
- 10-31 ページの "Purge Output Cells コマンド "
- 10-31 ページの "Toggle Graph Output for Cell コマンド "
- 10-32 ページの "Undefine Cells コマンド "
- 10-32 ページの "Ungroup Cells コマンド "

Bring MATLAB to Front コマンド

Bring MATLAB to Front コマンドは、MATLAB コマンドウィンドウを前面に表示します。

Define Autoinit Cell コマンド

Define AutoInit Cell コマンドは、カレントパラグラフ、選択されたテキスト、入力セルを変換して、自動実行型セルを作成します。自動実行型（Autoinit）セルとは、M-ブックを開いたときに自動的に計算が実行される入力セルのことです。

コマンドの機能

カーソルがテキストパラグラフ上にある場合、このコマンドを選択すると、ノートブックは、そのパラグラフ全体を自動実行型セルに変換します。テキストを選

択している場合、このコマンドを選択すると、ノートブックは選択したテキストを自動実行型セルに変換します。カーソルが入力セルの中にある場合、このコマンドを選択すると、ノートブックは入力セルを自動実行型セルに変換します。

フォーマット

ノートブックは、AutoInit スタイル（太字、ダークブルー、フォントは 10 ポイントの Courier New）を適用して自動実行型セルをフォーマットします。

参照

自動実行型セルに関する詳細は、10-10 ページの " 自動実行型セルの定義 " を参照してください。

Define Calc Zone コマンド

Define Calc Zone コマンドは、選択したテキスト、入力セル、出力セルを、calc zone として定義します。calc zone は、特定の操作、または、問題を記述するテキスト、入力セル、出力セルが隣り合わせに配置されたブロックです。

コマンドの機能

ノートブックは、calc zone を Word の文書セッションとして定義し、calc zone の前後に節の区切りを挿入します。但し、ワードは文書の最初と最後に節の区切りを表示しません。

参照

calc zone の計算を実行する方法に関する詳細は、10-14 ページの "Calc Zone の計算の実行 " を参照してください。文章節の詳細についてや、Microsoft Word のマニュアルを参照してください。

Define Input Cell コマンド

Define Input Cell コマンドは、カレントのパラグラフ、選択されたテキスト、自動実行型セルを変換して入力セルを作成します。入力セルは、MATLAB コマンドを含みます。

コマンドの機能

カーソルがテキストパラグラフにある場合、このコマンドを選択すると、ノートブックは、このパラグラフ全体を入力セルに変換します。また、テキストが選択

されているときに、このコマンドを選択すると、ノートブックは、テキストを入力セルに変換します。

自動実行型セルの中にカーソルが位置するときにこのコマンドを選択すると、ノートブックは選択された自動実行型セルを入力セルに変換します。

フォーマット

ノートブックは、テキストをセルマーカーで括り、Input スタイル太字、ダークグリーン、Courier New 10 ポイント) を適用してセルをフォーマットします。

参照

入力セルの作成に関する詳細は、10-8 ページの "MATLAB コマンドを入力セルとして定義 " を参照してください。入力セルの評価に関する詳細は、10-12 ページの "MATLAB コマンドの実行 " を参照してください。

Evaluate Calc Zone コマンド

Evaluate Calc Zone コマンドは、カレント calc zone の入力セルを MATLAB に送って、計算を実行します。calc zone は、特定の操作、または、問題を記述するテキスト、入力セル、出力セルが隣り合わせに配置されたブロックです。

カレント calc zone とは、カーソルが位置するワード節のことです。

コマンドの機能

ノートブックは、各入力セルの計算を実行し、出力セルを生成します。関連付けられた出力セルがない入力セルの計算を実行した時は、ノートブックは出力セルを、そのセルを出力した入力セルの直後に挿入します。関連付けられた出力セルが既に存在する入力セルの計算を実行した場合は、ノートブックは、既存の出力セルが M- ファイルのどこにあるかにかかわらず、その出力セルを置き換えます。

参照

詳細は、10-14 ページの "Calc Zone の計算の実行 " を参照してください。

Evaluate Cell コマンド

Evaluate Cell コマンドは、カレントの入力セル、または、セルグループを MATLAB に送り、計算を実行します。入力セルは、MATLAB コマンドを含みます。セルグループは、1 つ以上の MATLAB コマンドを含む、単一または複数の入力セルです。ノートブックは、出力またはエラーメッセージを出力セルに表示します。

コマンドの機能

関連付けられた出力セルがない入力セルの計算を実行すると、ノートブックは新しい出力セルを、そのセルを生成した入力セルの直後に挿入します。既に関連付けられた出力セルが存在する入力セルの計算を実行した場合、ノートブックは、その出力セルがM-ブックのどこにあるかにかかわらず､既存の出力セルに表示される出力結果を置き換えます。セルグループの計算を実行すると、グループを構成するセルの出力全てが単一の出力セルに表示されます。

ある入力セル、または、セルグループがカレントの入力セル、または、セルグループとなるのは、つぎの場合です。

- カーソルが、対象入力セル、または、セルグループ内に置かれている。
- カーソルが、対象入力セル、または、セルグループを括っている最後のセルマーカーを含むラインの末尾にある。
- カーソルが、対象入力セル、または、セルグループに関連付けられた出力セル内部にある。
- 対象入力セル、または、セルグループが選択されている。

注意　長い演算を含むセルの計算を実行すると、タイムアウトを引き起こす可能性があります。タイムアウトが起きると、ワードはタイムアウトメッセージを表示し、出力されるまでまつか、それとも実行を中止するのかという二者択一を迫ります。ここで、実行を続けることを選択すると、ワードはタイムアウト値をリセットし、出力が返されるまで待ち続けます。ワードは、タイムアウトを設定していますが、ユーザがこれを変更することはできません。

参照

詳細は、10-12 ページの "MATLAB コマンドの実行 " を参照してください。M-ブック全体の評価に関する詳細は、10-15 ページの "M-ブック全体の計算の実行 " を参照してください。

Evaluate Loop コマンド

Evaluate Loop コマンドは、選択された入力セルの計算を繰り返し実行します。

詳細は、10-15 ページの " ループを使って入力セルの繰り返し計算 " を参照してください。

Evaluate M-Book コマンド

Evaluate M-book コマンドは、M- ブックを構成する全ての入力セルを MATLAB に送り、その計算を実行し、M- ブック全体の計算を実行するコマンドです。カーソルの位置に関わらず、ノートブックは、M- ブックのトップから実行を開始します。

ファイルの機能

ノートブックは、M- ブックを構成する各入力セルの計算を実行し、出力セルを生成します。関連付けられた出力セルがない入力セルの計算を実行すると、ノートブックは新しい出力セルを、そのセルを生成した入力セルの直後に挿入します。既に関連付けられた出力セルが存在する入力セルの計算を実行した場合には、ノートブックは、その出力セルが M- ブックのどこにあるかに関わらず、既存の出力セルに表示される出力結果を置き換えます。

参照

詳細は、10-15 ページの "M- ブック全体の計算の実行 " を参照してください。

Group Cells コマンド

Group Cells コマンドは、選択した複数の入力セルを、セルグループと呼ばれる複数ラインからなる単一の入力セルに変換します。ここで、作成したセルグループの計算を行うには、**Evaluate Cell** コマンドを使用してください。セルグループの計算を実行すると、出力結果は全てセルグループに続いて挿入された単一の出力セルに表示されます。

ファイルの機能

選択した範囲にテキストが含まれている場合、ノートブックは、そのテキストをセルグループの後ろへ移動します。しかし、グループを構成する最初の入力セルの前にテキストが位置している場合には、テキストはセルグループの前にそのまま配置されます。

選択した範囲に出力セルが含まれている場合、ノートブックは、この出力セルを削除します。このコマンドを選択する前に出力セルの全て、または、一部を選択すると、ノートブックは、その出力セルに関連付けられた入力セルをグループに格納します。

セルグループを構成する最初のラインが自動実行型セルの場合、グループ全体が一連の自動実行型セルであるかのような挙動を示します。最初のラインが自動実行型出ないときは、グループは一連の入力セルとして機能します。セルグループ全体を

自動実行型セルに変換するには、**Define AutoInit Cell** コマンドを使用します。

参照
詳細は、10-8 ページの " セルグループの定義 " を参照してください。なお、セルグループを個々の入力セルに変換する方法については、10-32 ページの "Ungroup Cells コマンド " を参照してください。

Hide Cell Markers コマンド

Hide Cell Markers コマンドは、M- ブックのセルマーカを隠すコマンドです。

このコマンドを選択すると、このコマンドは、**Show Cell Markers** に変化します。

注意 ノートブックは、セルマーカをスクリーン上で表示、非表示のいずれにしているかに関わらず、印刷時には、セルマーカを印刷しません。

Notebook Options コマンド

Notebook Options コマンドは、数値やグラフィック出力の表示オプションをチェック、変更するためのコマンドです。

参照
詳細は、10-18 ページの "M- ブックの印刷とフォーマット " を参照してください。

Purge Output Cells コマンド

Purge Output Cells コマンドは、カレントの選択範囲から全ての出力セルを削除します。

参照
詳細は、10-17 ページの " 出力セルの削除 " を参照してください。

Toggle Graph Output for Cell コマンド

Toggle Graph Output for Cell コマンドは、入力セルからのグラフィック出力を省略、または、出力を可能にするコマンドです。

入力、または、自動実行型セルが表示を省略したいフィギュアを生成している場合、入力セルの中にカーソルを置き、このコマンドを選択します。文字列 (図なし) が、入力セルのすぐ後に挿入され、この入力セルからのグラフ出力が省略されていることを表示します。

このセルからのグラフィック出力を可能にするには、この入力セルの中にカーソルを置いて、**Toggle Graph Output for Cell** をもう 1 度選択します。すると、(図なし) マーカが削除されます。このコマンドは、M- ブックの Embed Graphic Output オプションに優先します。なお、Embbed Graphic Output オプションは、**Notebook Options** ダイアログで設定します。

参照

詳細は、10-21 ページの "M- ブックにグラフィック出力を埋め込む " と 10-21 ページの " 個々の入力セルに対するグラフィック出力を省略する " を参照してください。

Undefine Cells コマンド

Undefine Cells コマンドは、選択したセルをテキストに変換します。セルが何ら選択されていないが、カーソルがあるセルの中に置かれているという場合では、ノートブックは、このカーソルが置かれたセルを未定義にします。ノートブックは、セルマーカを除去し、標準スタイルでセルの書式設定を行います。

ユーザが入力セルを未定義にすると、ノートブックは自動的に、そのセルに関連付けられていた出力セルも未定義にします。但し、出力セルを未定義にした場合、そのセルに関連付けられた入力セルを未定義にする操作を行いません。埋め込みグラフィックを含む出力セルを未定義にした場合、その埋め込みグラフィックは M- ブックに、そのまま残りますが、入力セルとの関連付けは削除されます。

参照

Normal スタイルに関する詳細は、10-18 ページの "M- ブックテンプレートで M- スタイルを修正する " を参照してください。出力セルの削除に関する詳細は、10-31 ページの "Purge Output Cells コマンド " を参照してください。

Ungroup Cells コマンド

Ungroup Cells コマンドは、カレントのセルグループを個々の入力セル、または、自動実行型セルに切り離します。セルグループが入力セルの場合、ノートブックはセルグループを複数の入力セルに変換します。一方、セルグループが自動実行型であれば、ノートブックはそのセルグループを複数の自動実行型セルに変換し

ます。この場合、ノートブックは、セルグループに関連付けられていた出力セルを削除します。

あるセルグループがカレントとなるのは、つぎの条件の時です。

- カーソルが対象セルグループの中に置かれている。
- カーソルが対象セルグループを括っている最後のセルマーカーを含むラインに置かれている。
- カーソルが対象セルグループに関連付けられた出力セルの中に置かれている。
- セルグループが選択されている。

参照

セルグループの作成に関する詳細は、10-8 ページの " セルグループの定義 " を参照してください。

数学

MATLAB は、数学演算やデータ解析を行なう多くの関数を用意しています。つぎのリストは、それらについて、まとめたものです。

- 行列と 線形代数 – MATLAB で直接サポートしている行列作成や行列演算に関して、記述しています。カバーしているトピックは、行列代数、線形方程式、固有値、特異値、行列の因子分解です。
- 多項式と 内挿 – 多項式の根。多項式の計算、微分等の標準の多項式演算に関する関数を記述しています。カバーしている付加的なトピックは、カーブフィッテングや部分分数展開です。
- D データ解析と 統計 – MATLAB での、データ解析用の配列の作成法、簡単な記述的統計関数の使い方、データの前処理手法に関する記述を行ないます。カバーしている付加的なトピックは、レギュレション、カーブフィッテング、フィルタリング、フーリエ変換 (FFTs) です。
- 関数を引数とする 関数 – 数値配列の代わりに数学関数を取り扱う MATLAB 関数を記述します。これらの関数を引数とする関数は、プロット、最適化、ゼロ検出、数値積分 (求積法) を含んでいます。
- 微分 方程式 – MATLAB での、常微分方程式 (ODEs) や微分代数方程式 (DAEs) に関する初期値問題の解法、ODEs に関する境界値問題に関する解法を記述しています。ここでは、放物線型、楕円型偏微分方程式 (PDEs) の初期 - 境界値問題の解法についても記述しています。カバーしているトピックは、MATLAB での問題の表現、ソルバインタックス、積分パラメータの使い方も含んでいます。
- スパース 行列 – MATLAB の中でのスパース行列の作成法、特殊な数学演算、一般的な数学演算でのスパースの使い方を説明しています。

11

行列と線形代数

関数のまとめ 11-3

MATLAB の中での行列 11-5
作成 11-5
加算と減算 11-6
ベクトル積と転置 11-7
行列の乗算 11-9
単位行列 11-10
Knonecker テンソル積 11-11
ベクトルと行列のノルム 11-12

線形方程式の解法 11-13
概要 11-13
正方システム 11-14
過多システム 11-15
過少システム 11-18

逆行列と行列式 11-21
概要 11-21
擬似逆行列 11-22

LU, QR, コレスキ因子分解 11-25
コレスキ因子分解 11-25
LU 因子分解 11-26
QR 因子分解 11-28

行列のベキと指数 11-32

固有値 11-35

特異値分解 11-39

MATLAB は、行列計算、線形方程式、固有値、特異値、行列分解を含む*線形代数*で通常用いられる多くの行列演算をサポートします。MATLAB を使って行列全体を迅速、また簡単に作業することができます。

本章では、MATLAB での基本的な行列演算を説明し、問題を解く際の利用法を説明します。以下の項目を含みます。

関数のまとめ
MATLAB 線形代数関数をまとめています。

MATLAB の中での行列
MATLAB の中での行列と基本的な行列演算の説明

線形方程式の解法
正方システム、過多システム、過少システムを含む連立線形方程式の MATLAB での解法を説明します。

逆行列と行列式
線形連立方程式の解法の中で、逆行列、行列式、擬似逆行列の MATLAB での使い方を説明します。

LU, QR, コレスキ因子分解
コレスキ因子分解、ガウス消去法、直交化を使って、三角行列を含む連立方程式を MATLAB での解法について、議論します。

行列のベキと指数
MATLAB 記法を使って、種々のベキと指数を計算する方法を説明します。

固有値
固有値の説明と MATLAB の中での固有値分解の説明

特異値分解
MATLAB での直交行列の特異値分解を説明します。

関数のまとめ

線形代数関数は、MATLAB の中のデレクトリ matfun に配置されています。

関数のまとめ

カテゴリ	関数	詳細
行列解析	norm	行列のノルムまたはベクトルのノルム
	normest	行列の 2 ノルムを計算
	rank	行列のランク
	det	行列式
	trace	対角要素の和
	null	空空間
	orth	直交化
	rref	行処理過程を表示
	subspace	2 つの部分空間間の角度
線形方程式	\ および /	線形方程式の解
	inv	逆行列
	cond	逆変換に対する条件数
	condest	1 ノルムの条件数推定
	chol	Cholesky 因子分解
	cholinc	不完全 Cholesky 因子分解
	lu	LU 因子分解
	luinc	不完全 LU 因子分解
	qr	直交三角分解
	lsqnonneg	非負最小二乗

関数のまとめ（続き）

カテゴリ	関数	詳細
	pinv	擬似逆行列
	lscov	共分散が既知の条件下での最小二乗
固有値と特異値	eig	固有値と固有ベクトル
	svd	特異値分解
	eigs	2,3 の固有値
	svds	2,3 の特異値
	poly	特性多項式
	polyeig	多項式固有値問題
	condeig	固有値に対する条件数
	hess	Hessenberg 型
	qz	QZ 因子分解
	schur	Schur 分解
行列関数	expm	行列のべき乗計算
	logm	行列の対数
	sqrtm	行列の平方根
	funm	一般的な行列関数の計算

MATLAB の中での行列

*行列*は、実数または複素数からなる 2 次元配列です。*線形代数*は、MATLAB によって直接サポートされる多くの行列演算を定義します。行列計算、線形方程式、固有値、特異値、行列の分解が含まれます。

このセルでは、つぎの行列演算について説明します。

- 作成
- 加算と減算
- ベクトル積と転置
- 行列の乗算

つぎの事柄を作成するために使用する MATLAB 関数の記述を行ないます。

- 単位行列
- 2 つの行列の Knonecker テンソル積
- ベクトルと行列のノルム

作成

非公式に、行列と配列と言う言葉は、しばしばお互いに混同して使われます。より詳しく言うと、行列は、線形変換を表現する実数または複素数からなる 2 次元長方形配列です。行列上で定義される線形代数演算は、広く技術分野で使われています (Symbolic Math Toolbox は、MATLAB の機能を数値でない行列の種々のタイプものに拡張することができます)。

MATLAB は、種々の異なる行列を作成する関数をいくつか用意しています。これらの中の二つは、この章全体を通して使用する 3 行 3 列の例題行列を作成するために使っています。最初の例題は、対称行列です。

```
A = pascal(3)

A =
     1     1     1
     1     2     3
     1     3     6
```

2 番目の例題は、非対称行列です。

```
B = magic(3)
```

```
B =
     8     1     6
     3     5     7
     4     9     2
```

他の例題は、ランダムな整数から作られる 3 行 2 列の長方形行列です。

```
C = fix(10*rand(3,2))

C =
     9     4
     2     8
     6     7
```

*列ベクトル*は *m* 行 1 列の行列で、*行ベクトル* は 1 行 *n* 列の行列で、*スカラ*は 1 行 1 列の行列です。ステートメント

```
u = [3; 1; 4]

v = [2 0 -1]

s = 7
```

は、列ベクトル、行ベクトル、スカラを作成します。

```
u =
     3
     1
     4

v =
     2     0    -1

s =
     7
```

加算と減算

行列の加算と減算は、配列、すなわち、要素単位で定義されるものです。A と B を加えたり、その結果から A を引き、B になることを示します。

```
X = A + B
```

```
X =
     9     2     7
     4     7    10
     5    12     8

Y = X -A

Y =
     8     1     6
     3     5     7
     4     9     2
```

加算と減算は、二つの行列が同じ大きさであるか、または、そのどちらかがスカラである必要があります。次元に矛盾があるならば、エラーが生じます。

```
X = A + C
Error using ==> +
Matrix dimensions must agree.

w = v + s

w =
     9     7     6
```

ベクトル積と転置

同じ長さの行ベクトルと列ベクトルは、どちらか一方のベクトルを基準に他のベクトルを順番に従って、乗算していきます。結果は、スカラ、すなわち、同じ長さの行ベクトルと列ベクトルは、どちらか一方のベクトルを基準に他のベクトルを順番に従って、乗算していきます。結果は、スカラ、すなわち、*内* 積になるか、行列すなわち*外* 積のどちらかになります。

```
x = v*u

x =
     2

X = u*v

X =
     6     0    -3
```

```
     2     0    -1
     8     0    -4
```

実数行列に対して、*転置* 演算は、a_{ij} を a_{ji} に変換することです。MATLAB はアポストロヒを使って、転置を定義しています。この例題で A は*対称* なので、A' は A に等しくなります。しかし、B は対称ではないので、B' と等しくはありません。

```
X = B'

X =
     8     3     4
     1     5     9
     6     7     2
```

転置は、行ベクトルを列ベクトルに置き換えます。

```
x = v'

x =
     2
     0
    -1
```

x と y が共に実数列ベクトルならば、積 x*y は定義できません。しかし、つぎの積

```
x'*y
```

と

```
y'*x
```

は、定義でき、同じスカラ値を出力します。これは、非常に頻繁に使われ、3 つの異なる名前、*内* 積、*スカラ* 積、*ドット* 積のいずれかの名前で呼ばれます。

複素数ベクトルまたは行列 z に対して、z' は*複素共役転置*を定義します。共役を取らない複素転置は、他の配列演算と同様に、z.' で定義します。それで、

```
z = [1+2i  3+4i]
```

の場合、z' は、つぎのようになります。

```
1-2i
3-4i
```

一方、z.' は、つぎのようになります。

```
1+2i
3+4i
```

複素ベクトルに対して、2 つのスカラ積 x'*y と y'*x は、お互いの複素共役で、スカラ積 x'*x は実数です。

行列の乗算

行列の乗算は、その中に含まれている線形変換の構成を反映する一つの方法で、連立線形方程式のコンパクトな表現です。行列積 *C = AB* は、*A* の列次元が *B* の行次元と等しいときか、またはどちらか一方がスカラのとき、定義されます。*A* が *m* 行 *p* 列で *B* が *p* 行 *n* 列の行列ならば、積 *C* は *m* 行 *n* 列の行列になります。積は、MATLAB の for ループ、コロン記法、ベクトルのドット積を使って定義されます。

```
for i = 1:m
     for j = 1:n
        C(i,j) = A(i,:)*B(:,j);
     end
end
```

MATLAB は、行列乗算を定義するのに、単一アスタリスクを使います。つぎの二つの例は、行列積が累積になっていないことを示すものです。すなわち、*AB* は、通常、*BA* と異なります。

```
X = A*B

X =
      15     15     15
      26     38     26
      41     70     39

Y = B*A

Y =
      15     28     47
      15     34     60
      15     28     43
```

行列は、右の列ベクトルと左の行ベクトルとの乗算になります。

```
x = A*u
```

```
x =
     8
    17
    30

y = v*B

y =
    12    -7    10
```

長方形行列乗算は、次元の整合性を満足していなければなりません。

```
X = A*C

X =
    17    19
    31    41
    51    70

Y = C*A

Error using ==> *
Inner matrix dimensions must agree.
```

他には、スカラによる乗算ができます。

```
w = s*v

w =
    14     0    -7
```

単位行列

一般に受け入れられる数学記法では、対角要素が 1 で、他の要素が 0 である種々の大きさの行列を、*単位* 行列と言い、大文字 *I* を使って定義します。これらの行列は、次元の整合性が保たれている範囲で、*AI = A* また *IA = A* の性質をもっています。MATLAB の元々のバージョンでは、大文字、小文字の区別がなく、i は既にサブスクリプトや複素数単位として使っていたので、単位行列として *I* を使うことができませんでした。それで、語呂合わせを使って記号を作りました。すなわち、I と同じ発音をもつ eye を使うことにしました。関数

```
eye(m,n)
```

は、*m* 行 *n* 列の長方形単位行列を出力し、eye(n) は、*n* 行 *n* 列の正方単位行列を出力します。

Kronecker テンソル積

2 つの行列の Kronecker 積、kron(X,Y) は、X の要素と Y の要素の中で取り得る可能な組み合わせ積から作成できる大きな行列になります。X が *m* 行 *n* 列で、Y が *p* 行 *q* 列ならば、kron(X,Y) は *mp* 行 *nq* 列の行列になります。要素は、つぎの順番で並べられます。

```
[X(1,1)*Y  X(1,2)*Y  . . .  X(1,n)*Y
                     . . .
 X(m,1)*Y  X(m,2)*Y  . . .  X(m,n)*Y]
```

Kronecker 積は、0 と 1 からなる行列を使って、小さな行列の繰り返しコピーを作成します。たとえば、X が、2 行 2 列の行列

```
X =
     1     2
     3     4
```

で、I = eye(2,2) は、2 行 2 列の単位行列とすると、2 つの行列

```
kron(X,I)
```

と

```
kron(I,X)
```

は、つぎのように

```
     1     0     2     0
     0     1     0     2
     3     0     4     0
     0     3     0     4
```

と

```
     1     2     0     0
     3     4     0     0
     0     0     1     2
     0     0     3     4
```

となります。

ベクトルと行列のノルム

ベクトル *x* の *p* ノルムは、つぎのように定義され、

$$\|x\|_p = \left(\sum |x_i|^p\right)^{1/p}$$

norm(x,p) で計算されます。これは、$p > 1$ を満足する任意の値について定義されていますが、実際に使う値は、1, 2, と ∞ です。デフォルト値は、$p = 2$ です。これは、*Euclidean length* と言われます。

```
[norm(v,1) norm(v) norm(v,inf)]

ans =
       3.0000    2.2361    2.0000
```

行列 *A* の *p* ノルム

$$\|A\|_p = \max_x \frac{\|Ax\|_p}{\|x\|_p}$$

は、norm(A,p) により $p = 1, 2, \infty$ に対して計算されます。再び、デフォルト値は、$p = 2$ です。

```
[norm(C,1) norm(C) norm(C,inf)]

ans =
      19.0000   14.8015   13.0000
```

線形方程式の解法

この節は、つぎの事柄について、説明します。

- 線形方程式の一般的な解法
- 正方システムの解法
- 過多システムの解法
- 過少システムの解法

概要

技術計算の中で最も重要な問題の一つは、連立線形方程式を解くことです。行列記法の中で、この問題は、つぎのように記述できます。

2 つの行列 *A* と *B* を与え、つぎの 2 つの条件 *AX = B* または *XA = B* のいずれかを満足するユニークな行列 *X* が存在しますか？

1 行 1 列の簡単な例を考えましょう。

つぎの方程式

$$7x = 21$$

は、ユニークな解をもっていますか？

この解は、もちろん、' はい ' です。方程式は、ユニークな解 $x = 3$ をもちます。方程式は*割り算*により簡単に解を得ます。

$$x = 21/7 = 3$$

解は、通常、7 の逆数、すなわち、$7^{-1} = 0.142857$ ノ を計算し、それを乗算する方法は、*使いません*。それで、7^{-1} に 21 を乗算します。仮に 7^{-1} が有限の小数値で与えられれば、精度は落ちますが、うまく機能します。複数の未知数をもった連立方程式に同様な考えを適用します。MATLAB は、逆行列を使わないで、このような方程式を解きます。

標準の数学的な記法ではありませんが、MATLAB は、一般的な連立方程式を解く方法を記述するスカラの場合に良く使用する除算の方法を使います。2 つの除算記号、*スラッシュ* / と*バックラッシュ* \ が、未知の行列を係数行列の左除算、右除算を示します。

X = A\B　　行列方程式 *AX = B* の解です。

X = B/A　　行列方程式 *XA = B* の解です

AX = B または *XA = B* の方程式の両側を *A* で割ることを考えてください。係数行列 A は常に分母になります。

X = A\B に対する次元の整合性の 条件は、2 つの行列 A と B が、同じ長さの行であることです。それで、解 X は、B と同じ列数になり、行の大きさは A の列の大きさと等しくなります。X = B/A において、行と列の役割は、反転します。

実際に、*AX = B* の線形方程式は、*XA = B* の型のものよりも、非常に頻繁に生じます。結果として、バックスラッシュは、スラッシュよりも頻繁に使われます。この節の残りの部分は、バックスラッシュ演算を集中的に取り扱います。スラッシュ演算子の対応する性質は、つぎの等式から推定できます

```
(B/A)' = (A'\B')
```

係数行列 A は、正方である必要がありません。A が *m* 行 *n* 列の場合は、つぎの 3 ケースが考えられます。

m = n　　正方システム、厳密解が得られます。

m > n　　過多システムで、最小二乗解を求めましょう。

m < n　　過少システムで、*m* 個のゼロでない成分をもつ基本解を求めます。

バックスラッシュ演算子は、種々のタイプの係数行列を取り扱うため、種々のアルゴリズムを使います。種々のケースで、係数行列を調べることにより、自動的に判断します。

- 三角行列の置換
- 対称、正定行列
- 正方、非特異行列
- 長方形、過多システム
- 長方形、過少システム

正方システム

共通なことは、正方係数行列 A と単一右辺列ベクトル b を含んでいることです。

解 x = A\b は、b と同じサイズになります。例えば、

```
x = A\u

x =
      10
     -12
       5
```

A*x が、厳密な意味で u になることを確認してください。

A,B が正方で、同じサイズならば、X = A\B もまた同じサイズになります。

```
X = A\B

X =
      19     -3     -1
     -17      4     13
       6      0     -6
```

A*X が、厳密な意味で B になることを確認してください。

これら 2 つの例題は、厳密な意味で、整数解になります。これは、係数行列は、pascal(3) で求めているためです。すなわち、この行列の行列式は 1 になっているからです。節の後半で、丸め誤差の影響が、現実的な計算に生じているものを考えます。

正方行列 *A* が、線形に独立な列をもっていない場合、*特異*、非正則と言います。非正則ならば *AX = B* での解は、存在しないか、または、ユニークな解ではありません。バックスラッシュ演算 A\B は、A が特異に近い場合、ワーニングを表示し、特異である場合、エラーになります。

過多システム

連立線形方程式の過多システムは、実験データの種々の曲線近似で使われています。ここでは、仮説の例題を考えましょう。値 *y* は、時刻 *t* で測定された観測値です。

t	*y*
0.0	0.82
0.3	0.72
0.8	0.63
1.1	0.60
1.6	0.55
2.3	0.50

このデータは、MATLAB に、つぎのステートメントで読み込むことができます。

```
t = [0 .3 .8 1.1 1.6 2.3]';
y = [.82 .72 .63 .60 .55 .50]';
```

データは、指数的に減衰する関数でモデル化できるとします。

$$y(t) \approx c_1 + c_2 e^{-t}$$

この方程式は、ベクトル y が、定数 1 を含む定数ベクトルと成分 e^{-t} をもつベクトルの 2 つのベクトルの線形結合で近似するものです。未知係数 c_1 と c_2 は、モデルとデータとの偏差の二乗和を最小にする*最小二乗* を行なうことで計算します。ここでは、2つの未知数をもつ6つの方程式により、6行2列の行列で表されます。

```
E = [ones(size(t)) exp(-t)]

E =
    1.0000    1.0000
    1.0000    0.7408
    1.0000    0.4493
    1.0000    0.3329
    1.0000    0.2019
    1.0000    0.1003
```

最小二乗解は、バックスラッシュ演算で求まります。

```
c = E\y

c =
       0.4760
       0.3413
```

別な表現をすると、データの最小二乗適合は、つぎのように表せます。

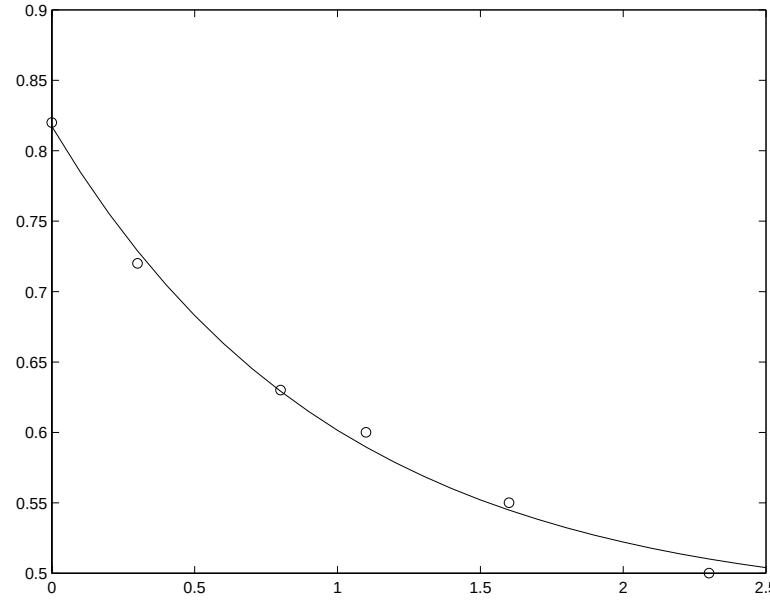
$$y(t) \approx 0.4760 + 0.3413\ e^{-t}$$

つぎのステートメントは、定間隔に刻まれた時刻 t について、モデルを計算し、オリジナルのデータと共にプロットします。

```
T = (0:0.1:2.5)';
Y = [ones(size(T)) exp(-T)]*c;
plot(T,Y,'-',t,y,'o')
```

E*c は、y と正確に一致はしません。が、その違いは、オリジナルデータの中に含まれる測定誤差よりもかなり小さいものです。

長方形行列 *A* は、線形に独立でない列があると、*ランク落ち*になります。*A* がランク落ちならば、*AX = B* の最小二乗解はユニークでなくなります。バックスラッシュ演算 A\B は、*A* がランク落ちの状態なら、ワーニングを表示し、ゼロでない要素ができるだけ少ない*基本解* を出力します。



過少システム

過少線形システムは、方程式の数より未知数の数が多いシステムです。これは、付加的な制約を付けると、*線形計画法*になります。それ自身、バックスラッシュ演算子は、制約のないシステムのみを取り扱います。解はユニークではありません。MATLAB は、少なくとも *m* 個のゼロでない成分をもつ基本解を探します。しかし、この解でさえ、ユニークにはならない可能性があります。実際に計算される特別な解は、列ピボットを使った QR 因子分解 (後半の QR 因子分解を参照) で決定されます。

ここで、小さいランダム行列の例題を考えます。

```
R = fix(10*rand(2,4))

R =
       6     8     7     3
       3     5     4     1

b = fix(10*rand(2,1))

b =
       1
       2
```

線形方程式 *Rx = b* が、4 つの未知数をもつ 2 つの方程式を含むものとします。係数行列は小さな整数を含んでいるので、*rational* フォーマットで表現するのが適切です。特定の解が、つぎのステートメントで得られます。

```
format rat

p = R\b

p =
       0
     5/7
       0
   -11/7
```

R(:,2) が最大ノルムをもつ R の列なので、非セロ成分の一つは、p(2) です。R(:,2) が取り除かれた後、R(:,4) が卓越しているので、他の非ゼロ要素は p(4) になります。

過多システムの完全な解は、ヌル空間の任意のベクトルを付加することにより特徴付けられます。それは、有理基底を必要とするオプションをもった関数 null を使って求められます。

```
Z = null(R,'r')

Z =
      -1/2         -7/6
      -1/2          1/2
```

```
     1          0
     0          1
```

R*z はゼロになり、任意のベクトル q に対して、

```
x = p + Z*q
```

の型の任意のベクトルが R*x=b を満足します。

逆行列と行列式

この節では、つぎの事柄の説明を行ないます。

- 線形システムの正則なシステムを解くための逆行列と行列式の使用法の概要
- 線形方程式の長方形システムを解くための Moore-Penrose 擬似逆行列の使用

概要

A が正方で、正則ならば、方程式 *AX = I* と *XA = I* は、同じ解 *X* を持ちます。この解は、*A* の*逆行列と呼ばれ*、A^{-1} で表し、関数 inv で計算できます。行列の行列式は理論的な考察やある種の数式計算で利用可能ですが、そのスケーリングや丸め誤差の性質が、数値的な計算に信頼性を低下させます。その条件下で、関数 det は、正方行列の行列式を計算します。

```
A = pascal(3)

A =
      1     1     1
      1     2     3
      1     3     6

d = det(A)
X = inv(A)

d =
      1

X =
      3    -3     1
     -3     5    -2
      1    -2     1
```

再度、A は対称で、整数要素で、行列式が 1 であるので、逆行列の行列式も 1 になります。一方、

```
B = magic(3)

B =
      8     1     6
      3     5     7
      4     9     2
```

```
d = det(B)
X = inv(B)

d =
     -360

X =
      0.1472   -0.1444    0.0639
     -0.0611    0.0222    0.1056
     -0.0194    0.1889   -0.1028
```

X の要素の細かいチェック、または、有理形式 (format rat) を使用すると、これらは 360 である整数を割ったものになることがわかります。

A が正方で、正則ならば、丸め誤差を考えないで、X = inv(A)*B は、理論的には X = A\B と等価で、Y = B*inv(A) は Y = B/A と等価です、しかし、バックスラッシュとスラッシュを含む計算が好まれます。これは、計算時間や、メモリが小さく、よりエラーの検出が可能であるためです。

擬似逆行列

長方形行列が、逆行列または行列式をもたないことがあります。少なくとも、方程式 $AX = I$ と $XA = I$ のどちらか少なくとも一つは解をもちません。逆行列に対する部分置換は、*Moore-Penrose pseudoinverse* により与えられます。これは、関数 pinv で与えられます。

```
X = pinv(C)

X =
      0.1159   -0.0729    0.0171
     -0.0534    0.1152    0.0418
```

行列

```
Q = X*C

Q =
      1.0000    0.0000
      0.0000    1.0000
```

は、2 行 2 列の単位行列になりますが、行列

```
P = C*X
```

```
P =
       0.8293    -0.1958     0.3213
      -0.1958     0.7754     0.3685
       0.3213     0.3685     0.3952
```

は、3 行 3 列の単位行列ではありません。しかし、P は対称で、P*C は C に等しく、X*P は X に等しいと言う意味で、その空間の一部で単位行列のように機能します。

A が *m* > *n* の *m* 行 *n* 列でフルランク *n* の場合、つぎの 3 つのステートメント

```
x = A\b
x = pinv(A)*b
x = inv(A'*A)*A'*b
```

の結果は、同じ最小二乗解 x を理論的には計算します。しかし、バックスラッシュ演算が、より高速に処理されます。

しかし、A がフルランクでなければ、最小二乗問題に対する解はユニークでなくなります。つぎのノルム

```
norm(A*x -b)
```

を最小にする多くのベクトル x があります。x = A\b により計算される解は、*基本* 解と言われます。これは、*r* を A のランクとすると、高々 *r* 個の非ゼロの要素をもっています。x = pinv(A)*b により計算される解は、*最小ノルム* 解と言われます。この解は、norm(x) を最小にするものです。x = inv(A'*A)*A'*b を使って解を計算しようとする試みは、A'*A が特異なのでうまくいきません。

ここでは、種々の解を示す例題を示します。

```
A =
    [ 1  2  3
      4  5  6
      7  8  9
     10 11 12 ]
```

は、フルランクではありません。2 番目の列は、1 番目の列と 3 番目の列の平均になっています。

```
b = A(:,2)
```

が、2 番目の列ならば、A*x = b になる解は、x = [0 1 0]' になります。しかし、この x を計算するアプローチはありません。バックスラッシュ演算子は、

```
x = A\b

Warning: Rank deficient, rank = 2.

x =
      0.5000
      0
      0.5000
```

この解は、2 つのゼロでない要素をもっています。擬似逆行列のアプローチは、

```
y = pinv(A)*b

y =
      0.3333
      0.3333
      0.3333
```

ランクの欠落に対するワーニングはありません。しかし、norm(y) = 0.5774 は、norm(x) = 0.7071 よりも小さくなります。最終的に、

```
z = inv(A'*A)*A'*b
```

は全く、うまくいきません。

```
Warning: Matrix is singular to working precision.

z =
     Inf
     Inf
     Inf
```

LU, QR, コレスキ因子分解

MATLAB の線形方程式機能は、3 つの基本的な行列分解をベースにしています。

- 対称、正定行列に対しては、コレスキ分解
- 一般的な正方行列に対しては、LU 分解 (Gauss 消去法)
- 長方形行列に対しては、QR(直交化)

これら 3 つの因子分解は、関数 chol,lu,qr を使って行われます。

これらの 3 つの因子分解すべては、対角要素の上部または下部のどちらかのすべての要素がゼロである*三角* 行列を使います。三角行列を含む線形方程式システムは、*前置代入* または、*後退代入*のどちらかを使って、簡単に、容易に解かれます。

コレスキ因子分解

コレスキ因子分解は、対称行列を三角行列とその転置行列との積として表現します。

$$A = R'R$$

ここで、*R* は、上三角行列です。

対称行列すべてが、この方法で分解できることはありません。すなわち、因子分解による行列は*正定行列* です。これは、*A* の対角要素は正で、非対角要素は " あまり大きくない " ものです。Pascal 行列は、興味のある例題です。この章を通して、例題の行列 A は、3 行 3 列の Pascal 行列です。ちょっと、一時的ですが、6 行 6 列の行列を考えましょう。

```
A = pascal(6)

A =
     1     1     1     1     1     1
     1     2     3     4     5     6
     1     3     6    10    15    21
     1     4    10    20    35    56
     1     5    15    35    70   126
     1     6    21    56   126   252
```

A の要素は、二項係数になります。各々の要素は、その上と左の要素の和になります。コレスキ因子分解は、つぎのようになります。

```
R = chol(A)
```

```
R =
     1     1     1     1     1     1
     0     1     2     3     4     5
     0     0     1     3     6    10
     0     0     0     1     4    10
     0     0     0     0     1     5
     0     0     0     0     0     1
```

要素は、再び、二項係数になっています。R'*R が A に等しいことは、二項係数の積の和を含んでいることを示すものです。

注意 コレスキ因子分解は、複素数行列にも適用できます。コレスキ因子分解を行なった複素数行列は *A' = A* を満足し、*Hermitian positive definite* と言われます。

線形システム

$$Ax = b$$

は、

$$R'Rx = b$$

で置き換えることで、コレスキ因子分解を行なうことができます。バックスラッシュ演算子は、三角システムで使えるので、つぎのようにして簡単に解くことができます。

```
x = R\(R'\b)
```

A が、*n* 行 *n* 列の行列の場合、chol(A) の計算の誤差は $O(n^3)$ になりますが、連続バックスラッシュによる解の誤差は、たった $O(n^2)$ です。

LU 因子分解

LU 因子分解または Gauss 消去法は、正方行列 *A* を下三角行列の置換行列と上三角行列の積として表わします。

$$A = LU$$

ここで、*L* は、対角要素に 1 をもつ下三角行列を並べ替えたもので、*U* は上三角行列です。

並べ替えは、理論的にも、計算上の双方の理由から必要です。行列

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

は、2 つの行を交換しないでは、三角行列の積として表現できません。しかし、行列

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 0 \end{bmatrix}$$

は、三角行列の積として表すことができ、因子の中の小さな要素ε が大きいとき誤差は大きくなり、組み替えが厳密には必要ないとしても、これは望ましいものです。*部分的なピポット* は、*L* の要素が大きさで 1 に制限し、*U* の要素は *A* の要素より大きくなりません。

例えば、

```
[L,U] = lu(B)

L =
    1.0000         0         0
    0.3750    0.5441    1.0000
    0.5000    1.0000         0

U =
    8.0000    1.0000    6.0000
         0    8.5000   -1.0000
         0         0    5.2941
```

となります。A の LU 因子分解は、線形システム

```
A*x = b
```

を簡単に、迅速に解くために使います。

```
x = U\(L\b)
```

行列式と逆行列は、つぎの関係を使って、LU 因子分解から計算されます。

```
det(A) = det(L)*det(U)
```

と

```
inv(A) = inv(U)*inv(L)
```

です。行列式の符号が反対であっても、det(A) = prod(diag(U)) を使って行列式を計算することができます。

QR 因子分解

*直交*列、または、*直交性の列* をもつ行列は、各列が単位長さをもち、お互いが直交関係になる実数行列に分解されます。*Q* が直交であれば、

$$Q'Q = 1$$

になります。最も簡単な直交行列は、2 次元の座標回転変換に使うものです。

$$\begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

複素数行列に対して、対応する項は*ユニタリ* です。直交ユニタリ行列は、長さと角度が保存され、そのため誤差が大きくならないので、数値計算に対して望ましいものです。

直交または QR 因子分解は、任意の直交行列を一つの直交行列またはユニタリ行列と一つの上三角行列の積で表現します。列の並べ替えも含まれています。

$$A = QR$$

または

$$AP = QR$$

で、ここで、*Q* は直交またはユニタリ行列で、*R* は上三角行列で、*P* は並べ替え行列です。

QR 因子分解には、フルサイズ、エコノミサイズ、列置換をもつもの、もたないものと、4 つの種類があります。

過多線形システムは、列よりも多くの行をもつ長方形行列を含んでいます。すなわち、*m* 行 *n* 列で *m* > *n* です。*フル サイズ* QR 因子分解は、*m* 行 *m* 列の正方直交行列 *Q* と *m* 行 *n* 列上三角行列 *R* の積になります。

```
[Q,R] = qr(C)

Q =
   -0.8182    0.3999   -0.4131
   -0.1818   -0.8616   -0.4739
   -0.5455   -0.3126    0.7777

R =
  -11.0000   -8.5455
         0   -7.4817
         0         0
```

多くの場合、*Q* の最後の *m* - *n* 列は、*R* の下部にゼロを乗算することになるので、必要ありません。*エコノミサイズの QR 因子分解は、直交要素をもつ m* 行 *n* 列長方形行列 *Q* と正方 *n* 行 *n* 列の上三角行列 *R* の積で表されます。ここでの 3 行 2 列の例題では、あまり大きな影響はありません。しかし、非常に大きい長方形行列ならば、時間とメモリの節約になり、このことが最も重要なことになります。

```
[Q,R] = qr(C,0)

Q =
   -0.8182    0.3999
   -0.1818   -0.8616
   -0.5455   -0.3126

R =
  -11.0000   -8.5455
         0   -7.4817
```

LU 因子分解と比べ、QR 因子分解は、ピポットや並べ替えを必要としません。しかし、オプションの列の並び替えは、設定した三番目の引数に出力され、非正則を見つけたり、ランク落ちをを調べるのに有効です。因子分解の各々のステップで、

大きなノルムをもつ残りの分解されていない行列の列が、そのステップでの基底として使われます。これは、*R* の対角要素が減少する順番になり、列の間である線形依存性がこれらの要素を調べることにより、表現されます。ここで使っている小さな行列の例題に対して、C の 2 番目の列は 1 番目の列のノルムよりも大きいノルムをもちます。それで、二つの列を交換します。

```
[Q,R,P] = qr(C)

Q =
   -0.3522    0.8398   -0.4131
   -0.7044   -0.5285   -0.4739
   -0.6163    0.1241    0.7777

R =
  -11.3578   -8.2762
         0    7.2460
         0         0

P =
     0     1
     1     0
```

エコノミサイズと列の並び替えを組み合わせたとき、3 番目の引数は、並べ替え行列でなく、並べ替えベクトルになります。

```
[Q,R,p] = qr(C,0)

Q =
   -0.3522    0.8398
   -0.7044   -0.5285
   -0.6163    0.1241

R =
  -11.3578   -8.2762
         0    7.2460


p =
     2     1
```

QR 因子分解は、過多線形システムを等価な三角システムに変換します。表現

```
norm(A*x - b)
```

は、つぎの表現と等価です。

```
norm(Q*R*x - b)
```

直交行列の乗算は、ユークリッドノルムを保存するので、この表現は、つぎのものとも等価です。

```
norm(R*x - y)
```

ここで、y = Q'*b です。*R* の最後の *m-n* 行はゼロなので、この表現は、2 つの部分に分けられます。

```
norm(R(1:n,1:n)*x - y(1:n))
```

と

```
norm(y(n+1:m))
```

A がフルランクのとき、x に対して解くことができ、最初のものはゼロです。それで、2 番目のものは残差のノルムを与えます。A がフルランクでないとき、R の三角構造は、最小二乗問題に対する基本解を探します。

行列のベキと指数

この節は、つぎの行列のベキと指数を MATLAB の中で計算する方法を説明します。

- 正の整数
- 逆数と分数
- 要素単位
- 指数

正の整数のベキ

A が正方行列で、p が正の整数ならば、A^p は、A 自身を p 回乗算します。

```
X = A^2

X =
     3     6    10
     6    14    25
    10    25    46
```

逆数と分数のベキ乗

A が正方行列で、非正則ならば、A^(-p) は、inv(A) を p 回乗算します。

```
Y = B^(-3)

Y =
    0.0053   -0.0068    0.0018
   -0.0034    0.0001    0.0036
   -0.0016    0.0070   -0.0051
```

A^(2/3) のような分数のべき乗もまた可能です。結果は、行列の固有値の分布に依存します。

要素単位のベキ乗

要素単位のべき乗は、.^ で実行されます。例えば、

```
X = A.^2

A =
     1     1     1
     1     4     9
```

```
     1     9    36
```

指数

関数

```
sqrtm(A)
```

は、より正確なアルゴリズムを使って、A^(1/2) を計算します。sqrtm の m は、要素単位に機能する A.^(1/2) のような sqrt(A) と区別する関数です。

線形、定数係数、常微分方程式システムは、つぎにように表現されます。

$$dx/dt = Ax$$

ここで、$x = x(t)$ は t の関数ベクトルで、A は t に独立な行列です。解は、*行列指数*として表現されます。

$$x(t) = e^{tA}x(0)$$

関数

```
expm(A)
```

は、行列指数を計算します。つぎの 3 行 3 列の係数行列を考えます。

```
A =
     0    -6    -1
     6     2   -16
    -5    20   -10
```

そして、初期条件 $x(0)$ をつぎのように考えます。

```
x0 =
     1
     1
     1
```

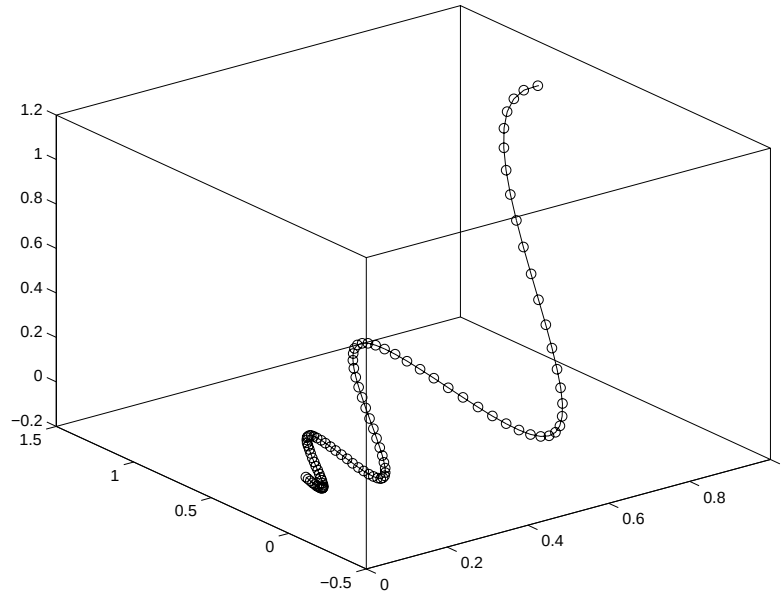
行列指数は、$0 \le t \le 1$ の範囲で、101 点で微分方程式の解 $x(t)$ を計算するために使います。

```
X = [];
for t = 0:.01:1
   X = [X expm(t*A)*x0];
end
```

3 次元位相空間プロットが得られます。

```
plot3(X(1,:),X(2,:),X(3,:),'-o')
```

これは、解が原点に向かって螺旋状になることを示しています。この挙動は、つぎの節で述べる係数行列の固有値に関連しています。



固有値

正方行列 A の*固有値* と*固有ベクトル*は、スカラλとベクトル v で表し、つぎの関係を満たします。

$$Av = \lambda v$$

この節は、つぎのことを説明します。

- 固有値分解
- (対角化されていない) フルランクでない行列に関連した問題
- 固有値分解に関連した問題を避けるため、Schur 分解の使い方

固有値分解

対角行列Λ の対角要素に固有値を配置し、対応する固有ベクトルからなる行列を V とすると、

$$AV = V\Lambda$$

の関係があります。*V が正則ならば、これは固有値分解*になります。

$$A = V\Lambda V^{-1}$$

前の節の常微分方程式の係数行列を考えます。

```
A =
     0    -6    -1
     6     2   -16
    -5    20   -10
```

ステートメント

```
lambda = eig(A)
```

は、固有値を含んだ列ベクトルを作成します。この例題の行列に対して、固有値は複素数になります。

```
lambda =
    -3.0710
    -2.4645+17.6008i
    -2.4645-17.6008i
```

各々の固有値の実数部は負で、$e^{\lambda t}$ は t が増加するときにゼロに近付きます。2 つの固有値のゼロでない虚数成分±ω は、微分方程式の解の中の振動成分 sin(ωt) に関連します。

2 つの出力引数を設定すると、eig は固有値ベクトルと固有値を計算し、対角行列の対角成分に固有値を出力します。

```
[V,D] = eig(A)

V =
  -0.8326          0.2003 - 0.1394i   0.2003 + 0.1394i
  -0.3553         -0.2110 - 0.6447i  -0.2110 + 0.6447i
  -0.4248         -0.6930             -0.6930

D =
  -3.0710                  0                  0
        0           -2.4645+17.6008i         0
        0                  0           -2.4645-17.6008i
```

最初の固有ベクトルは実数で、他の 2 つのベクトルはお互いに複素共役になっています。3 つのベクトルすべては、Euclidean 長で、norm(v,2) が 1 になるように正規化されています。

V*D/V のように、より簡潔に表現できる行列 V*D*inv(V) は、A の丸め誤差の範囲内に入ります。そして、inv(V)*A*V、または、V\A*V 共、D の丸めの範囲内に入ります。

フルランクでない行列

固有ベクトル分解できない行列もあります。これらの行列は、*フルランクでなく、対角化できないもの* です。例えば、

```
A =
     6    12    19
    -9   -20   -33
     4     9    15
```

この行列に対して、

```
[V,D] = eig(A)
```

は、つぎのものを出力します。

```
V =
```

```
   -0.4741   -0.4082   -0.4082
    0.8127    0.8165    0.8165
   -0.3386   -0.4082   -0.4082

D =

   -1.0000         0         0
         0    1.0000         0
         0         0    1.0000
```

λ = 1 に 2 つの固有値が存在しています。V の 2 番目と 3 番目の列は、同じです。この行列に対して、完全な線形的な独立な固有ベクトルの組みが存在していません。

オプションの Symbolic Math Toolbox は、強力な計算代数システムである Maple と関連付けて MATLAB の機能を拡張します。ツールボックスにより与えられる関数の一つは、Jordan Canonical 型を計算します。ここでの例題は、3 行 3 列の行列に対して、整数要素になることがわかります。

```
[X,J] = jordan(A)

X =
   -1.7500    1.5000    2.7500
    3.0000   -3.0000   -3.0000
   -1.2500    1.5000    1.2500

J =
    -1     0     0
     0     1     1
     0     0     1
```

Jordan Canonical Form は、重要な理論的な概念ですが、大きな行列や丸め誤差や他の不確かさを含む要素をもつ行列に対して、信頼性の高い計算ツールではありません。

MATLAB 行列計算の中での Schur 分解

MATLAB のアドバンスな行列計算は、固有値分解を必要としません。その代わりに、*Schur 分解*を使います。

$$A = U S U^T$$

ここで、*U は直交行列で*、*S* は対角要素が 1 行 1 列または 2 行 2 列のブロックを対角要素にもつブロック上三角行列です。固有値は、*S* の対角要素とブロックによって表わされます。一方 *U* の列は、固有ベクトルよりも数値的に良い性質をもつ基底を与えます。フルランクでないものの Schur 分解の例を示します。

```
[U,S] = schur(A)

U =
    0.4741   -0.6571    0.5861
   -0.8127   -0.0706    0.5783
    0.3386    0.7505    0.5675

S =
   -1.0000   21.3737   44.4161
         0    1.0081    0.6095
         0   -0.0001    0.9919
```

重根になる固有値は、S の下 2 行 2 列ブロックに含まれています。

特異値分解

長方形行列 A の*特異値* と対応する*特異ベクトル* は、つぎの式を満足するスカラ σ と u と v の一組のベクトルで表されます。

$$Av = \sigma u$$

$$A^T u = \sigma v$$

対角行列Σの対角要素で表す特異値と、それに対応し、2 つの直交行列の列からなる特異ベクトル U と Vは、つぎの関係があります。

$$AV = U\Sigma$$

$$A^T U = V\Sigma$$

U と V は、直交なので、これは*特異値分解*になります。

$$A = U\Sigma V^T$$

m行 n 列の行列のフル特異値分解は、m行 m 列の U と m行 n 列のΣ と n行 n 列の V になります。言い換えれば、U と V は共に正方で、Σは A と同じサイズです。A が列数より行数のほうが多いと、結果求まる U は非常に大きくなり、その列の大部分は、Σの中のゼロと乗算されるものになります。このような場合、*エコノミサイズ*の分解は、m行 n 列の Uと n行 n 列のΣと同じ大きさの Vを作成し、時間的にもスペース的にもセーブされます。

固有値分解は、あるベクトル空間からそれ自身への射影を表すとき、すなわち、常微分方程式に対して行うとき、行列を解析するために適切なツールです。一方、特異値分解は、一つのベクトル空間を他のベクトル空間への射影を解析する適切なツールです。これは、異なる次元についても取り扱えます。ほとんどの連立方程式システムは、この 2 番目のカテゴリに入ります。

A が正方で、対称、正定ならば、この固有値と特異値は等しくなります。しかし、A が対称、正定でなくなると、2 つの分解の結果のズレは大きくなります。特に、実数行列の特異値分解は、常に実数ですが、実数で非対称行列の特異値は複素数になることもあります。

例題の行列に対して、

```
A =
     9     4
     6     8
     2     7
```

フル特異値分解は、つぎのようになります。

```
[U,S,V] = svd(A)

U =
    -0.6105    0.7174    0.3355
    -0.6646   -0.2336   -0.7098
    -0.4308   -0.6563    0.6194

S =
   14.9359         0
         0    5.1883
         0         0

V =
    -0.6925    0.7214
    -0.7214   -0.6925
```

U*S*V' が、丸め誤差の範囲内で A に等しいことを確かめましょう。この小さな行列での問題に対しては、経済的な大きさでの分解が、わずかですが小さくなります。

```
[U,S,V] = svd(A,0)

U =
    -0.6105    0.7174
    -0.6646   -0.2336
    -0.4308   -0.6563

S =
   14.9359         0
         0    5.1883

V =
    -0.6925    0.7214
    -0.7214   -0.6925
```

U*S*V' は、丸め誤差の範囲内で A に等しいことを確かめてください。

12

多項式と内挿

多項式 12-3
多項式関数のまとめ 12-3
多項式の表現 12-4
多項式の根 12-4
特性多項式 12-5
多項式の計算 12-5
コンボリューションとデコンボリューション 12-5
多項式の微分 12-6
多項式のカーブフィッテング 12-7
部分分数展開 12-8

内挿 12-10
内挿関数のまとめ 12-10
1 次元内挿 12-11
2 次元内挿 12-13
内挿法の比較 12-13
内挿と多次元配列 12-16
散布データの三角メッシュと内挿 12-19
高次の散布データの三角メッシュと内挿 12-24

参考文献 12-34

この章では、多項式と共に機能し、1 次、2 次、多次元データを内挿する MATLAB 関数を 説明します。以下の項目を含みます。

多項式

標準の多項式演算用の関数を記述しています。加えて、フィッティングや部分分数展開に関するトピックスも含んでいます。

内挿

処理スピード、使用メモリ、平滑化を考えた内挿手法を記述しています。

多項式

この節では、つぎの事柄を説明します。

- MATLAB 多項式関数のまとめ
- MATLAB での多項式表現のサンプル

MATLAB 多項式関数についても記述しています。

- 多項式の根の計算
- 行列の特性多項式の係数の計算
- 設定した値での多項式の値
- 多項式のコンボリューション（乗算）とデコンボリューション（除算）
- 多項式の微係数の計算
- データセットへの多項式のフィット
- 部分分数展開と多項式係数との間の変換

多項式関数のまとめ

MATLAB は、多項式の根、多項式の値、差分のような標準の多項式演算に対する関数を提供します。それに加え、カーブフィッテング、部分分数展開のような、より進んだアプリケーションに対する関数も用意されています。

多項式関数は、MATLAB Toolbox の中のディレクトリ polyfun に保存されています。

多項式関数のまとめ

関数	詳細
conv	多項式の乗算
deconv	多項式の除算
poly	設定した値を根とする多項式
polyder	多項式微分
polyfit	多項式カーブフィッテング
polyval	多項式の値
polyvalm	行列をベースにした多項式の値

多項式関数のまとめ（続き）

関数	詳細
residue	部分分数展開
roots	多項式の根の算出

Symbolic Math Toolbox は、多項式演算に特別な機能を追加することができます。

多項式の表現

MATLAB は、次数の高い係数から行ベクトルとして多項式を表現します。例えば、つぎの方程式を考えましょう。

$$p(x) = x^3 - 2x - 5$$

これは、フランスアカデミで Newton 法を最初に表現するときに Wallis が使った例です。この多項式を MATLAB に入力するには、つぎのように行ないます。

```
p = [1 0 -2 -5];
```

多項式の根

関数 roots は、多項式の根を計算します。

```
r = roots(p)

r =
       2.0946
      -1.0473 +       1.1359i
      -1.0473 -       1.1359i
```

便宜的に、MATLAB は根を列ベクトルとして保存します。関数 poly は、多項式係数を出力します。

```
p2 = poly(r)

p2 =
     1    8.8818e-16    -2    -5
```

poly と roots は、並び順、スケーリング、丸め誤差まで逆関数の関係です。

特性多項式

関数 poly は、行列の特性多項式の係数も計算します。

```
A = [1.2 3 -0.9; 5 1.75 6; 9 0 1];
poly(A)

ans =
    1.0000   -3.9500   -1.8500  -163.2750
```

roots を使って計算した、この多項式の根は、行列 A の*特性根* または固有値と言われます (直接、行列の固有値を計算するために eig を使うこともできます)。

多項式の計算

polyval は、設定した値に対する多項式の値を計算します。例えば、$s = 5$ で多項式 p を計算します。

```
polyval(p,5)

ans =
   110
```

この計算は、行列に対しても可能です。この場合、$p(s) = x^3 - 2x - 5$ を $p(X) = X^3 - 2X - 5I$ と考えます。ここで、X は正方行列で、I は単位行列です。例えば、つぎのように正方行列 X を作成し、X に対する多項式 p の値を計算します。

```
X = [2 4 5; -1 0 3; 7 1 5];
Y = polyvalm(p,X)

Y =
   377   179   439
   111    81   136
   490   253   639
```

コンボリューションとデコンボリューション

多項式の乗算と除算は、コンボリューションとデコンボリューション演算に対応します。関数 conv と deconv が、この処理を行ないます。

多項式 $a(s) = s^2 + 2s + 3$ と $b(s) = 4s^2 + 5s + 6$ を考えます。これらの積を計算するため、

```
a = [1 2 3]; b = [4 5 6];
```

```
c = conv(a,b)

c =
     4    13    28    27    18
```

この積を$a(s)$で割り算するためにデコンボリューションを使います。

```
[q,r] = deconv(c,a)

q =
     4     5     6

r =
     0     0     0     0     0
```

多項式の微分

関数 polyder は、任意の多項式の微係数を計算します。多項式 p = [1 0 -2 -5] の微係数を得るには、つぎのようにします。

```
q = polyder(p)

q =
     3     0    -2
```

polyder は、2 つの多項式の積または商の微係数も計算できます。例えば、2 つの多項式 a と b を作ります。

```
a = [1 3 5];
b = [2 4 6];
```

積a*bの微係数を計算するには、出力引数を1つ設定して関数polyderを使います。

```
c = polyder(a,b)

c =
     8    30    56    38
```

商a/bの微係数を計算するには、出力引数を2つ設定して関数polyderを使います。

```
[q,d] = polyder(a,b)

q =
    -2    -8    -2
```

```
d =
     4     16     40     48     36
```

q/d が演算の結果になります。

多項式のカーブフィッテング

関数 polyfit は、最小二乗的にデータ群に適合する多項式の係数を出力します。

```
p = polyfit(x,y,n)
```

x と y は適合されるデータ群の *x* データ、*y* データで、n は出力される多項式の次数です。例えば、つぎの *x* と *y* のテストデータを考えます。

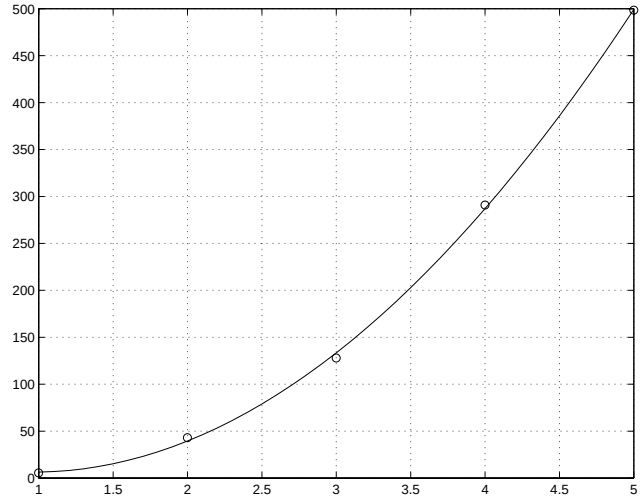
```
x = [1 2 3 4 5]; y = [5.5 43.1 128 290.7 498.4];
```

データを近似する多項式の次数を 3 にします。

```
p = polyfit(x,y,3)

p =
   -0.1917    31.5821   -60.3262    35.3400
```

狭い範囲で、polyfit 推定値が計算され、実データと比較するために重ねてプロットします。

```
x2 = 1:.1:5;
y2 = polyval(p,x2);
plot(x,y,'o',x2,y2)
grid on
```



これらの関数をアプリケーションの例題で用いるためには、" データ解析と統計量 " を参照してください。

部分分数展開

関数 residue は、2 つの多項式の比として部分分数展開を行ないます。これは、伝達関数でシステムを表すときに有効なものです。多重根をもたない多項式 b と a を考えます。

$$\frac{b(s)}{a(s)} = \frac{r_1}{s-p_1} + \frac{r_2}{s-p_2} + \ldots + \frac{r_n}{s-p_n} + k_s$$

ここで、r は余りの列ベクトルで、p は極の位置の列ベクトル、k は直達項の行ベクトルです。伝達関数を計算しましょう。

$$\frac{-4 + 8s^{-1}}{1 + 6s^{-1} + 8s^{-2}}$$

```
b = [-4 8];
a = [1 6 8];
[r,p,k] = residue(b,a)

r =
   -12
     8

p =
    -4
    -2

k =
     []
```

3 つの入力引数 (r, p, k) を使って、residue は多項式型に変換します。

```
[b2,a2] = residue(r,p,k)

b2 =
    -4     8

a2 =
     1     6     8
```

内挿

内挿は、既知のデータ点間の値を推定するプロセスです。このような作業は、信号処理、イメージ処理等の分野で重要なアプリケーションになります。

この節では、つぎのような事柄が含まれています。

- MATLAB 内挿関数のまとめ
- 1 次元内挿に関する議論
- 2 次元内挿に関する議論
- 内挿手法である、nearest neighbor, bilinear, bicubic 法の比較を例題を使って実行
- 多次元データに関する内挿
- 散布データの三角形の作成や内挿

内挿関数のまとめ

MATLAB は、実行速度やメモリ使用とデータフィットの平滑化とのバランスを考えた数種類の内挿手法を用意しています。

内挿関数は、MATLAB Toolbox の polyfun と呼ばれるディレクトリに用意しています。

内挿関数のまとめ

関数	詳細
griddata	データのグリッド化とサーフェス近似
griddata3	3 次元データに対するデータのグリッド化と超平面近似
griddatan	(次元 >= 3) のデータのグリッド化と超平面近似
interp1	1 次元内挿 (テーブルルックアップ)
interp2	2 次元内挿 (テーブルルックアップ)
interp3	3 次元内挿 (テーブルルックアップ)
interpft	FFT 法を使った 1 次元内挿
interpn	N 次元内挿 (テーブルルックアップ)

内挿関数のまとめ（続き）

関数	詳細
pchip	区分的キュービックエルミート内挿多項式 (PCHIP)
spline	キュービックスプラインデータ内挿

1 次元内挿

MATLAB では、2 種類の 1 次元内挿法を用意しています。

- 多項式内挿
- FFT ベースの内挿

多項式内挿

interp1 は、データ解析やカーブフィッテングに重要な演算である 1 次元内挿を行ないます。この関数は、与えられたデータに対して、データ間を多項式で近似し、希望する内挿点で適切な関数を推定します。この最も一般的な型は

```
yi = interp1(x,y,xi,method)
```

y は、ある関数値を含むベクトルで、x は、y の値が与えられる点を含む y と同じ長さのベクトルです。xi は、内挿される点を含むベクトルです。*method* は、内挿法を設定するオプションの文字列です。

- *Nearest neighbor interpolation* (method = 'nearest'). この方法は、内挿する点の最も近くにあるデータを使います。
- *Linear interpolation* (method = 'linear'). この方法は、存在しているデータ点のお互いの組毎に別々の関数をフィットさせ、xi で設定されている点で、その関数値を計算して出力します。これは、interp1 関数に対するデフォルトの方法です。
- *Cubic spline interpolation* (method = 'spline'). この方法は、存在しているデータ点の各組の間を種々のキュービック関数でフィットさせ、関数 spline を使って、データ点でキュービックスプライン内挿を行います。
- *Cubic interpolation* (method = 'pchip' または 'cubic'). これらの方法は、同じものです。関数 pchip を使って、ベクトル x と y とを区分的なキュービックエルミート内挿を行います。これらの方法は、単調で、データの型を保ちます。

xi の任意の要素が、x で張られる区間の外に位置する場合、設定した内挿法は、外挿を実行するものとして使われます。また、yi = interp1(x,Y,xi,method,extrapval) は、外挿された値を extrapval で置き換えます。NaN が、extrapval に対して、しばしば使われます。

すべての方法は、一様分布していないデータに対しても機能します。

スピード、メモリ、スムージング

内挿法を選択すると、他の処理より計算時間とスペースが必要になることは覚えておいてください。また、これらの必要な事柄と結果の求まるスムージング状態との間でトレードオフを行なうことが必要です。

- Nearest neighbor interpolation は、最も高速な方法です。しかし、スムージングと言う意味では、最も悪い結果を出力します。
- Linear interpolation は、nearest neighbor 法よりも多くのメモリを使い、実行時間もわずかですが多くなります。Nearest neighbor 内挿法と異なり、その結果は連続ですが、頂点で勾配が変わります。
- Cubic spline interpolation は、実行時間を比較的必要とします。しかし、Cubic 内挿よりはメモリを必要としません。この手法は、すべての内挿法の中で最も平滑なものを作成します。しかし、入力データが一様でなかったり、ある部分に他の部分よりデータが集中していると、予期せぬ結果を出力する場合があります。
- Cubic interpolation は、nearest neighbor 法や linear 法のどちらよりもメモリおよび実行時間を必要とします。しかし、内挿されたデータやその微係数は共に連続です。

各々の方法の相対的な性能は、2 次元または多次元の内挿に対してさえも同じ結果になります。内挿法のグラフィカルな比較は、12-13 ページの " 内挿法の 比較 " を参照してください。

FFT ベースの内挿

関数 interpft は、FFT をベースにした 1 次元の内挿を行ないます。この方法は、周期的な関数の値を含むベクトルの Fourier 変換を計算します。それで、より多くの点を使って、逆 Fourier 変換を計算します。この書式は、

```
y = interpft(x,n)
```

です。x は、等間隔にサンプルされた周期的な関数値を含むベクトルです。n は、出力する等間隔な点の数です。

2 次元内挿

関数 interp2 は、イメージ処理やデータの可視化に重要な演算である 2 次元の内挿を行ないます。最も一般的な型は、

```
ZI = interp2(X,Y,Z,XI,YI,method)
```

です。Z は、2 次元関数の値を含んだ長方形配列で、X と Y は、同じ大きさの配列で、Z の値を与える点を表します。XI と YI は、データを内挿する点を含んだ行列です。method は、内挿法を設定するオプションの文字列です。

2 次元データに対して、3 つの異なる内挿法があります。

- *Nearest neighbor interpolation* (method = 'nearest'). この方法は、データ値を通る区分的に一定なサーフェスで近似します。内挿される点の値は、最も近くのものを使います。
- *Bilinear interpolation* (method = 'linear'). この方法は、存在するデータ点を通して双一次サーフェスで近似します。内挿された点の値は、4 つの最も近い値を組み合わせたものです。この方法は、区分的に双一次で、二次の内挿より速く、メモリを使いません。
- *Bicubic interpolation* (method = 'cubic'). この方法は、存在する点を二次平面で近似します。内挿された値は、近傍の 16 点を組み合わせて作成されます。この方法は、区分的に二次型で、線形内挿より、よりスムーズな表面を作成します。これは、イメージ処理のようなアプリケーションに対して、重要なものです。二次キュービック内挿は、内挿されたデータとその微係数が連続であることが条件になります。

これらのすべての方法は、X と Y が単調である必要があります。つまり、点の点の間で常に増加か、あるいは減少でなければなりません。これらの行列は、関数 meshgrid を使って準備するか、または、meshgrid の出力に相当する点のパターンにするかのどちらかを行ってください。さらに、各方法は、内挿の前に入力を等間隔の領域に自動的に写像します。X と Y が既に等間隔の場合は、たとえば '*cubic' のようにメソッド文字列の前にアスタリスクを付加することにより、実行速度を短縮することができます。

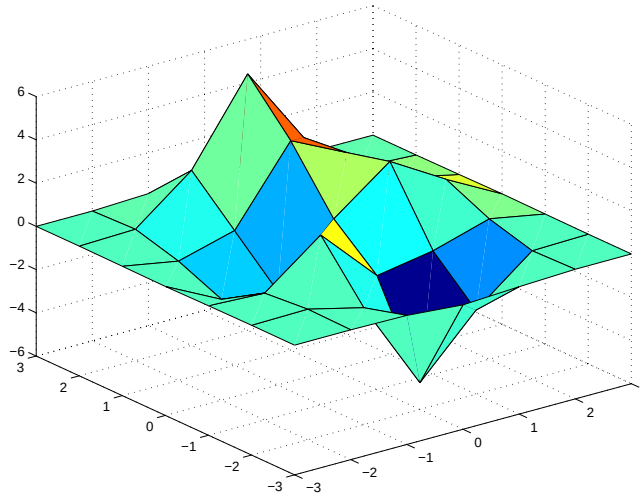
内挿法の 比較

この例題は、7 行 7 列のデータ行列に 2 次元内挿法を適用したものを比較します。

1 解像度の peaks 関数を作成します。

```
[x,y] = meshgrid(-3:1:3);
```

```
z = peaks(x,y);
surf(x,y,z)
```



2 内挿を使って、より細かいメッシュを作成します。

```
[xi,yi] = meshgrid(-3:0.25:3);
```

3 nearest neighbor interpolation を使って内挿します。

```
zi1 = interp2(x,y,z,xi,yi,'nearest');
```

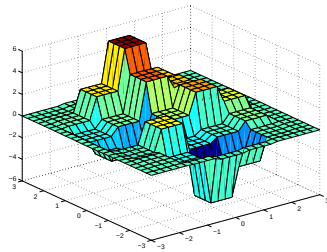
4 bilinear interpolation を使って内挿します。

```
zi2 = interp2(x,y,z,xi,yi,'bilinear');
```

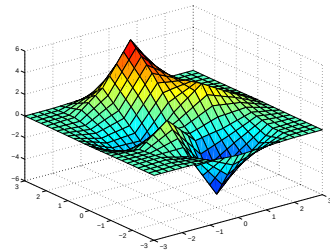
5 bicubic interpolation を使って内挿します。

```
zi3 = interp2(x,y,z,xi,yi,'bicubic');
```

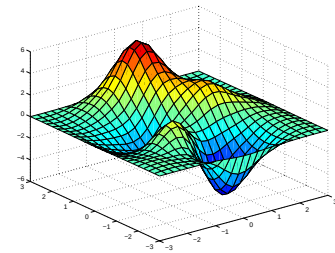
6 種々の内挿法による結果をプロット表示し、比較しましょう。



surf(xi,yi,zi1)
% nearest

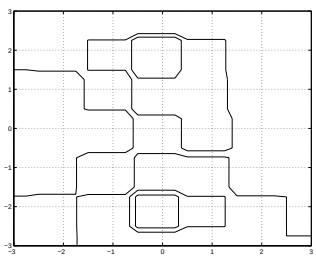


surf(xi,yi,zi2)
% bilinear

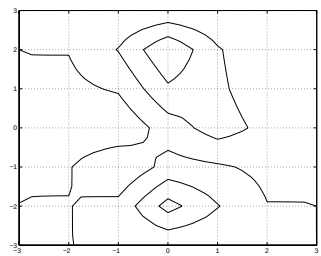


surf(xi,yi,zi3)
% bicubic

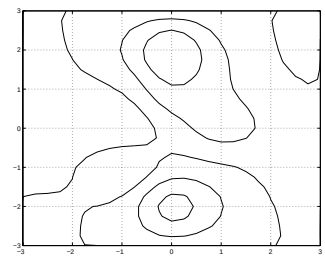
7 種々の内挿法のコンターマップを比較しましょう。



contour(xi,yi,zi1)
% nearest



contour(xi,yi,zi2)
% bilinear



contour(xi,yi,zi3)
% bicubic

特に、bicubic 法は、スムーズなコンターを表示します。これは、常に重要なことではありません。しかし、医学的なイメージ処理のようなアプリケーションにおいては、nearest neighbor のような方法が、新しいデータ値を生みださないので好まれます。

内挿と多次元配列

いくつかの内挿関数は、多次元データにも機能します。

多次元データの内挿関数

関数	詳細
interp3	3 次元データの内挿
interpn	多次元データの内挿
ndgrid	多次元データのグリッド化 (ndfun ディレクトリ)

この節では、つぎの事柄を説明します。

- 3 次元データの内挿
- より高次元データの内挿
- 多次元のデータのグリッド化

3 次元データの内挿

関数 interp3 は、3 次元のサンプル値 V の間の値を探す 3 次元内挿を行ないます。ユーザは、既知のデータ点をもっていなければなりません。

- X, Y, Z 行列は、与えられた値 V の位置を設定します。
- 行列 V は、X, Y, Z の点に対応する値を含んでいます。

interp3 に対する最も一般的な型は、s

```
VI = interp3(X,Y,Z,V,XI,YI,ZI,method)
```

です。ここで、XI, YI, ZI は、関数 interp3 が V を内挿する点を示すものです。範囲外のものについては、interp3 は NaN を出力します。

3 次元データの内挿には、3 種類のものがあります。

- *Nearest neighbor interpolation* (method = 'nearest'). この方法は、最も近傍の点を使います。
- *Trilinear interpolation* (method = 'linear'). この方法は、近傍の 8 点を使って区分的に線形な内挿を行ないます。
- *Tricubic interpolation* (method = 'cubic'). この方法は、近傍 64 点の値を使って区分的にキュービックな内挿を行ないます。

これらのすべての方法は、X, Y, Z が*単調* である必要があります。つまり、特定の方向において常に増加であるか、あるいは減少です。さらに、これらの行列は、関数 meshgrid を使うか、meshgrid の出力に相当する点の " パターン " にするかのどちらかを行ってください。

個々の方法は、内挿を行なう前に入力を等間隔の領域に射影します。x が既に等間隔になっているならば、method 文字の前に * を付けて実行時間を短縮することができます。例えば、'*cubic' です。

高次元データの内挿

関数 interpn は、多次元のサンプルデータ V の点間の内挿値を求めるために多次元内挿を行ないます。interpn の最も一般形は、

```
VI = interpn(X1,X2,X3...,V,Y1,Y2,Y3,...,method)
```

です。1, 2, 3, ... は、与えられた V の値に対する点を設定する行列です。V は、これらの点に対応する値を含んでいる行列です。1, 2, 3, ... は、interpn が V の内挿された値を出力する点です。範囲外の値に対しては、NaN を出力します。

Y1, Y2, Y3, ... は、同じ大きさの配列かまたはベクトルのどちらかになります。これらが異なる大きさのベクトルならば、interpn は、これらを ndgrid に転送し、その後、結果の並列を使います。

多次元データに対して 3 つの異なる方法があります。

- *Nearest neighbor interpolation* (method = 'nearest'). この方法は、最も近傍な値を使います。
- *Linear interpolation* (method = 'linear'). この方法は、個々の次元で最も近傍の 2 つの値を使って区分的に線形な内挿を行ないます。
- *Cubic interpolation* (method = 'cubic'). この方法は、各々の次元で近傍の 4 つの値を使って区分的キュービックな内挿を行ないます。

これらすべての方法は、X1, X2,X3 は単調でなければなりません。加えて、ndgrid 関数を使ってこれらの行列を準備するか、または ndgrid の出力に相当する点のパターンを用意するかのどちらかが必要です。

各々の方法は、内挿が行われる前に入力を等間隔の領域に射影する必要があります。X が既に等間隔になっているならば、method 文字変数の先頭に * を付けることにより実行時間をセーブすることができます。例えば、'*cubic' です。

多次元データのグリッド化

関数 ndgrid は、多次元関数の関数値や内挿にデータの配列を作成します。ndgrid は、一連の入力ベクトルにより設定される領域を一連の出力配列に変換します。これらの出力配列の i 番目の次元は、入力ベクトル x_i の要素のコピーです。

ndgrid の書式は

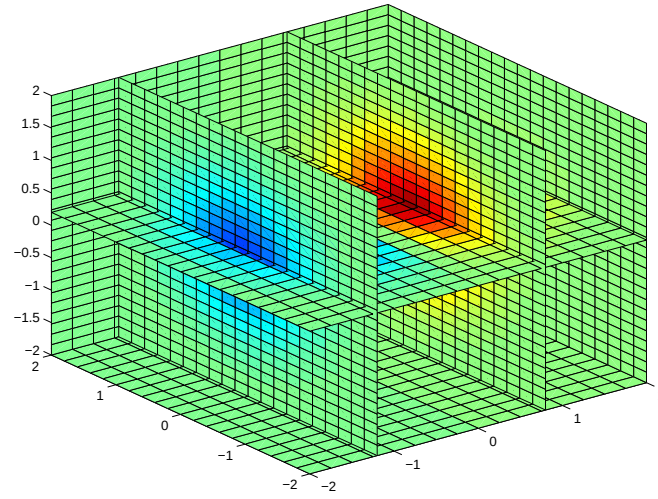
```
[X1,X2,X3,...] = ndgrid(x1,x2,x3,...)
```

例えば、与えられた範囲に渡り、3 つの変数からなる関数を計算しようとします。つぎの関数を考えます。

$$z = x_2 e^{(-x_1^2 - x_2^2 - x_3^2)}$$

ここで、$-2\pi \le x_1 \le 0$ ， $2\pi \le x_2 \le 4\pi$ ， $0 \le x_3 \le 2\pi$ の範囲で考えます。この関数を計算し、プロットするには、つぎのようにします。

```
x1 = -2:0.2:2;
x2 = -2:0.25:2;
x3 = -2:0.16:2;
[X1,X2,X3] = ndgrid(x1,x2,x3);
z = X2.*exp(-X1.^2 -X2.^2 -X3.^2);
slice(X2,X1,X3,z,[-1.2.8 2],2,[-2 0.2])
```



散布データの三角メッシュと内挿

MATLAB は、最近傍点問題の解析と幾何解析に手助けになるルーチンを用意しています。

最近傍点の解析と幾何解析のための関数

関数	詳細
convhull	凸包
delaunay	Delaunay 三角形作成
delaunay3	3- 次元の Delaunay 三角形分割
dsearch	最近傍点に対する Delaunay 三角形の探索
inpolygon	多角形内部に点が存在すると真
polyarea	多角形の面積
rectint	2 つまたはそれ以上の長方形に囲まれた面積
tsearch	最近傍の三角形の探索
voronoi	Voronoi ダイアグラム

この節は、MATLAB で与える seamount データセットに、つぎの手法を適用します。

- Delaunay 三角形
- Voronoi ダイアグラム
- 凸包

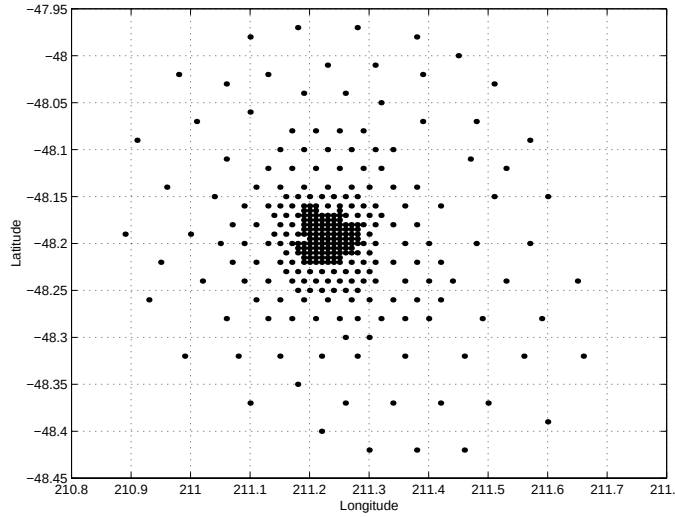
12-24 ページの " 高次の散布データの三角メッシュと内挿 " も参照してください。

Delaunay 三角形

関数 delaunay は、ある三角形の領域に含まれているけれども、実際にデータとして存在しない点を一組の三角形として出力します。delaunay を実行するために、MATLAB で用意されている seamount データをロードし、簡単なバラツキのプロットが表示されます。

```
load seamount
plot(x,y,'.','markersize',12)
xlabel('Longitude'), ylabel('Latitude')
```

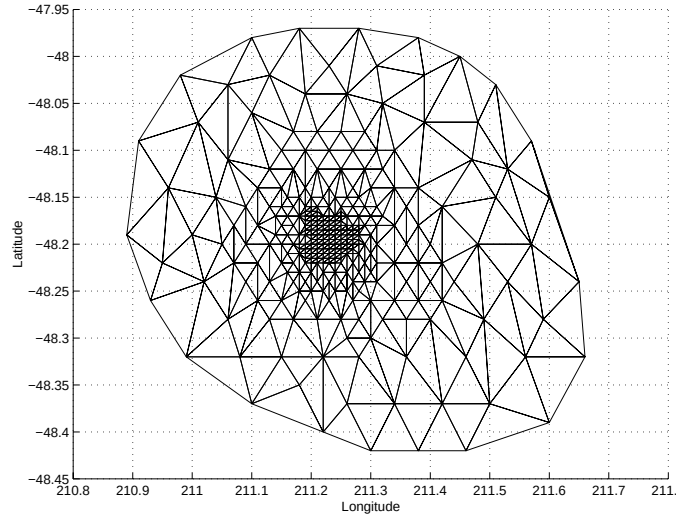
```
grid on
```



注意 seamount については、Parker [1], pp 17-40 を参照してください。

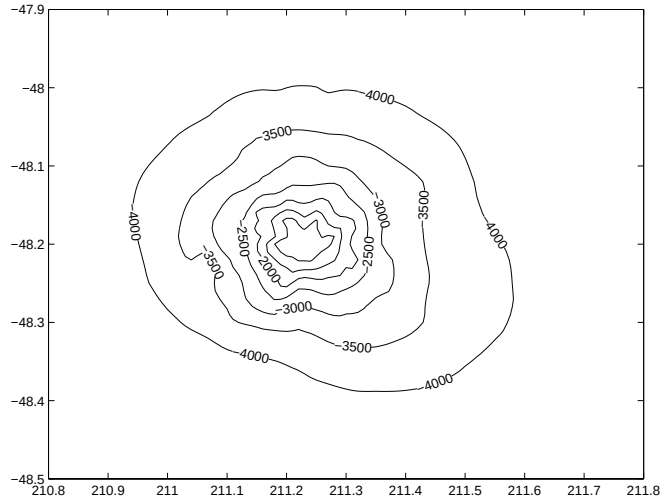
Delaunay三角法を適用し、バラツキのあるデータのプロット図に重ね書きします。

```
tri = delaunay(x,y);
hold on, trimesh(tri,x,y,z), hold off
hidden off; grid on
xlabel('Longitude'); ylabel('Latitude')
```



コンタプロットも示します。

```
[xi,yi] = meshgrid(210.8:.01:211.8,-48.5:.01:-47.9);
zi = griddata(x,y,z,xi,yi,'cubic');
[c,h] = contour(xi,yi,zi,'c-'); clabel(c,h)
```



meshgrid に対する引数は、オリジナルの seamount データの中の x と y の最大値と最小値の範囲を含んでいます。これらのデータを得るには、つぎのステートメントを使ってください。

```
min(min(x))
max(max(x))
```

と

```
min(min(y))
max(max(y))
```

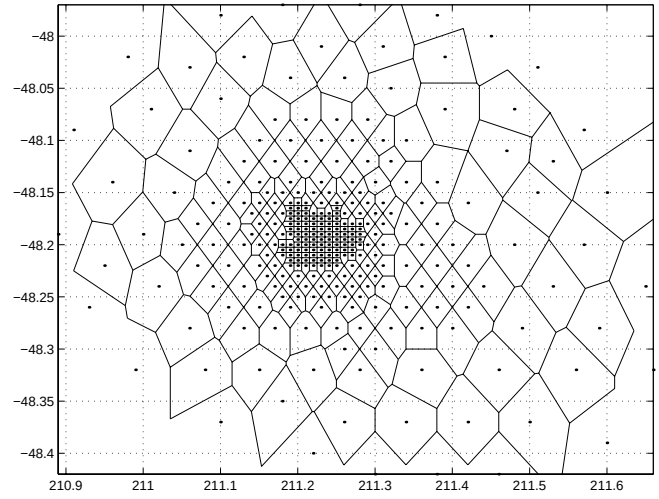
最近傍点の探索． つぎの2つの関数を使って、Delaunay三角法データから求めます。

- dsearch は、ユーザが設定した点に最も近い点を求めます。
- tsearch は、ある点 (xi,yi) を与えて、それを囲む三角形を設定する delaunay 出力へのインデックスを求めます。

Voronoi ダイアグラム

Voronoi ダイアグラムは、Delaunay 三角法に関連した最近傍点プロット法です。seamount データに対して、Voronoi ダイアグラムは、つぎのステートメントで求めます。

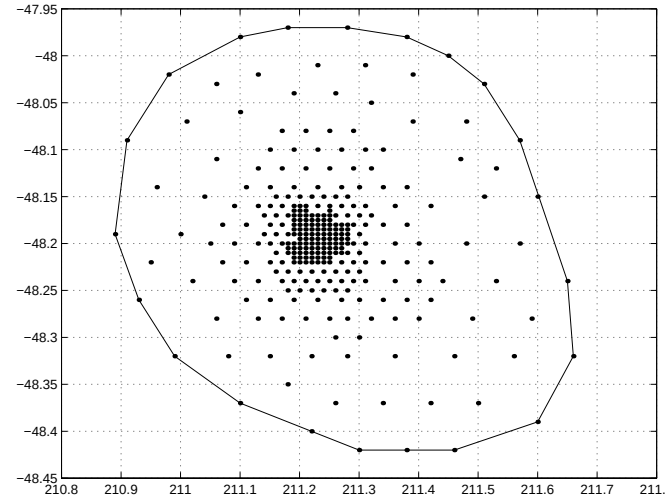
```
load seamount
voronoi(x,y)
grid on
```



凸包

convhull は、データ集合に対して、凸部を構成する点のインデックスを出力します。例えば、seamount データに対して、凸部を構成する部分を表示しましょう。

```
load seamount
plot(x,y,'.','markersize',10)
k = convhull(x,y);
hold on, plot(x(k),y(k)), hold off
grid on
```



高次の散布データの三角メッシュと内挿

科学、工学、統計、数学における多くのアプリケーションは、凸包、Voronoi ダイアグラム、Delaunay 三角形分割のような構造を必要とします。MATLAB 関数を使って一般的な次元、たとえば、3,4, あるいは高次についてこれらの構造を計算することができます。

多次元計算用関数

関数	詳細
convhulln	n- 次元の凸包
delaunayn	n- 次元の Delaunay 三角形分割

多次元計算用関数

関数	詳細
dsearchn	n- 次元の最近傍点探索
griddatan	n- 次元のデータグリッドとハイパーサーフェス近似
tsearchn	n- 次元の最近傍点に対する三角形の探索
voronoin	n 次元の Voronoi ファセット

この節では、以下の構造を計算しグラフ化します。

- 凸包
- Delaunay 三角形分割
- Voronoi ダイアグラム

以下についてもデモンストレーションします。

- N 次元データの内挿

凸包

d- 次元空間 R^d のサブセット S に対して、凸包 c(S) は S を含む R^d の最小の convex として定義されます。

凸包の計算は、R^d の有限個の n 点 $S = \{p_1, p_2, \ldots, p_n\}$ に対して c(S) を決定することと同じです。

c(S) を決定する通常の方法は、半空間の交点として表わすことです。あるいは、より正確には、線形不等式の最小システムの解として表わすことです。これは、c(S) = $\{x \mid Ax \le b\}$ のような m に対して、行列 $A \in R^{m \times d}$ とベクトル $b \in R^m$ を計算することと同じです。

凸包の計算． X が立方体の 8 個の頂点からなる 8 行 3 列行列と仮定します。X の凸包は、立方体の12の面で構成されます。convhulln を使ってこの凸包を見てみます。

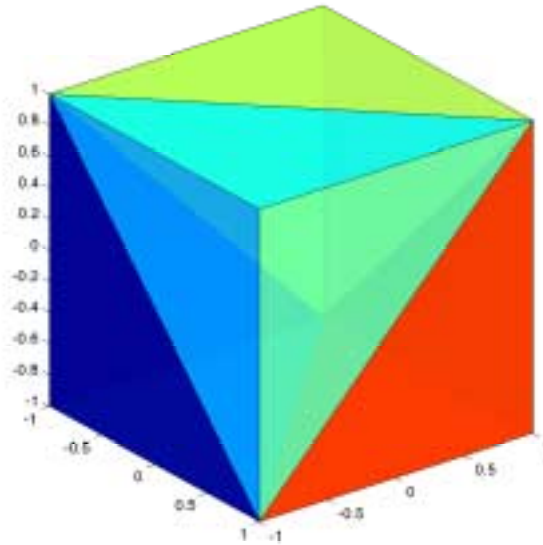
```
d = [-1 1];
[x,y,z] = meshgrid(d,d,d);
X = [x(:),y(:),z(:)];       % 8 corner points of a cube
tri = convhulln(X)

tri =
```

```
     3     1     5
     1     2     5
     2     1     3
     7     3     5
     8     7     5
     7     8     3
     6     8     5
     2     6     5
     6     2     8
     8     4     3
     4     2     3
     2     4     8
```

tri の 12 行は、12 個の三角形を表わします。数値は、X 内の点のインデックスです。たとえば第一の行、3 1 5 は、第一の三角形が頂点として X(3,:), X(1,:), X(5,:) をもつことを意味します。

3 次元の patch を使ってこの凸包を表示します。

```
figure, hold on
d = [1 2 3 1];
for i = 1:size(tri,1)
    c = tri(i,d);
    h(i)=patch(X(c,1),X(c,2),X(c,3),i,'FaceAlpha',0.9);
end                % 'FaceAlpha' is used to make it transparent.
hold off
view(3), axis equal
camorbit(90,-5);    % To view it from another angle
```



Delaunay 三角形

点の集合の Delaunay 三角形は、集合内の点が 4 面体に外接する球の内部に点が含まれないような性質をもつものです。

Delaunay 三角形分割は、面に含まれるものを反転させた 2 つの複素数の間の全単射であるという意味で二重の Voronoi セルです。

Delaunay 三角形分割の計算． この例題は、凸包の例題と同じ X を用います。立方体の 8 隅の点と中心です。

```
d = [-1 1];
[x,y,z] = meshgrid(d,d,d);
X = [x(:),y(:),z(:)];   % 8 corner points of a cube
X(9,:) = [0 0 0];       % Add center to the vertex list.
tri = delaunayn(X)      % Generate Delaunay tessellation.

tri =
     9     7     3     5
     1     9     3     5
```

```
     1     2     9     5
     4     9     7     3
     4     9     7     8
     4     1     9     3
     4     1     2     9
     6     2     9     5
     6     9     7     5
     6     9     7     8
     6     4     9     8
     6     4     2     9
```

tri の 12 行は、立方体を分割する 12 個の 4 面体を表わします。convhulln の例題のように、各行は、1 つの 4 面体で、数値は X のポイントのインデックスです。

3 次元の patch で見てみます。

```
figure, hold on
d = [1 1 1 2; 2 2 3 3; 3 4 4 4];
for i = 1:size(tri,1)
   y = tri(i,d);
   x1 = reshape(X(y,1),3,4);
   x2 = reshape(X(y,2),3,4);
   x3 = reshape(X(y,3),3,4);
   h(i)=patch(x1,x2,x3,(1:4)*i,'FaceAlpha',0.9);
end
hold off
view(3), axis equal
axis off
camorbit(65,120)   % To view it from another angle
```

cameramenu を使って任意の方向に図を回転することができます。

Voronoi ダイアグラム

R^d の n 個の点の集合 S を与えた場合、Voronoi ダイアグラムは n 個の多面体の領域 $\mathrm{vo}(p)$ への R^d のパーティションです (p in S). 各領域 $\mathrm{vo}(p)$ は、p の Voronoi セルと呼ばれ、S の他の点よりも p に近い R^d の点として定義されます。あるいは、より詳しくは、以下のようになります。

$$\mathrm{vo}(p) = \{x \text{ in } R^d \mid \mathrm{dist}(x, p) \leq \mathrm{dist}(x, q) \text{ for } q \text{ in } S\}$$

ここで、dist は、標準の Euclidean 距離関数です。

注意 他の Voronoi ダイアグラムを定義するため、別の距離関数を使うことは可能ですが、ここでは説明しません。

すべての Voronoi セルとそれらの面がセル complex を形成します。この complex の頂点は、Voronoi 頂点と呼ばれ、extreme rays(境界のないエッジ) は Voronoi rays です。

Voronoi ダイアグラムの計算． 凸包や Delaunay の計算と異なり、Voronoi の計算は、2 つの出力を生成します。1 つは、新規に生成される Voronoi center で、もう 1 つは、Voronoi セルのファセットです。

- 新規に生成される center は、*d*- 空間の *m* 点で構成されます。点は、*m* 行 *d* 列行列として表わされ、その値は新規に生成された低の座標です。
- 一方、ファセットは、まったく不規則です。ファセットは、3 面、4 面等をもちます。この不規則性のため、ファセットは、通常の行列ではなくセル配列を使って表わされます。セル配列の各要素は、入力点の 1 つに対応する Voronoi セルに対して新規に生成されたデータのインデックスを含みます。

これらの計算は、Deluanay の例題と同じ X を使います。立方体の 8 隅の点と中心です。

```
d = [-1 1];
[x,y,z] = meshgrid(d,d,d);
X = [x(:),y(:),z(:)];   % 8 corner points of a cube
X(9,:) = [0 0 0];       % Add center to the vertex list.
[c,v] = voronoin(X);

c =

      Inf      Inf      Inf
   0.5000   0.5000   1.2500
   0.5000   0.5000   1.2500
   0.5000  -0.2500   0.5000
  -0.2500   0.5000   0.5000
   0.5000  -0.2500   0.5000
   1.2500   0.5000   0.5000
   1.2500   0.5000   0.5000
   0.5000   0.5000  -0.2500
   0.5000   0.5000  -0.2500
  -0.2500   0.5000   0.5000
   0.5000   1.2500   0.5000
   0.5000   1.2500   0.5000

v =

   [1x5 double]
   [1x5 double]
   [1x6 double]
```

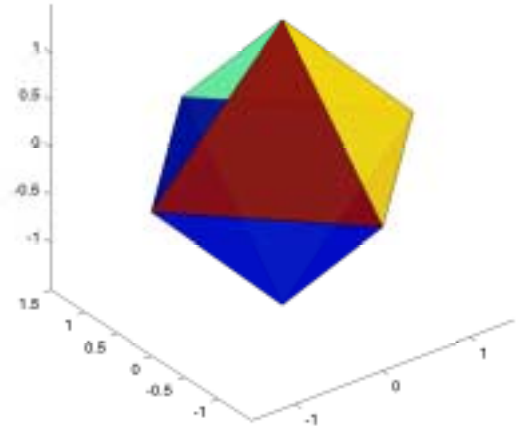
```
    [1x6 double]
    [1x6 double]
    [1x6 double]
    [1x5 double]
    [1x5 double]
    [1x12 double]
```

c は 13 行 3 列の行列で、13 行は、13 個の新規に生成された Voronoi 点の座標です。c の最初の要素は、通常 Inf で、これは、ポイントが無限大にあることを意味します。たとえば、v は 9 行 1 列のセル配列で、9 個の Voronoi セルを表わします。

```
v{9} = 2 3 4 5 6 7 8 9 10 11 12 13
```

1 (無限大を意味する) を含まない Voronoi セルを表示するには、そのセルのインデックスをもつ新規に生成された点について関数 convhulln を使うことができます。たとえば、上記の例題の 9 番目の Voronoi セルを見るためには、凸包で述べた手順を使います。

```
X = c(v{9},:);
tri = convhulln(X);
figure
hold on
d = [1 2 3 1];
for i = 1:size(tri,1)
   c = tri(i,d);
   h(i)=patch(X(c,1),X(c,2),X(c,3),i,'FaceAlpha',0.9);
end
hold off
view(3)
axis equal
```

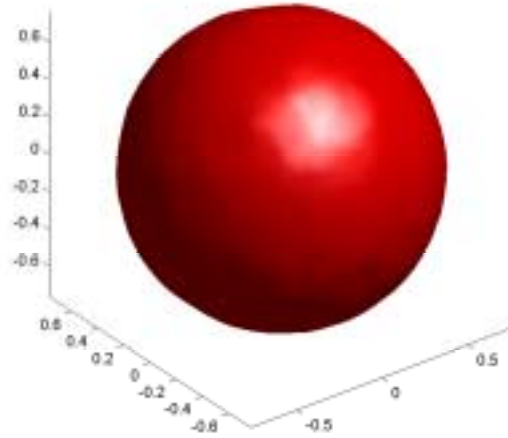


N- 次元データの内挿

X が 3 次元空間の散布データの n 行 3 列の行列と仮定し、V はそれらの対応する値のベクトルと仮定します。X と V は、4 次元空間のハイパーサーフェスを形成します。この例題は、griddatan を使って X 内の点ではなく点 X0 でのハイパーサーフェスの値を検索します。結果は、球 $x^2 + y^2 + z^2 = \text{const}$ です。この例題は、実行に少し時間がかかります。

```
n = 5000;
X = 2*rand(n,3)-1;
v = sum(X.^2,2);
d = -1:0.05:1;
[x0,y0,z0] = meshgrid(d,d,d);
X0=[x0(:) y0(:) z0(:)];
v0 = reshape(griddatan(X,v,X0),size(x0));
p = patch(isosurface(x0,y0,z0,v0,0.6));
isonormals(x0,y0,z0,v0,p)
set(p,'FaceColor','red','EdgeColor','none');
view(3);
```

```
camlight;
lighting phong
axis equal
```



参考文献

[1] Parker, R. L., L. Shure, & J. Hildebrand, ｢The Application of Inverse Theory to Seamount Magnetism.｣ *Reviews of Geophysics*. Vol 25, 1987.

13

データ解析と統計量

列方向のデータセット 13-4

基本的なデータ解析関数 13-8
関数のまとめ 13-8
共分散係数と相関係数 13-11
有限差分 13-12

データの前処理 13-14
欠測値 13-14
異常値の除去 13-15

回帰とカーブフィッティング 13-17
多項式回帰 13-17
パラメータが線形である方程式の回帰 13-19
多変数回帰 13-20

ケーススタディ：カーブフィッティング 13-22
多項式近似 13-22
残差の解析 13-24
指数近似 13-27
誤差の範囲 13-30
Basic Fitting インタフェース 13-31

差分方程式とフィルタリング 13-41

フーリエ解析と高速フーリエ変換 (FFT) 13-43
関数のまとめ 13-43
はじめに 13-44
変換されたデータのゲインと位相 13-49
FFT の長さとスピード 13-50

この章は、MATLAB のデータ解析機能の紹介です。ここで、データ解析に対して配列をどのように作成するか、簡単な記述的統計関数をどのように使うか、データの前処理を MATLAB でどのように行なうかを記述します。また、最小二乗、カーブフィッティング、データのフィルタリング、高速 Fourier 変換 (FFT) を含む、データ解析トピックスについても記述します。

列方向のデータセット
データ解析用の配列の作成

基本的なデータ解析関数
基本的なデータ解析関数と関数のいくつかを使った例題を示します。この節では、相関係数や共分散、有限差分機能の計算に関する関数についても説明します。

データの前処理
データセットの中の欠測値、異常値、間違ったデータの取り扱いを説明します。

回帰とカーブフィッティング
種々の回帰法を使って、観測された変数間の関係を記述する関数を求めます。

ケーススタディ：カーブフィッティング
ケーススタディを使って、MATLAB の基本的なデータ解析機能を調べます。この節は、Basic Fitting インタフェースに関する情報も提供します。

差分方程式とフィルタリング
差分方程式とフィルタの取り扱いに関する MATLAB 関数の説明

フーリエ解析と高速フーリエ変換 (FFT)
MATLAB の中でのフーリエ解析の説明

データ解析と統計関数
データ解析と統計関数は、MATLAB ツールボックスの中のデレクトリ datafun に配置されています。各関数の説明は、オンラインヘルプを参照してください。

関連したツールボックス

特定のデータ解析アプリケーションに対するよりアドバンスな機能を提供するツールボックスもいくつか用意されています。

ツールボックス	データ解析アプリケーション
Optimization	非線形カーブフィッティングと回帰
Signal Processing	信号処理、フィルタリング、周波数解析
Spline	カーブフィッティングと回帰
Statistics	アドバンスな統計解析、非線形カーブフィッティングと回帰
System Identification	パラメトリック /ARMA モデル化
Wavelet	ウエーブレット解析

列方向のデータセット

単変数統計データは、MATLAB 内で使うとき、個々のベクトルとして保存されます。ベクトルは、1 行 *n* 列または *n* 行 1 列のどちらかの型で保存されます。多変数データに対して、行列は非常に自然な表現ですが、基本的には、並べる方向として 2 つの可能性があります。MATLAB の使い易さのために、個々の変数は列の型で、観測の変化を行方向に設定しています。そのため、3 変数の 24 個のサンプルからなるデータ集合は、大きさ 24 行 3 列の行列になります。

交通量サンプルデータセット

サンプルデータは、24 時間に渡り、3 地点で観測した交通量です。

交通量サンプルデータセット

時刻	場所 1	場所 2	場所 3
01h00	11	11	9
02h00	7	13	11
03h00	14	17	20
04h00	11	13	9
05h00	43	51	69
06h00	38	46	76
07h00	61	132	186
08h00	75	135	180
09h00	38	88	115
10h00	28	36	55
11h00	12	12	14
12h00	18	27	30
13h00	18	19	29
14h00	17	15	18

交通量サンプルデータセット（続き）

時刻	場所 1	場所 2	場所 3
15h00	19	36	48
16h00	32	47	10
17h00	42	65	92
18h00	57	66	151
19h00	44	55	90
20h00	114	145	257
21h00	35	58	68
22h00	11	12	15
23h00	13	9	15
24h00	10	9	7

データのロードとプロット

実データを ASCII ファイルとして count.dat に保存します。

```
 11   11    9
  7   13   11
 14   17   20
 11   13    9
 43   51   69
 38   46   76
 61  132  186
 75  135  180
 38   88  115
 28   36   55
 12   12   14
 18   27   30
 18   19   29
 17   15   18
 19   36   48
 32   47   10
 42   65   92
```

```
    57    66   151
    44    55    90
   114   145   257
    35    58    68
    11    12    15
    13     9    15
    10     9     7
```

load コマンドを使って、データを読み込みます。

```
load count.dat
```

これは、ワークスペースの中に行列 count を作成します。

この例題では、3 変数に対して 24 の観測量があります。つぎの方法で、確認することができます。

```
[n,p] = size(count)
n =
    24
p =
     3
```

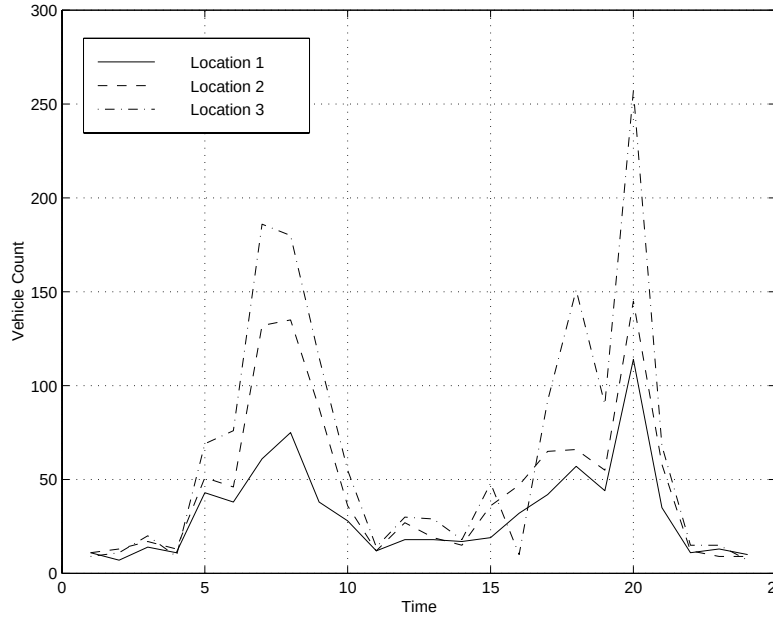
11 から n までの整数からなる時間ベクトル t を作成します。

```
t = 1:n;
```

時間に対する count 値をプロットし、コメントもプロットします。

```
set(0,'defaultaxeslinestyleorder',’-|--|-.’)
set(0,'defaultaxescolororder',[0 0 0])
plot(t,count), legend('Location 1','Location 2','Location 3',0)
xlabel('Time'), ylabel('Vehicle Count'), grid on
```

プロット図は、24 時間に渡る 3 地点の交通量を示しています。


Location 1
Location 2
Location 3
Vehicle Count
Time
0
50
100
150
200
250
300
0
5
10
15
20
25

基本的なデータ解析関数

この節は、つぎの事柄に関する関数を紹介します。

- 基本的な列方向のデータ解析
- 相関係数と共分散係数の計算
- 有限差分の計算

関数のまとめ

関数は、基本的に列方向にデータ解析を行ないます。これらの関数は、MATLAB の datafun ディレクトリに配置されています。

この節は、行データと列データを使用することのいくつかのヒントを記述しています。そして、いくつかの基本的な例題も示しています。

基本的な解析関数のまとめ

関数	詳細
cumprod	要素の累積積
cumsum	要素の累積和
cumtrapz	累積台形数値積分
diff	微分関数と微係数近似
max	要素の中の最大値の検出
mean	平均値
median	中央値
min	要素の中の最小値の検出
prod	要素の積
sort	大きい順に並び替え
sortrows	行を大きい順に並び替え
std	標準偏差

基本的な解析関数のまとめ（続き）

関数	詳細
sum	要素の和
trapz	台形数値積分

プロットしたデータに対して、最大値、最小値、平均値、中央値、レンジ、標準偏差の計算や、これらの統計量のプロット作成に関する情報について、MATLAB グラフィックスドキュメントの「データの統計量をグラフに付加」を参照してください。

行データと列データの取り扱い

これらの関数へのベクトル入力引数に関して、ベクトルは行方向または列方向のどちらでも構いません。しかし、配列引数に対して、関数は配列の中のデータの列方向に機能します。例えば、ある配列に max を適用するとすれば、各々の列に対して適用され、最大値を含む行ベクトルを出力します。

注意 M- ファイルを使って、より多くの機能を加えることができます。しかし、それを行なうのに、行ベクトルを取り扱う場合は注意が必要です。ユーザ自身の列方向を処理する M- ファイルを書くならば、他の M- ファイルもチェックしてください。例えば、mean.m や diff.m を参照してください。

基本的な例題

交通量の例題を続けます。つぎのステートメント

```
mx = max(count)
mu = mean(count)
sigma = std(count)
```

は、つぎの結果を出力します。

```
mx =
         114         145         257

mu =
      32.0000     46.5417     65.5833
```

```
sigma =
      25.3703       41.4057       68.0281
```

最小値と最大値になる部分のインデックスを求めるには、2 番目の出力引数も設定します。例えば、

```
[mx,indx] = min(count)

mx =
      7     9     7


indx =
      2    23    24
```

は、最初の観測位置 (列 1) では 02h00、2 番目では 23h00、3 番目では 24h00 に最小の交通量が記録されています。

n 要素すべてが 1 の行列との外積を使って、各々の列データから平均値を取り除きます。

```
[n,p] = size(count)
e = ones(n,1)
x = count - e*mu
```

データの並べ替えは、全データセットに渡り、ベクトル関数を計算するのに利用可能です。例えば、全データセットの中で最小値を求めるには、つぎのようにすると簡単です。

```
min(count(:))
```

結果は、つぎのようになります。

```
ans =
      7
```

count(:) により、24 行 3 列の行列を 72 行 1 列の列ベクトルに並べ替えます。

共分散係数と相関係数

MATLAB の統計計算機能は、相関係数と共分散の計算に対して、2 つの関数を用意しています。

共分散係数と相関係数関数のまとめ

関数	詳細
cov	ベクトルの共分散 - サンプル変数の広がり、または、バラツキ度 行列の共分散 - 変数間の線形関係の強さの尺度
corrcoef	相関係数 - 変数間の線形関係の強さの正規化された尺度

cov は、データベクトルに対して分散を計算します。count の最初の列のデータの分散は、つぎの式を計算します。

```
cov(count(:,1))

ans =
      643.6522
```

配列データに対して、cov は共分散行列を計算します。配列の列に対して、分散値は共分散行列の対角要素として配置されます。非対角要素は、オリジナルの配列の列間の共分散値を示しています。*m* 行 *n* 列の行列に対して、共分散行列は *n* 行 *n* 列の行列になります。例えば、count,cov(count) に対する共分散行列は、つぎのような配列になります。

$$\begin{bmatrix} \sigma_{11}^2 & \sigma_{12}^2 & \sigma_{13}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 & \sigma_{23}^2 \\ \sigma_{31}^2 & \sigma_{32}^2 & \sigma_{33}^2 \end{bmatrix}$$

$$\sigma_{ij}^2 = \sigma_{ji}^2$$

corrcoef は、各行が観測で、各列が変数であるデータ配列に対して、相関係数行列を計算します。*相関係数*は、2 つの変数間で線形な関係の強さを表す正規化された尺度です。相関のないデータは相関係数がゼロになります。全く等価なデータセットは相関係数が 1 になります。

m 行 *n* 列の行列に対して、相関係数行列は *n* 行 *n* 列の行列になります。相関係数行列の中の要素の位置は、上に記述されている共分散行列の中の要素の位置に対応します。

ここで扱っている交通量の例題に対して、

```
corrcoef(count)
```

結果は、つぎのようになります。

```
ans =
    1.0000    0.9331    0.9599
    0.9331    1.0000    0.9553
    0.9599    0.9553    1.0000
```

3 地点で観測された 3 つの交通量の間には、結果が 1 に近いので、強い線形の相関があることがわかります。

有限差分

MATLAB は、有限差分計算に 3 つの関数を用意しています。

関数	詳細
diff	ベクトルの隣の要素との差 ベクトルの数値偏微分
gradient	数値偏微分行列
del2	行列の離散ラプラシアン

関数 diff は、数値ベクトルの中の隣り合った要素間の差を計算します。すなわち、[X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)] です。あるベクトル A に対して、

```
A = [9 -2 3 0 1 5 4];
diff(A)

ans =
   -11     5    -3     1     4    -1
```

1 階微分を計算する以外に、diff はベクトルのある特性を決めるのにも役に立ちます。例えば、ベクトルが単調増加または単調減少か、またはベクトルが等間隔に分けられた要素をもっているかどうかを決めるのに diff を使うことができま

す。例えば、以下の表は、ベクトル x を与えて diff を実行するための方法を示しています。、

テスト	詳細
diff(x)==0	同じ値を繰り返しているかのチェック
all(diff(x)>0)	単調性であるかチェック
all(diff(diff(x))==0)	ベクトル要素すべてが等間隔であるかどうかのチェック

データの前処理

この節は、つぎのデータの取り扱いに関して記述します。

- 欠測値
- 異常値や間違いデータ

欠測値

特殊な値 NaN は MATLAB の中で、Not-a-Number を省略したものです。IEEE 浮動小数点代数変換は、0/0 のような定義されない表現の結果として NaN を出力します。

欠測データの正しい取り扱いは、難しい問題です。そして、種々の立場で、異なる取り扱いがなされます。データ解析目的に対して、欠測値や*利用できないデータ* を表現するのに NaN を使うことが便利になります。

MATLAB は、一様で厳格な方法で NaN を取り扱います。すなわち、ある計算の中で NaN を生じると、最後の結果まで、この値を持ち続けます。NaNs を含んでいる数学計算は、結果にも NaN を含みます。

例えば、行列の中心に NaN をもつ 3 行 3 列の魔方陣を含む行列を考えます。

```
a = magic(3); a(2,2) = NaN

a =
     8     1     6
     3   NaN     7
     4     9     2
```

行列の各列の和を計算します。

```
sum(a)

ans =
    15   NaN    15
```

NaN を含むある数学計算は、最終結果まで NaNs をもたらせます。

統計計算を行なう前に、データから NaN を取り去ってください。それを行なうのに種々の方法があります。

コード	詳細
i = find(~isnan(x)); x = x(i)	NaN でないベクトルの中の要素のインデックスを探し、NaN でない要素のみを残します。
x = x(find(~isnan(x)))	ベクトルから NaN を取り除く
x = x(~isnan(x));	ベクトルから NaN を取り除く (高速)
x(isnan(x)) = [];	ベクトルから NaN を取り除く
X(any(isnan(X)'),:) = [];	NaN を含む行列からある行を取り除く

注意　IEEE 代数変換により、論理比較 NaN == NaN は必ず 0 になるので、NaN を見つけるのに特殊関数 isnan を使うことができます。データから NaNs を取り除くのに、x(x==NaN) = [] を使うことはできません。

NaN を取り除くことが頻繁にあるならば、つぎのような M- ファイルを作成してください。

```
function X = excise(X)
X(any(isnan(X)'),:) = [];
```

そして、つぎのようにタイプしてください。

```
X = excise(X);
```

同じ結果になります。

異常値の除去

NaNsと同様にデータセットから異常値または信用できないデータ点を取り除くことができます。交通量データに対して、データの各列の平均と標準偏差を求めてみましょう。

```
mu = mean(count)
sigma = std(count)
```

```
mu =
       32.0000    46.5417    65.5833

sigma =
       25.3703    41.4057    68.0281
```

標準偏差の 3 倍よりも大きい値を異常値とし、そのようなデータを含んでいる行の番号をつぎのステートメントで得られます。

```
[n,p] = size(count)
outliers = abs(count - mu(ones(n, 1),:)) > 3*sigma(ones(n, 1),:);
nout = sum(outliers)
nout =
       1     0     0
```

最初の列に一つ異常値が含まれています。これに対応するすべての観測値を取り除きます。

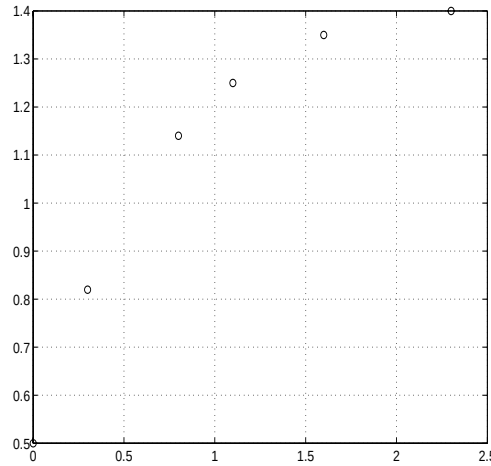
```
count(any(outliers'),:) = [];
```

回帰とカーブフィッティング

ユーザが取り扱ういくつかの変数間の関係を記述する関数を見つけることは、しばしば有効になります。関数の係数を同定することは、連立方程式の過多システムを定式化することから求まります。これらの係数は、MATLAB のバックスラッシュ演算子を使って求めることができます。

いくつかの時刻 t で測定された量があると考えます。

```
t = [0 .3 .8 1.1 1.6 2.3]';
y = [0.5 0.82 1.14 1.25 1.35 1.40]';
plot(t,y,'o'), grid on
```



つぎの節は、データをモデリングする 3 つの手法を示しています。

- 多項式による回帰
- パラメータが線形である方程式の回帰
- 多変数回帰

多項式回帰

プロット図をベースにすると、データは多項式

$$y = a_0 + a_1 t + a_2 t^2$$

でモデル化できる可能性があります。未知の係数 a_0, a_1, a_2 は、*最小二乗法*を使って計算されます。すなわち、モデルから割り出されたデータとの偏差の二乗和を最小にするように計算します。3 つの未知数に 6 つ方程式があります。

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} = \begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \\ 1 & t_4 & t_4^2 \\ 1 & t_5 & t_5^2 \\ 1 & t_6 & t_6^2 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

の型で 6 行 3 列の行列を表現します。

```
X = [ones(size(t))  t  t.^2]

X =
    1.0000         0         0
    1.0000    0.3000    0.0900
    1.0000    0.8000    0.6400
    1.0000    1.1000    1.2100
    1.0000    1.6000    2.5600
    1.0000    2.3000    5.2900
```

解はバックスラッシュ演算で求まります。

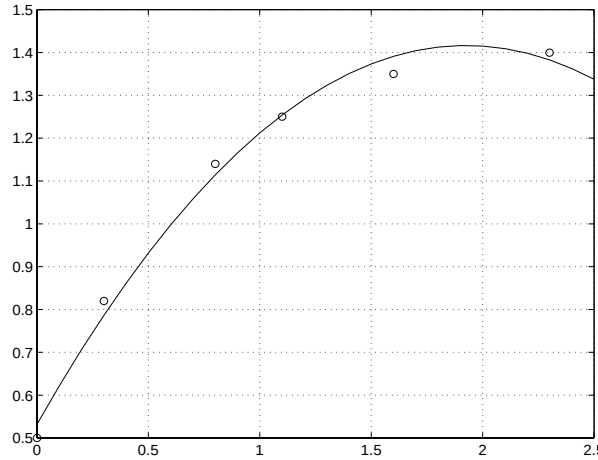
```
a = X\y

a =
    0.5318
    0.9191
  - 0.2387
```

データの 2 次多項式モデルは、つぎのようになります。

$$y = 0.5318 + 0.919(1)t - 0.2387t^2$$

さて、等間隔キザミでモデルを計算し、オリジナルプロット上に重ね書きします。

```
T = (0:0.1:2.5)';
Y = [ones(size(T))  T  T.^2]*a;
plot(T,Y,'-',t,y,'o'), grid on
```



明らかに、この近似は、データに完全にはフィットしていません。近似の度合いを高くするには、多項式の次数を高くするか、または、別の関数型を使うかのどちらかを行なってください。

パラメータが線形である方程式の回帰

多項式の代わりに、パラメータが線形である方程式を使うことを考えましょう。この場合、指数関数を考えます。

$$y = a_0 + a_1 e^{-t} + a_2 t e^{-t}$$

未知係数a_0, a_1, a_2は、*最小二乗近似*を使って計算されます。回帰行列 X を作ることで連立方程式を作成し、そして解くためにバックスラッシュ演算子を使って係数を求めます。

```
X = [ones(size(t))  exp(- t)  t.*exp(- t)];
a = X\y

a =
    1.3974
```

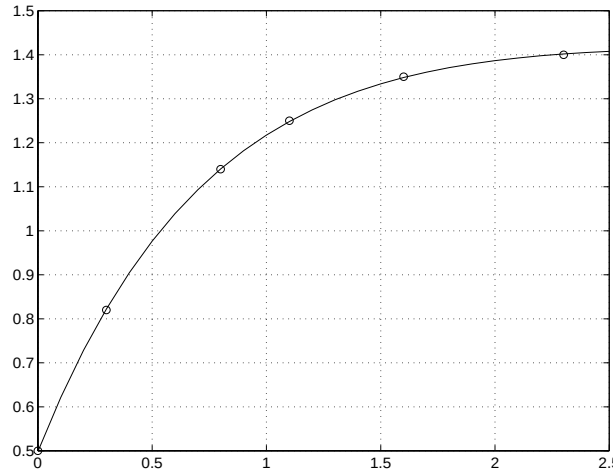
```
   - 0.8988
     0.4097
```

データの近似モデルは、つぎのようになります。

$$y = 1.3974 - 0.8988\ e^{-t} + 0.4097\ te^{-t}$$

さて、等間隔のキザミでモデルを計算し、オリジナルのデータと重ね書きしましょう。

```
T = (0:0.1:2.5)';
Y = [ones(size(T))  exp(- T)  T.exp(- T)]*a;
plot(T,Y,'-',t,y,'o'), grid on
```



この図は、2 次多項式近似よりも、良く近似していることを示しています。

多変数回帰

y が複数の独立変数からなる関数ならば、変数間の関係を表す行列方程式は、追加データを集めて展開します。

パラメータ x_1 と x_2 のいくつかの値に対して、値 y を観測していると仮定しましょう。観測値を、つぎのように表すとします。

```
x1 = [.2 .5 .6 .8 1.0 1.1]';
x2 = [.1 .3 .4 .9 1.1 1.4]';
```

```
y  = [.17  .26  .28  .23  .27  .24]';
```

データの多変数モデルを、つぎのように考えます。

$$y = a_0 + a_1 x_1 + a_2 x_2$$

未知係数a_0, a_1, a_2に関して、多変数の解は、*最小二乗近似*を使って解きます。回帰行列 X を作成して連立方程式を作成し、バックスラッシュ演算子を使って係数を解きます。

```
X = [ones(size(x1))  x1  x2];
a = X\y

a =
    0.1018
    0.4844
   -0.2847
```

データの最小二乗モデルは、つぎのようになります。

$$y = 0.1018 + 0.4844\ x_1 - 0.2847\ x_2$$

モデルを検証するために、モデルから求めた点と観測データ点の偏差の絶対値の最大値を求めます。

```
Y = X*a;
MaxErr = max(abs(Y - y))

MaxErr =
     0.0038
```

この値は、データへの適合がうまくいっているモデルであることを確信できるほど十分に小さいものです。

ケーススタディ：カーブフィッティング

この節は、ケーススタディの型で MATLAB の基本データ解析機能のいくつかの概要を示しましょう。この例題は、人口データを取り扱っていて、MATLAB の機能を使ってデータのカーブフィットを行なうものです。

- 多項式近似
- 残差の解析
- 指数近似
- エラーの範囲

この節は、カーブフィッティングを行う Basic Fitting インタフェースの使い方を説明します。

データのロード

ファイル census.mat は、1790 年から 1990 年までの US での人口のデータを含んでいます。MATLAB に読み込みます。

```
load census
```

ユーザのワークスペースに 2 つの新しい変数 cdate, pop が含まれています。

- cdate は、1790 年から 1990 年までの 10 年毎の時代を示す列ベクトルです。
- pop は、cdate の中の年に対応する US での人口を含む列ベクトルです。

P 多項式近似

人口データの近似の最初の試みは、簡単な多項式近似です。この処理には 2 つの MATLAB 関数が用意されています。

カーブフィッティング関数のまとめ

関数	詳細
polyfit	多項式近似
polyval	多項式近似値の計算

MATLAB の関数 polyfit は、与えられたデータセットに対して設定した次数の " 最良の近似と言う意味 (最小二乗の意味)" の多項式です。次数 4 の多項式は、

```
p = polyfit(cdate,pop,4)
```

```
Warning: Polynomial is badly conditioned. Remove repeated data points or try
centering and scaling as described in HELP POLYFIT.

p =
   1.0e+05 *

     0.0000   -0.0000    0.0000   -0.0126 6.0020
```

関数 polyfit は、非常に大きな値をもつ行列 (この計算の中で、Vandermonde 行列を作成します。詳細は、polyfit ファイルを参照してください) に対して基底として cdate 値を使っているので、ワーニングが表示されます。cdate 値の広がりは、スケーリング問題になります。これを取り扱う 1 つの方法は、cdate データを正規化する方法です。

前処理：データの正規化

正規化 は、連続的な数値計算の精度を改良するデータセットの中の数値をスケーリングする過程です。cdate を正規化する方法は、平均をゼロ、標準偏差を 1 になるようにスケーリングすることです。

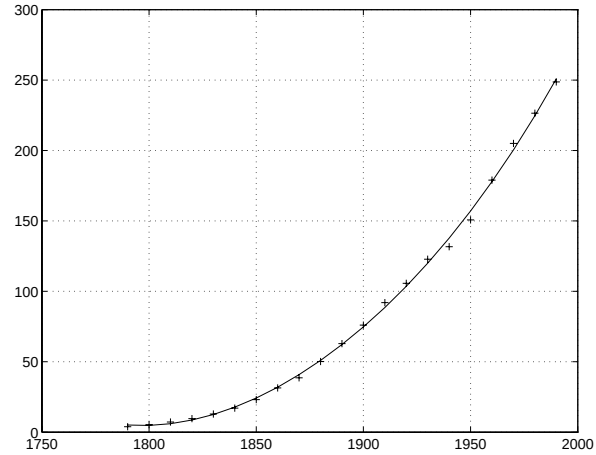
```
sdate = (cdate - mean(cdate))./std(cdate)
```

正規化されたデータを使って、4 次多項式モデルで計算してみましょう。

```
p = polyfit(sdate,pop,4)

p =
    0.7047    0.9210   23.4706   73.8598   62.2285
```

正規化したデータ (年代) で近似多項式値を計算し、観測データ点に対してプロットします。

```
pop4 = polyval(p,sdate);
plot(cdate,pop4,'-',cdate,pop,'+'), grid on
```



データを正規化する他の方法は、解に関するある知識と単位を使うことです。例えば、このデータセットを使って、1790 年のデータをゼロに設置することでも、満足のいく結果を出力します。

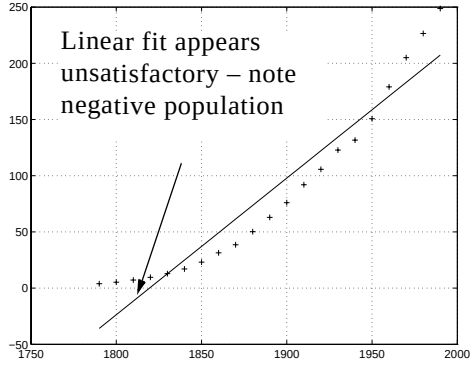
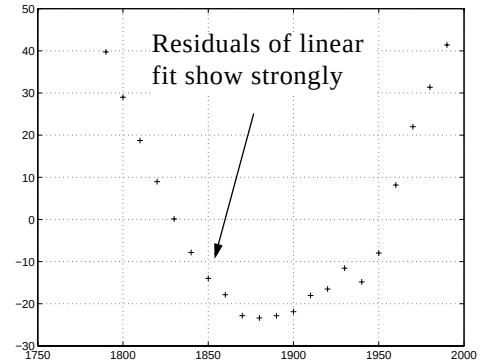
残差の解析

適合の " 度合い " の尺度の一つに*残差* があります。これは、観測データと予測データの差で表すものです。正規化した cdate 値を使って、種々の適合に対する残差の比較を行ないましょう。近似プロットと残差を使うと、データセットを使った簡単な近似よりも、良い結果が得られることは明らかです。

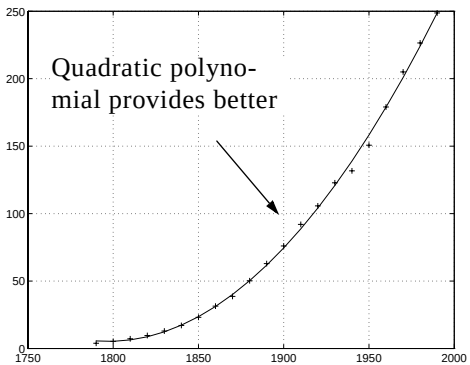
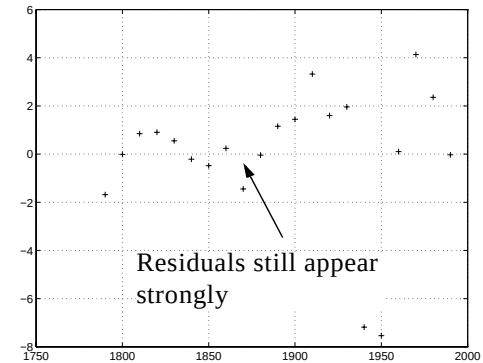
近似と残差のプロットの比較

近似	残差

p1 = polyfit(sdate,pop,1);
pop1 = polyval(p1,sdate);
plot(cdate,pop1,'–',cdate,pop,'+')
	res1 = pop – pop1;
	figure, plot(cdate,res1,'+')

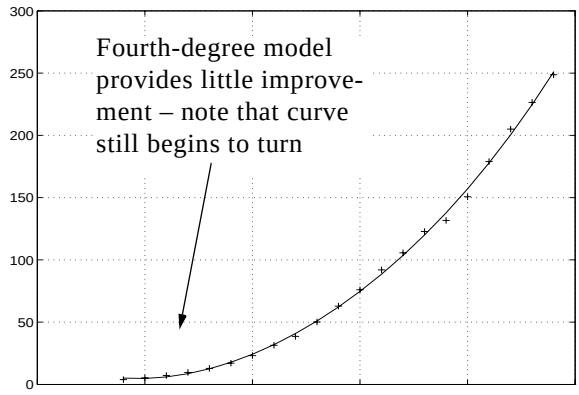
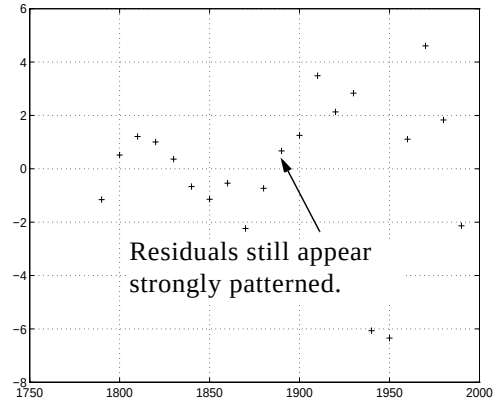
p = polyfit(sdate,pop,2);
pop2 = polyval(p,sdate);
plot(cdate,pop2,'-',cdate,pop,'+')
	res2 = pop – pop2;
	figure, plot(cdate,res2,'+')

近似と残差のプロットの比較（続き）

近似	残差

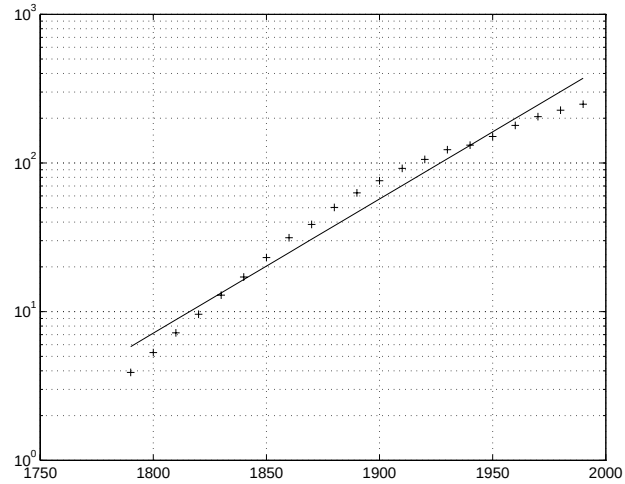
p = polyfit(sdate,pop,4);
pop4 = polyval(p,sdate);
plot(cdate,pop4,'–',cdate,pop,'+')
	res4 = pop – pop4;
	figure, plot(cdate,res4,'+')

指数近似

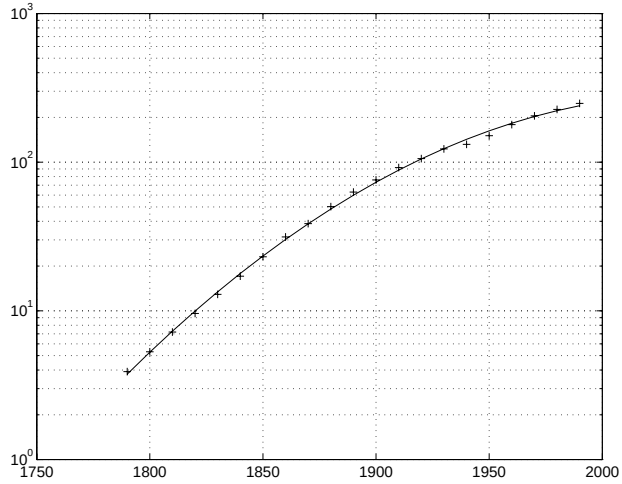
前のページで人口データのプロットを調べると、人口データ曲線は、見た目で、わずかですが指数的に見えます。この見え方を利用して、正規化された年に対する人口値の対数データでフィットして見ましょう。

```
logp1 = polyfit(sdate,log10(pop),1);
logpred1 = 10.^polyval(logp1,sdate);
semilogy(cdate,logpred1,'-',cdate,pop,'+');
grid on
```



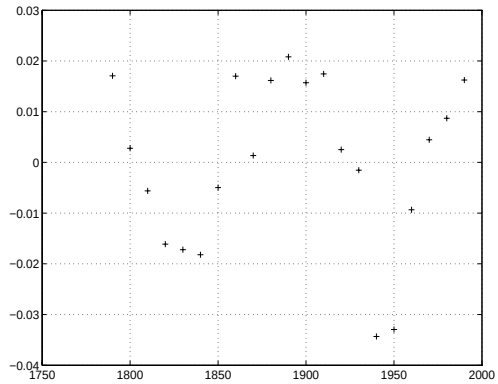
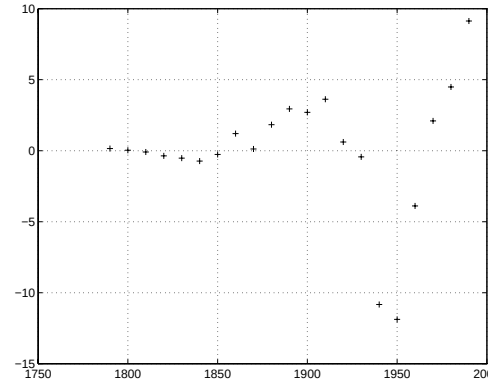
つぎに、2 次モデルを使って対数解析を行なって見ましょう。

```
logp2 = polyfit(sdate,log10(pop),2);
logpred2 = 10.^polyval(logp2,sdate);
semilogy(cdate,logpred2,'-',cdate,pop,'+'); grid on
```



これは、よりフィットしたモデルになります。プロットの中の上限部は先細りになっているように見え、一方、下限部の多項式フィットは、無限まで連続で、凸型になっています。

さて、2 次の対数モデルに対する残差を見てみましょう。

人口の対数スケールの残差	人口スケールの残差
logres2 = log10(pop) – polyval(logp2,sdate); plot(cdate,logres2,'+')	r = pop – 10.^(polyval(logp2,sdate)); plot(cdate,r,'+')
	

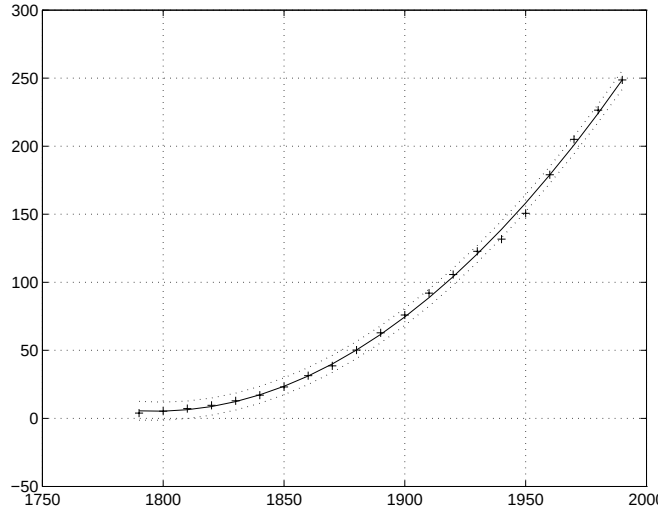
残差は、単純な多項式適合よりもランダムです。期待したように、残差は人口が増えるに従って大きくなります。しかし、全体的に、対数モデルは、人口データにより精度良くフィットします。

誤差の範囲

誤差の範囲は、ユーザのデータが適合により、うまくモデル化されているかを決めることが有効です。オプションの 2 番目の出力パラメータは polyfit から得られ、エラーの範囲を得るために polyval に一つの入力パラメータとして使われます。

例えば、polyfit と polyval を使って、2 次多項式モデルに対してエラーの範囲を計算します。年データは正規化されています。ここでは、95% の信頼区間に対応する ±2Δ の間隔を使います。

```
[p2,S2] = polyfit(sdate,pop,2);
[pop2,del2] = polyval(p2,sdate,S2);
plot(cdate,pop,'+',cdate,pop2,'g-',cdate,pop2+2*del2,'r:',...
   cdate,pop2-2*del2,'r:'), grid on
```



Basic Fitting インタフェース

MATLABは、基本的なカーブフィッティングインタフェースを使って、カーブフィッティング作業をサポートしています。このインタフェースを使って、使用し易い環境を使い、基本的なカーブフィッティングの仕事を高速で行うことができます。特に、つぎの事柄を実行することができます。

- データをキュービックスプライン、または、10 次までの多項式に近似
- 与えられたセータセットに対して、複数の適合結果を同時に表示
- 適合の残差
- 数値的な適合結果の表示
- 近似に対する (内挿や外挿) 計算
- プロットに、数値的な適合結果、残差のノルム、データの統計量を付加して表示
- 近似結果や計算結果を MATLAB ワークスペースにセーブ

ユーザの指定したカーブフィッティングアプリケーションに依存して、コマンドライン機能、Basic Fitting インタフェースを共に使用することができます。

Basic Fitting インタフェースでは、2 次元データのみしか使用できません。しかし、サブプロットを使って、複数のデータセットをプロットしたい場合でも、その内の少なくとも、一つのデータセットが 2 次元の場合に、インタフェースを使用することができます。

注意 HP, IBM, SGI プラットフォームに対して、Basic Fitting インタフェースは、リリース 12 ではサポートしていません。

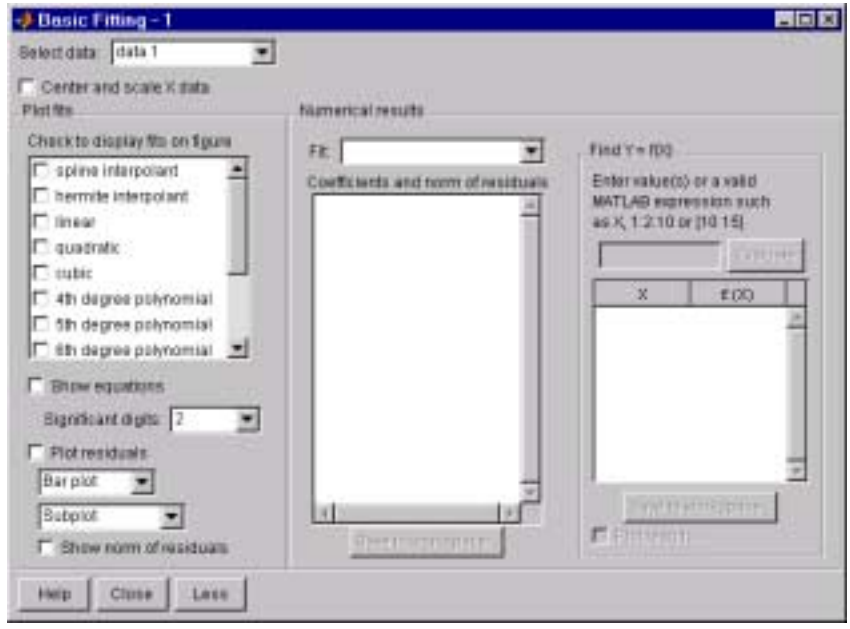
Basic Fitting インタフェースの概要

Basic Fitting インタフェース全体をつぎに示します。つぎの状態を再生するには、

1 あるデータをプロットします。

2 **ツール (Tools)** メニューから、**基本的な近似 (Basic Fitting)** を選択します。

3 **More** ボタンを 2 度クリックします。



データの選択 - このメニューは、Basic Fitting インタフェースに関連したフィギュアウインドウの中に表示されているすべてのデータセットの名前を表示します。

このメニューを使って、カレントデータセットを選択します。カレントデータセットは、近似されるデータセットとして定義します。ユーザは、一度に一つのデータセットのみを近似できます。しかし、カレントのデータセットに対して、複数の近似を行うこともできます。Plot Editor を使って、データセットの名前を変更することができます。

データの中心とスケーリング - チェックしていると、データは平均をゼロにして中心とし、標準偏差が 1 になるようにスケーリングします。その後に行う計算の精度の向上のために、データの中心を移動したり、スケーリングする必要がある場合があります。多項式近似に記述しているように、近似作結果が正しくない場合、MATLAB ワークスペースにワーニングを出力します。

近似プロット - このパネルは、カレントのデータセットに一つ、または、複数の近似を視覚的に表示するものです。

- **チェックすると、フィギュア上に近似を表示** - カレントのデータセットに対して表示したい近似を選択します。選択する近似のタイプには、interpolants と polynomials の 2 つがあります。スプライン内挿は、関数 spline を使い、エルミート内挿は、関数 pchip を使います。これらの 2 つの関数の比較については、pchip のオンラインヘルプを参照してください。多項式近似は、関数 polyfit を使います。与えられたデータに対して希望するあらゆる数の近似を選択することができます。

  ユーザのデータセットが N 点を含んでいるならば、最高で N 個までの係数をもつ多項式を使う必要があります。N 個以上の係数をもつ多項式を使って、近似すると、インタフェースは、計算の間に必要な数の係数を 0 に自動的に設定し、システムが算出できないことを避けます。
- **方程式の表示** - チェックしていると、近似方程式が、プロット上に表示されます。
  - **有効桁数** - 表示する近似係数に関連した有効桁数を選択します。
- **残差のプロット** - チェックしていると、近似の残差が表示されます。近似残差は、個々の横軸のデータ点に対して、近似結果と縦軸のデータ点との差として定義します。データを同じフィギュアウインドウを使って、バープロット、散布プロット、ラインプロットとして、残差を表示できます。また、別々なフィギュアウインドウを使っても表示できます。サブプロットを利用して、マルチデータセットをプロットする場合、残差は、別のフィギュアウインドウにのみプロットできます。
  - **残差のノルムの表示** - チェックしていると、残差のノルムが表示されます。残差のノルムは、適合度の尺度になります。ここで、より小さい値は、大きい値に比べて良い適合であることを示します。これらの値は、関数 norm や norm(V,2) を使って計算できます。ここで、V は、残差ベクトルです。

数値的な結果 - このパネルは、適合具合をプロットしないで、カレントデータセットへのある一つの適合の結果を数値的に表現します。

- **近似 (Fit)** - このメニューを使って、カレントデータに近似する方程式を選択できます。近似結果は、メニューの下のリストボックスに表示されます。このメニューでの方程式の選択は、**Plot fits** パネルの状態に影響を与えません。そのため、データプロット内に近似を表示したい場合は、**Plot fits** の対応するチェックボックスを選択する必要があります。
- **係数と残差ノルム** - **近似 (Fit)** に選択した方程式に対する結果を表示します。**Numerical Results** パネルを最初にオープンしたとき、最後の近似結果**近似をプロット (Plot fits)** で選択した最後の近似結果が表示されます。

- **ワークスペースへの保存** - 近似結果をワークスペース変数に保存するためのダイアログボックスを起動します。
- **Y = f(X) の検出** - カレント近似を内挿、または、外挿
  - **値の入力** - カレントデータセットに対するカレント近似を計算する MATLAB で定義した方程式を入力します。方程式は、**計算 (Evaluate)** ボタンを押すと計算され、結果は、関連したテーブルの中に表示されます。カレントの近似は、**近似 (Fit)** メニューに表示されます
  - **ワークスペースへの保存** - 計算結果をワークスペース変数にセーブするものを設定するダイアログボックスを起動します。
  - **結果のプロット** - チェックすると、計算結果が、プロット上に表示されます。

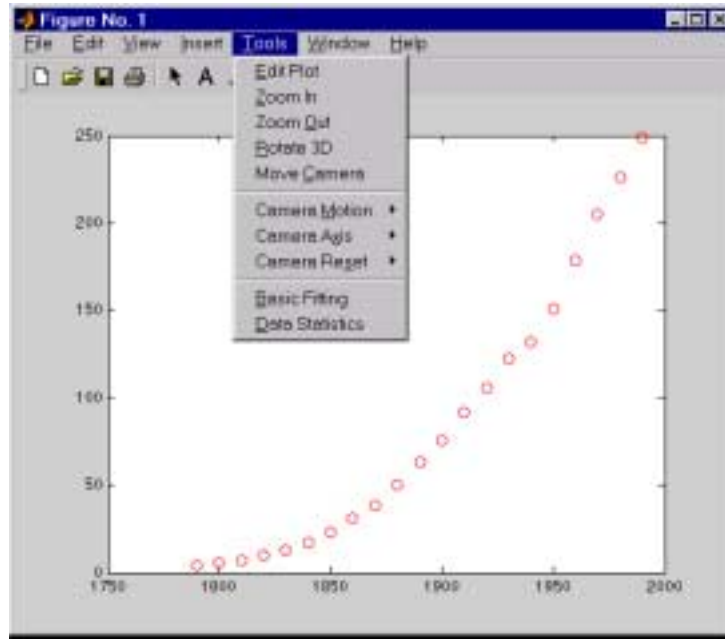
例題 : Basic Fitting インタフェースの利用

つぎの例題は、人口に関するデータを近似することで、Basic Fitting インタフェースの特徴をいくつか示すものです。ユーザは、種々の方程式を使って、この例題を繰り返し、結果を比較してみてください。インタフェースを起動するには、

1 データをプロットします。

```
plot(cdate,pop,'ro')
```

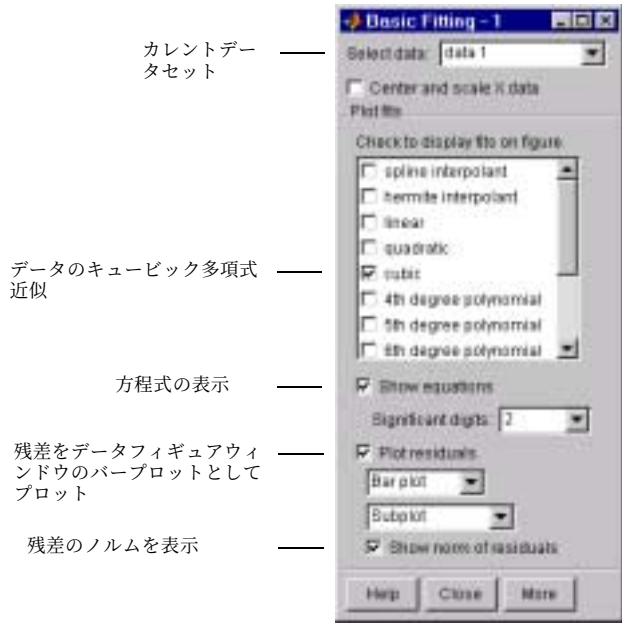
2 フィギュアの中の**ツール (Tools)** を使って、**Basic Fitting** を選択します。



Basic Fitting インタフェースは、つぎの機能をもっています。

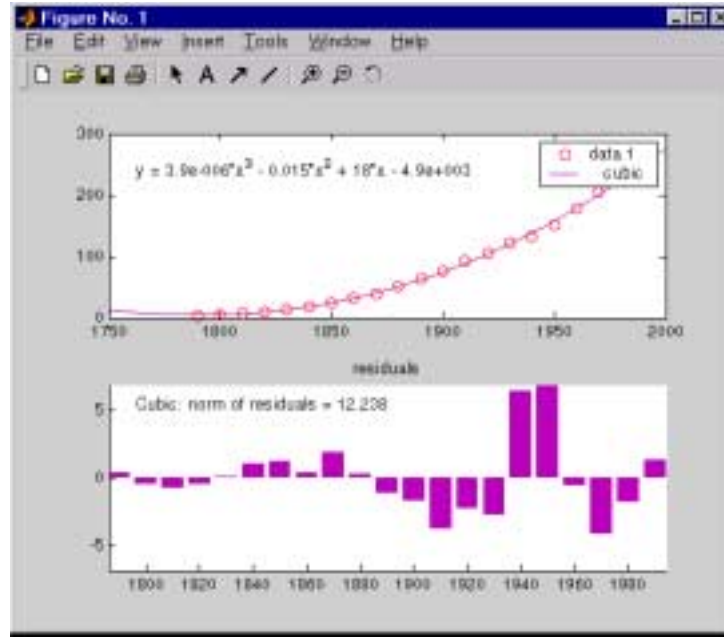
- データをキュービック多項式で近似
- フィギュアウインドウ内に方程式を表示
- 近似の残差をバープロットとしてプロットし、データフィギュアウィンドウのサブプロットとして残差を表示
- 残差のノルムを表示

このようなコンフィギュレーションを図に示します。



近似プロット (Plot fits) パネルは、カレントデータセットに複数の近似を視覚的に示すことができるものです。比較のために、適切なチェックボックスを選択して、人口に付加的な方程式を近似してみてください。方程式が数値的に正しくない結果を作成すると、MATLAB コマンドラインにワーニングが出力されます。この場合、**X データの中心とスケーリングの移動 (Center and scale X data)** チェックボックスを選択して。数値精度を改良してください。

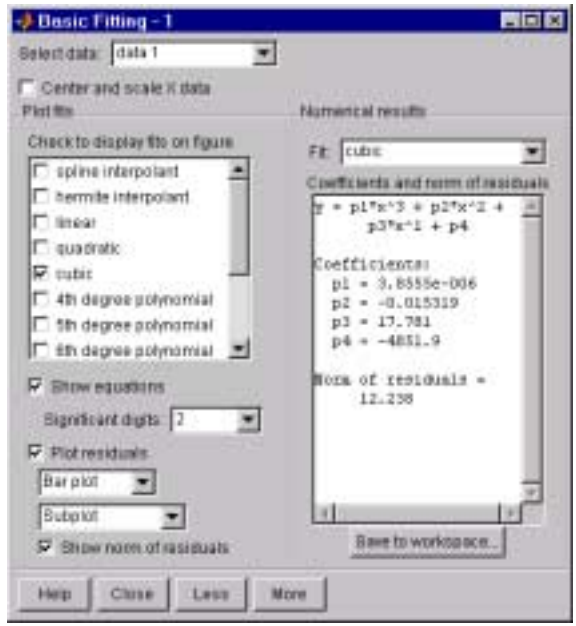
近似結果と残差をつぎに示します。



プロットの凡例は、データセットの名前と方程式を示しています。凡例は、データセットを加えたり、削除したり、近似すると、自動的に更新します。また、近似は、ライン様式やカラーのデフォルトセットを使って表示されます。Plot Editor を使って、デフォルトプロット設定の内容を変更することができます。しかし、ユーザが変更を加えても、その後に続けて他の近似を行なうと、適用されず、元にもどされます。変更をするには、ユーザのデータ近似が終了するまで待ってください。

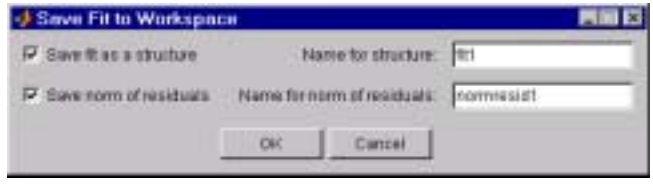
注意　凡例の中のデータセットの名前を変更する場合、名前は、**データの選択 (Select data)** メニューの中で自動的に変更されます。

More を選択して、近似係数と残差のノルムを表示することもできます。



近似 (Fit) メニューを使って、近似をプロットしないで、カレントのデータセットに対する数値的な近似を視覚化します。比較するため、希望する方程式を選択して、他の近似に対する数値的な結果を表示できます。データプロット内に近似を表示したい場合、**近似のプロット (Plot fits)** の中で、関連するチェックボックスを選択しなければなりません。

ワークスペースへの保存 (Save to workspace) ボタンを選択して、近似結果を MATLAB ワークスペースに保存することができます。



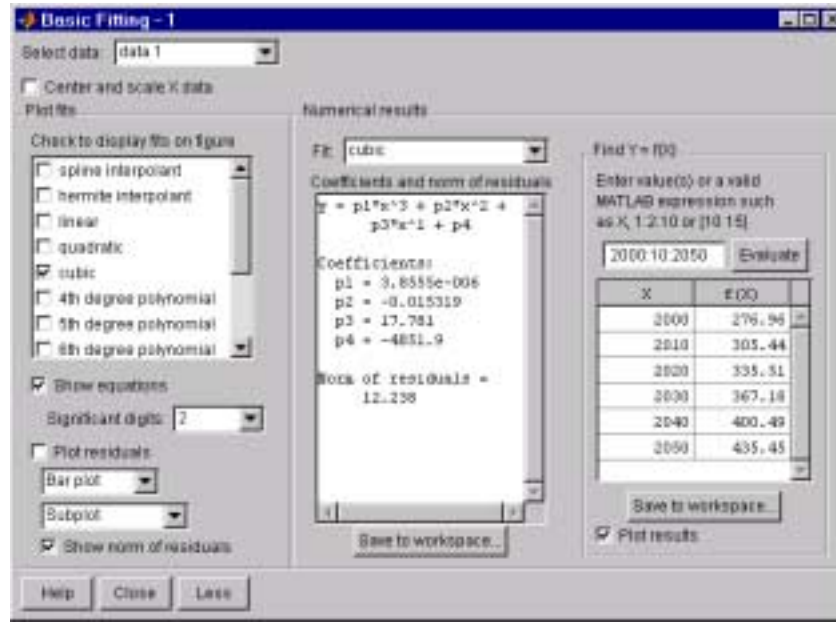
近似構造体をつぎに示します。

```
fit1
```

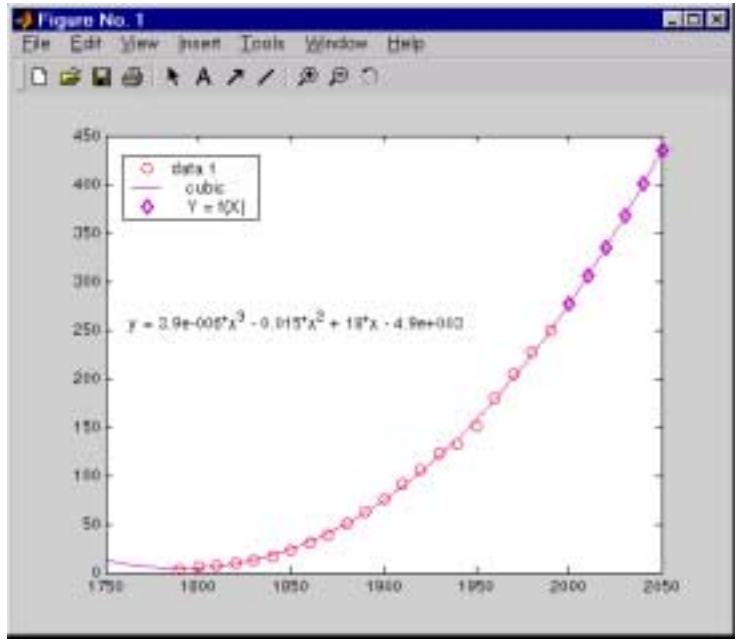
```
fit1 =
     type: 'polynomial degree 3'
    coeff: [3.8555e-006 -0.0153 17.7815 -4.8519e+003]
```

この構造体を使って、連続的に表示、または、解析をします。たとえば、保存した係数と関数 polyval を使って、コマンドラインで 3 次の多項式を推定することができます。

More ボタンを選択することで、インタフェースを使って、カレントの近似を計算することができます。カレントの近似が、近似 (Fit) メニューの中に表示されます。この例題では、2000 年から 2050 年までの人口を 10 年毎に計算し、プロットしています。



実行されたデータを以下に示します。



ワークスペースへ保存 (Save to workspace) ボタンを選択して、MATLAB ワークスペースに結果をセーブすることができます。

差分方程式とフィルタリング

MATLAB は、差分方程式とフィルタ操作に対応する関数を用意しています。これらの関数は、基本的には、ベクトルに作用します。

ベクトルは、信号処理やデータ解析に対して、サンプルホールド信号または連続信号を扱うのに使います。多入力システムに対して、行列の行は、行列の列として表現される入力をもつサンプル点に対応します。

関数

```
y = filter(b, a, x)
```

は、ベクトル a と b により記述されたフィルタでベクトル x のデータを処理し、フィルタ出力 y を作成します。

filterコマンドは､差分方程式を効果的に実行したものと考えることができます。フィルタの構造は、つぎのような差分方程式で記述される一般的に有限の遅れをもつものとして表現されます。ここで、*n* は、カレントサンプルのインデックスで、*na* はベクトル a により記述される多項式の次数で、*nb* はベクトル b により記述される多項式の次数です｡出力 *y*(*n*) は､カレントとその前の *x*(*n*) *x*(*n*-1)... と前の出力 *y*(*n*-1) *y*(*n*-2) ... の線形結合です。

$$a(1)y(n) = b(1)x(n) + b(2)x(n-1) + \ldots + b(nb)x(n-nb+1)$$

$$- a(2)y(n-1) - \ldots - a(na)y(n-na+1)$$

例えば、4 時間の窓を考えて、平均的な交通量を見るために移動平均フィルタを交通量データ count に適用してスムージングします。このプロセスは、つぎの差分方程式で表されます。

$$y(n) = \frac{1}{4}x(n) + \frac{1}{4}x(n-1) + \frac{1}{4}x(n-2) + \frac{1}{4}x(n-3)$$

対応するベクトルは、つぎのようになります。

```
a = 1;
b = [1/4 1/4 1/4 1/4];
```

つぎのコマンドを実行して、ワークスペースに行列 count を作成します。

```
load count.dat
```

この例題では、交通量の最初の列を取り出して、ベクトル x とします。

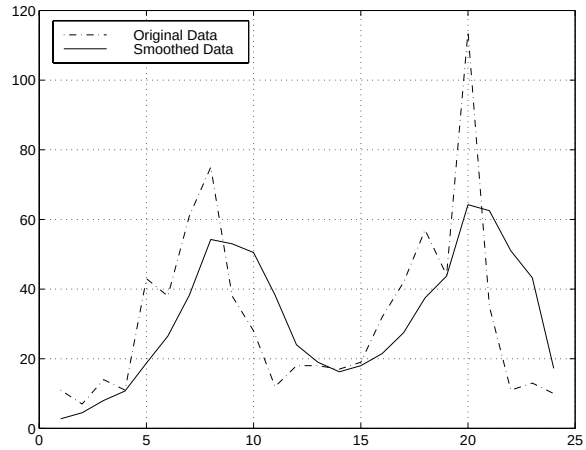
```
x = count(:,1);
```

データの 4 時間の移動平均を

```
y = filter(b,a,x);
```

で効率良く計算します。オリジナルデータとスムージングしたデータとを重ね書きして比べます。

```
t = 1:length(x);
plot(t,x,'-.',t,y,'-'), grid on
legend('Original Data','Smoothed Data',2)
```



実線で表示されるフィルタリングされたデータは、点線で表される観測された交通量の 4 時間移動平均です。

実際的なフィルタリングの適用に対して、Signal Processing Toolbox は、フィルタを設計したり、解析したりする多くの関数を用意しています。

フーリエ解析と高速フーリエ変換 (FFT)

フーリエ解析は、ある信号を異なる周波数の正弦波に分解するときに、非常に有効なものです。サンプリングされたベクトルデータに対して、フーリエ解析は、離散フーリエ変換 (DFT) を使って行ないます。

高速フーリエ変換 (FFT) は、連続なものの DFT を計算する効率的なアルゴリズムです。すなわち、分離変換ではありません。そして、信号処理、イメージ処理のような分野で特に有効なものです。この分野は、フィルタリング、コンボリューション、パワースペクトル推定の周波数解析に渡るものです。

この節では、つぎのことを説明します。

- フーリエ変換関数のまとめ
- 太陽の黒点の活動に関する例題を使って、フーリエ変換解析の説明
- 変換データのゲインと位相の計算
- 変換の長さに依存した実行時間の説明

関数のまとめ

MATLAB は、フーリエ変換を計算したり、取り扱う関数を用意しています。

FFT 関数のまとめ

関数	詳細
fft	離散フーリエ変換
fft2	2 次元離散フーリエ変換
fftn	N- 次元離散フーリエ変換
ifft	逆離散フーリエ変換
ifft2	2 次元逆離散フーリエ変換
ifftn	N- 次元逆離散フーリエ変換
abs	大きさ
angle	位相角
unwrap	ラジアン単位で表した連続性をもたせた位相角

FFT 関数のまとめ（続き）

関数	詳細
fftshift	DC 成分をスペクトルの中心に移動
cplxpair	複素共役の組に並べ替え
nextpow2	その値を含む最小の 2 のべき乗値

はじめに

長さ *N* の入力列 *x* に対して、DFT は長さ N のベクトル *X* になります。fft と ifft の関係は、つぎのようになります。

$$X(k) = \sum_{n=1}^{N} x(n) e^{-j2\pi(k-1)\left(\frac{n-1}{N}\right)} \qquad 1 \le k \le N$$

$$x(n) = \frac{1}{N} \sum_{k=1}^{N} X(k) e^{j2\pi(k-1)\left(\frac{n-1}{N}\right)} \qquad 1 \le n \le N$$

注意 MATLAB ベクトルの最初の要素は、インデックスが 1 としているので、上の方程式の和は、1 から *N* までのものに対応し、通常、記述するフーリエの 0 から *N*-1 までの和を使うものに対応します。

---

x(n) が実数の場合、上の方程式は、実数係数と sin と cos 関数の和として表わすことができます。

$$x(n) = \frac{1}{N} \sum_{k=1}^{N} a(k)\cos\left(\frac{2\pi(k-1)(n-1)}{N}\right) + b(k)\sin\left(\frac{2\pi(k-1)(n-1)}{N}\right)$$

ここで、つぎの関係があります。

$$a(k) = \text{real}(X(k)),\ b(k) = -\text{imag}(X(k)), 1 \le n \le N$$

列ベクトル x の FFT は、

```
x = [4 3 7 -9 1 0 0 0]' ;
```

つぎのステートメントを使います。

```
y = fft(x)
```

結果は、つぎのようになります。

```
y =
  6.0000
 11.4853 - 2.7574i
 -2.0000 -12.0000i
 -5.4853 +11.2426i
 18.0000
 -5.4853 -11.2426i
 -2.0000 +12.0000i
 11.4853 + 2.7574i
```

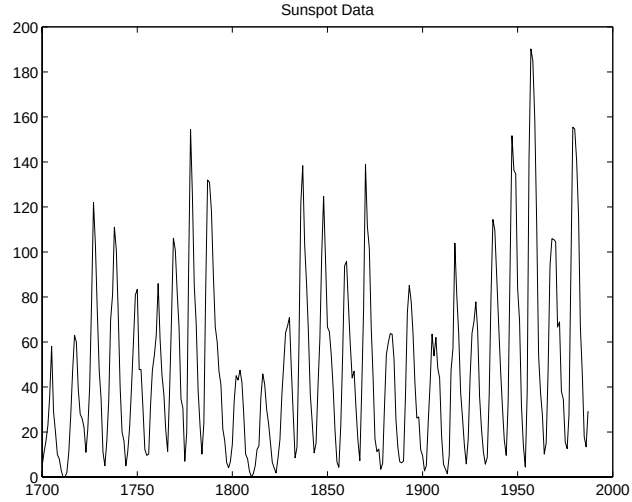
データ列 x は実数で、y は複素数であることに注意してください。変換されたデータの最初の成分は、定数として機能し、5 番目の要素は Nyquist 周波数に対応します。y の最後 3 つの値は、負の周波数に対応し、実数列 x に対して、y の前半の 3 つの要素の複素共役になります。

この 300 年間の太陽の黒点の活動の変化を解析しましょう。太陽の黒点の活動は、約 11 年の周期をもっていることが知られています。確認して見ましょう。

天文学者は、ほぼ 300 年間の Wolfer 数と呼ばれる量の一覧をもっています。この中には、太陽の黒点の数と大きさを含んでいます。

太陽の黒点データをロードし、プロットします。

```
load sunspot.dat
year = sunspot(:,1);
wolfer = sunspot(:,2);
plot(year,wolfer)
title('Sunspot Data')
```

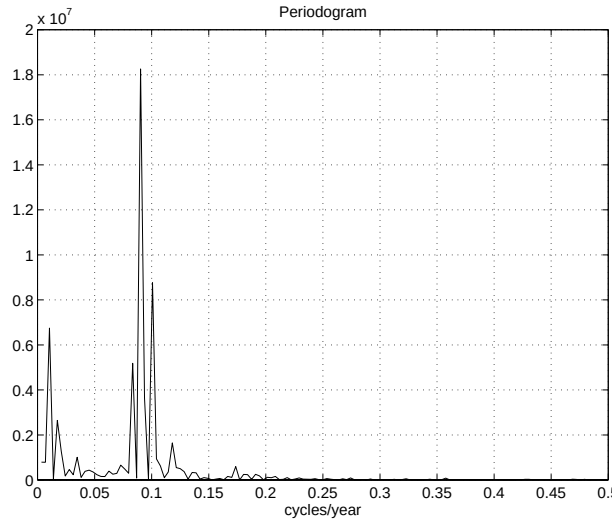


太陽の黒点データの FFT を計算します。

```
Y = fft(wolfer);
```

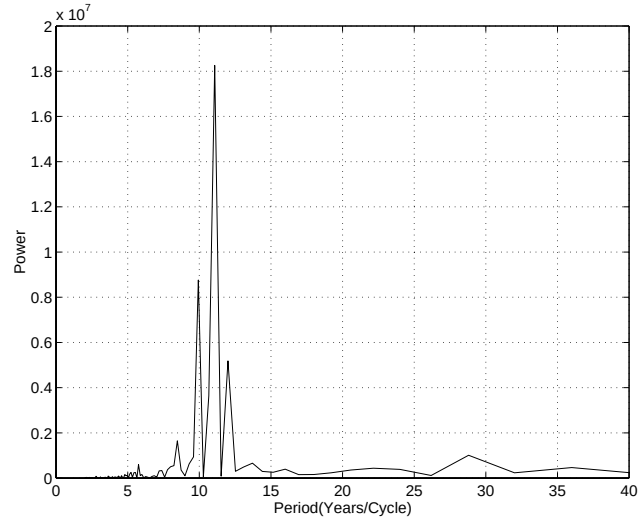
この変換の結果は、複素数ベクトル Y になります。Y の二乗の大きさは、パワーとよばれ、周波数に対してパワーのプロットは、" ピリオドグラム " と言われます。すべてのデータを加算したYの最初の要素を取り除き、結果をプロットします。

```
N = length(Y);
Y(1) = [];
power = abs(Y(1:N/2)).^2;
nyquist = 1/2;
freq = (1:N/2)/(N/2)*nyquist;
plot(freq,power), grid on
xlabel('cycles/year')
title('Periodogram')
```



年数に関する周期性のスケーリングは、幾分不便なものです。何が 1 サイクルなのかを推定するため 1 サイクルに対する年数をプロットします。便宜的に、0 から 40years/cycle の周期に対してパワーをプロットします (周期は period = 1./freq) です)。

```
period = 1./freq;
plot(period,power), axis([0 40 0 2e7]), grid on
ylabel('Power')
xlabel('Period(Years/Cycle)')
```



周期性をより詳しく決めるために、

```
[mp index] = max(power);
period(index)

ans =
       11.0769
```

変換されたデータのゲインと位相

変換されたデータ列に関して重要な情報は、その大きさと位相です。MATLAB 関数 abs と angle は、この情報を計算します。

これを試すために、時間ベクトル t を作成し、このベクトルを使って、異なる周波数をもつ 2 つの正弦波から構成されるデータ列 x を作成します。

```
t = 0:1/100:10-1/100;
x = sin(2*pi*15*t) + sin(2*pi*40*t);
```

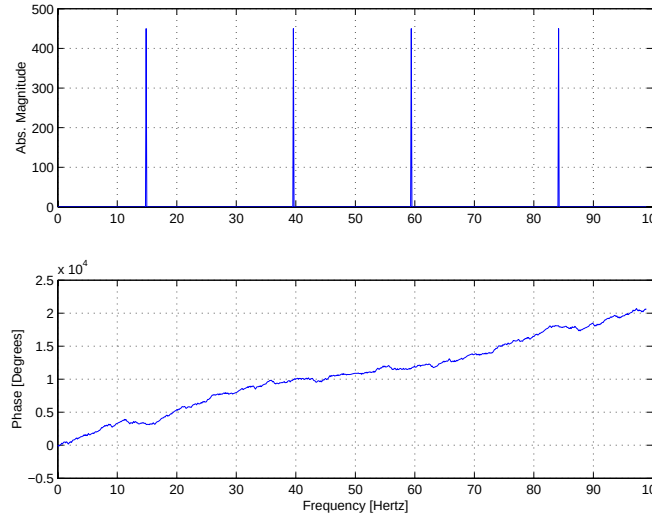
関数 fft を使ってデータ列の DFT を計算します。つぎのステートメントで、変換されたデータ列の大きさと位相を計算します。データの大きさを得るには abs で、位相情報は angle を使います。そして、pi より大きくてジャンプする位相に 2*pi 補正を適用するには unwrap を使います。

```
y = fft(x);
m = abs(y);
p = unwrap(angle(y));
```

x 軸に対して周波数ベクトルを作成し、大きさや位相をプロットします。

```
f = (0:length(y)-1)'*100/length(y);
subplot(2,1,1), plot(f,m),
ylabel('Abs. Magnitude'), grid on
subplot(2,1,2), plot(f,p*180/pi)
ylabel('Phase [Degrees]'), grid on
xlabel('Frequency [Hertz]')
```

大きさのプロットは、50 ヘルツの Nyquist 周波数を中心に完全に対称になっています。信号の中の有益な情報は、0 から 50 ヘルツの範囲に見られます。



FFT の長さとスピード

Fourier変換に使うデータ数nを fftへの2番目の引数に設定することができます。

```
y = fft(x,n)
```

この書式で、x のデータ数が n よりも小さいときデータにゼロを付け加え、n よりも長ければ打ち切ります。n を設定しなければ、fft への入力のデータ列数です。

fft の実行時間は、変換する長さに依存します。2 のベキ乗の長さが最も速く処理されます。つぎに、小さな素数に分解される長さのものが速くなります。そして、最も時間を必要とするものは、大きな素数に分解される長さのものです。

逆 FFT 関数 ifft は、変換される長さを引数として使えます。

FFT の実際の適用については、Signal Processing Toolbox に、スペクトル解析に関する多くの関数が含まれています。

14

関数を引数とする関数

関数のまとめ 14-3

MATLAB の中の関数表現 14-4

数学関数のプロット 14-6

関数の最小化と零点の検出 14-9
1 変数関数の最小化 14-9
複数変数関数の最小化 14-10
最小化オプションの設定 14-10
関数のゼロの探索 14-12
実際の問題 14-14
トラブルシュート 14-14
最適化コードを MATLAB 5 のシンタックスに変換 14-15

数値積分（求積法） 14-18
例題： 曲線の長さの計算 14-18
例題： 二重積分 14-19

この章は、*関数を引数とする関数* について記述しています。MATLAB の関数は、数値配列の代わりに数学関数を取り扱うことができます。これらの中には、つぎのことが含まれています。:

関数のまとめ
いくつかの関数を引数とする関数のまとめ

MATLAB の中での関数の表現
MATLAB の中での関数の表現に対するいくつかのガイドライン

数学関数のプロット
数学関数をプロットする fplot の使い方を説明

関数の最小化と零点の検出
最適化に関連した仕事を行なう高水準関数機能を説明

数値積分 (Quadrature)
MATLAB 二次関数に関する説明

注意　関数を引数とする他の関数の使い方についての情報は、" 微分方程式 " と " スパース行列 " を参照してください。

関数のまとめ

関数を引数とする関数は、MATLAB Toolbox の funfun ディレクトリに保存されています。

つぎの表は、この章の中に議論される関数の簡単な記述を与えたものです。関連した関数をカテゴリ別にグループ化しています。

関数のまとめ

カテゴリ	関数	記述
プロット	fplot	プロット関数
最適化とゼロ点の検出	fminbnd	範囲付きの制約をもつ 1 変数関数の最小化
	fminsearch	多変数関数の最小化
	fzero	1 変数関数のゼロ点の検出
数値積分	quad	適応 Simpson 法を使った数値積分
	quadl	適応 Lobatto 法を使った数値積分
	dblquad	二重数値積分

MATLAB の中の関数表現

MATLAB は、M- ファイルまたはインラインオブジェクトの中に MATLAB 関数として、数学関数を表現することができます。たとえば、以下の関数を考えます。

$$f(x)= \frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.04} - 6$$

この関数は、関数を引数とする関数の入力として使われます。

MATLAB 関数として

上に記述した関数を、M- ファイル humps.m で見ることができます。

```
function y = humps(x)
y = 1./((x - 0.3).^2 + 0.01) + 1./((x - 0.9).^2 + 0.04) - 6;
```

この関数 humps を 2.0 で計算するには、@ を使って、humps に対する関数ハンドルを得て、それを、feval に渡します。

```
fh = @humps;
feval(fh,2.0)

ans =
   -4.8552
```

インラインオブジェクトとして

コマンドラインで、数学関数を表す 2 番目の方法は、文字列表現から*インライン*オブジェクトを作成するものです。たとえば、humps 関数のインラインオブジェクトを作成することができます。

```
f = inline('1./((x-0.3).^2 + 0.01) + 1./((x-0.9).^2 + 0.04)-6');
```

そして、2.0 で、f を計算します。

```
f(2.0)
ans =
   -4.8552
```

文字表現法に従って、入力引数名を設定することにより、inlinew p 使って、複数の引数をもつ関数を作成することもできます。たとえば、つぎの関数は、2 つの入力引数 x と y を使います。

```
f= inline('y*sin(x)+x*cos(y)','x','y')
```

```
f(pi,2*pi)
ans =
    3.1416
```

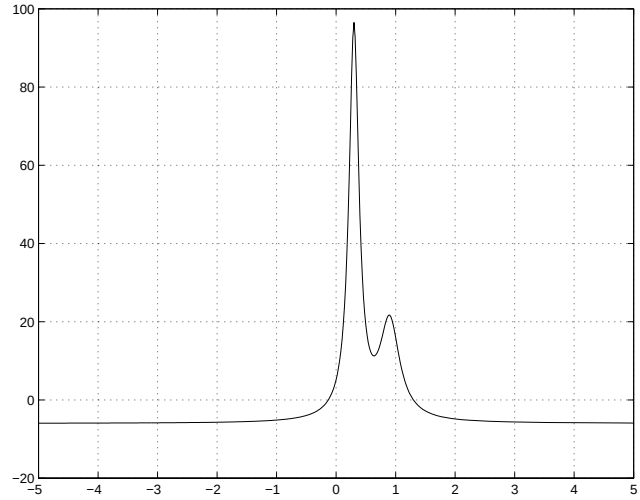
この章で記述するすべての関数は、入力引数として、humps のような関数への関数ハンドル、または、数学関数を定義するインラインオブジェクトのどちらかを使うことができるので、*関数を引数とする関数*と呼ばれます。

注意　関数ハンドルに関する情報は、リファレンスページの function_handle (@), func2str, str2func を参照してください。また、MATLAB ドキュメントのプログラミングとデータタイプ "の" 関数ハンドル” も参照してください。

数学関数のプロット

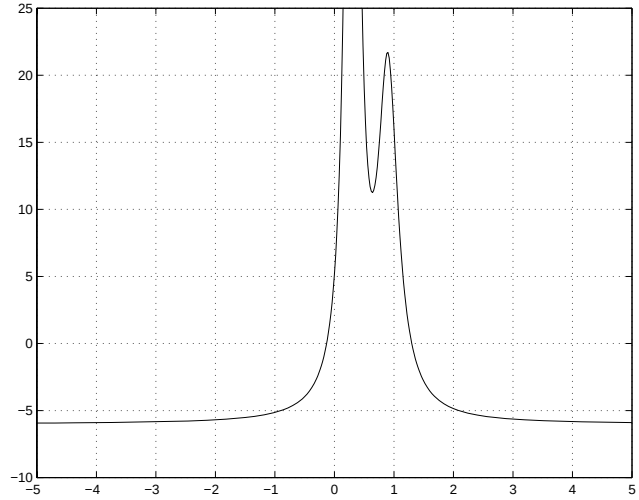
fplot は、設定した数学関数に対して、設定した二つの軸の範囲内で値を計算し、プロットします。ユーザは、*x* 軸のみか、または *x* 軸、*y* 軸共に設定することができます。たとえば、関数 humps を x 軸で、[-5 5] の範囲でプロットします。

```
fplot(@humps,[-5 5])
grid on
```



y 軸の範囲を -10 から 25 までに設定し直し、関数をズームアップすることができます。

```
fplot(@humps,[-5 5 -10 25])
grid on
```



ユーザは、つぎのように、関数 fplot に対するインラインをグラフに渡すことができます。

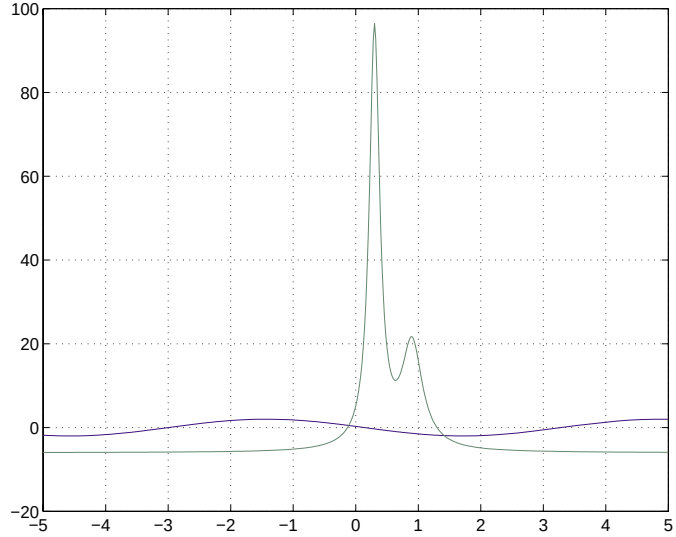
```
fplot(inline('2*sin(x+3)'),[-1 1])
```

関数 fplot への一度のコールで、同じグラフ上に複数の関数をプロットすることができます。一つの関数と共にこれを使用する場合、関数は、列ベクトル x を使い、各列が、x での各値で計算された個々の関数に対応する行列を出力します。

いくつかの関数のインラインオブジェクトを fplot へ渡す場合、インラインオブジェクトは、

```
fplot(inline('[2*sin(x+3), humps(x)]'),[-5 5])
```

と同じように、個々の列が、x の各値で計算した各関数値である行列も出力します。または、同じグラフに 1 番目と 2 番目の数学表現をプロットします。



インライン

```
f= inline('[2*sin(x+3), humps(x)]')
```

は、各関数値を列とする、2 列の行列を計算します。ここで、x は、列ベクトルです。

```
f([1;2;3])
```

は、つぎの出力を得ます。

```
-1.5136   16.0000
-1.9178   -4.8552
-0.5588   -5.6383
```

関数の最小化と零点の検出

MATLAB は、最適化関連の事柄を行なう関数を引数とする高水準の関数をいくつかもっています。この節では、つぎの事柄について記述します。

- 1 変数関数の最小化
- 複数変数関数の最小化
- 最小化オプションの設定
- 1 変数関数の零点の探索
- MATLAB Version 5 シンタックスへのコードの変換

より高精度の最適化機能については、Optimization Toolbox を参照してください。

1 変数関数の最小化

M- ファイルの中でコード化された 1 変数の数学関数が与えられるとすると、関数 fminbnd を使って、与えられた間隔で関数の極小値を算出します。例えば、区間 (0.3 1) の中で関数 humps の最小値を見つけるには、つぎのように行ないます。

```
x = fminbnd(@humps,0.3,1)
```

結果は、つぎのようになります。

```
x =
    0.6370
```

4 番目の引数を、最初の要素に対してゼロでない値をもつ fminbnd を optimset コマンドに転送することにより出力の表を作成しましょう。

```
x = fminbnd(@humps,0.3,1,optimset('Display','iter'))
```

つぎの出力を得ます。

```
Func-count      x          f(x)         Procedure
     1       0.567376     12.9098       initial
     2       0.732624     13.7746       golden
     3       0.465248     25.1714       golden
     4       0.644416     11.2693       parabolic
     5         0.6413     11.2583       parabolic
     6       0.637618     11.2529       parabolic
     7       0.636985     11.2528       parabolic
     8       0.637019     11.2528       parabolic
```

```
     9        0.637052       11.2528          parabolic

x =
    0.6370
```

これは、関数計算が行なわれる度に、カレントの x とそれに対する関数値 f(x) を示しています。fminbnd に対して、一つの関数計算は、アルゴリズムの 1 回の繰り返しに対応します。最後の列は、各繰り返しで使われる手法、すなわち、黄金分割探索または双曲線内挿のどちらかが表示されます。

複数変数関数の最小化

関数 fminsearch は、多くの変数を取り扱ったり、対象となる区間を設定するより、スタートベクトル (初期値ベクトル)x_0 を設定すること以外は、fminbnd と同様です。fminsearch は、スタート時に設定したベクトルの近くの数学的関数の極小値になるベクトル x を出力しようとします。

fminsearch を試すため、3 つの変数 x, y, z からなる変数を定義する M- ファイル three_var を作成します。

```
function b = three_var(v)
x = v(1);
y = v(2);
z = v(3);
b = x.^2 + 2.5*sin(y) - z^2*x^2*y^2;
```

x = -0.6, y = -1.2, z = 0.135 を初期値として、この関数の最小値を見つけましょう。

```
v = [-0.6 -1.2 0.135];
a = fminsearch(@three_var,v)

a =
    0.0000   -1.5708    0.1803
```

最小化オプションの設定

ユーザは、つぎのシンタックスを使って、fminbnd をコールすることにより、いくつかの最適化パラメータを設定するコントロール options ベクトルを設定することができます。

```
x = fminbnd(fun,x1,x2,options)
```

または、つぎのシンタックスをもつ fminsearch を使います。

```
x = fminsearch(fun,x0,options)
```

options は、最適化関数で使われる構造体です。optimset を使って、options 構造体の値を設定します。

```
options = optimset('Display','iter');
```

fminbnd と fminsearch は、つぎのテーブルの中に示される options パラメータのみを使います。Optimization Toolbox で使われるパラメータのすべてを表示している一覧については、optimset のリファレンスページを参照してください。

options.Display	最小化の中間ステップをスクリーン上に表示するか否かを決定するフラグ。'iter' に設定すると、中間のステップが表示されます。'off' に設定すると、表示されません。また、final に設定すると、最終出力のみが表示されます。
options.TolX	終了条件としての x に対するトレランスです。デフォルトでは、1.e-4 です。
options.TolFun	終了条件としての関数に対するトレランスです。デフォルトでは、1.e-4 です。このパラメータは、fminsearch で使われ、fminbnd では使われません。
options.MaxIter	繰り返し最大回数
options.MaxFunEvals	関数を計算する最大回数。デフォルト値は、fminbnd に対して、500 回、fminsearch に対して、200*length(x0) 回です。

つぎのように 4 番目の出力引数を設定して、fminbnd や fminsearch を使うと、

```
[x,fval,exitflag,output] = fminbnd(@humps,0.3,1);
```

または

```
[x,fval,exitflag,output] = fminsearch(@three_var,v);
```

は、計算する関数の回数、繰り返し回数、アルゴリズムを、構造体 output に出力します。

関数のゼロの探索

関数 fzero は、1 変数の関数のゼロを探索します。この関数は、最初の間隔を設定する 2 要素ベクトル、または、1 要素のスタート点のどちらかと共に読み込まれます。初期値 x_0 として fzero を設定すると、関数は符号の変わる部分を挟む区間を探します。この区間が見つからなければ、fzero は NaN を出力します。また、関数の符号が変化する 2 点が既知ならば、2 要素ベクトルを使ってこの初期の区間を設定することができます。そして、fzero は、区間を狭くしようとし、符号の変化する近傍の値を出力します。

-0.2 の近傍で、humps 関数のゼロ点を探すため、fzero を使います。

```
a = fzero(@humps,-0.2)

a =
-0.1316
```

この初期値に対して、fzero は -0.10949 と -0.264 の間で符号が変更するまで、-0.2 の近傍を探索します。この区間は、-0.1316 まで狭められます。-0.1316 で関数は非常にゼロに近付くなることを確認しましょう。

```
humps(a)

ans =
   8.8818e -16
```

humps の関数値が、x = 1 や x = -1 のように、符号の変わる 2 つの位置を知っていると仮定します。

```
humps(1)

ans =
     16

humps(-1)

ans =
   -5.1378
```

そして、この区間から fzero を始め、関数の符号が変化する点の近傍を出力しましょう。fzero の処理過程を表示しましょう。

```
options = optimset('Display','iter');
a = fzero(@humps,[-1 1],options)
```

```
 Func-count        x          f(x)          Procedure

      1             -1       -5.13779        initial
      1              1             16        initial
      2       -0.513876       -4.02235        interpolation
      3        0.243062        71.6382        bisection
      4       -0.473635       -3.83767        interpolation
      5       -0.115287        0.414441       bisection
      6       -0.150214       -0.423446       interpolation
      7       -0.132562       -0.0226907      interpolation
      8       -0.131666       -0.0011492      interpolation
      9       -0.131618     1.88371e-07       interpolation
     10       -0.131618     -2.7935e-11       interpolation
     11       -0.131618     8.88178e-16       interpolation
     12       -0.131618    -9.76996e-15       interpolation

a =
    -0.1316
```

アルゴリズムの進行に伴い Procedure の列に bisection と interpolation のどちらも使われています。例題が最初の区間の代わりに 1 つのスカラの初期値を使って処理を始めようとすると、fzero は符号の変化する区間を含んで探索を行なう一方、初期関数値を算出した後、最初のステップはいくつかの探索ステップを含んでいます

optimset を使って、3 番目の入力引数として、相対誤差の許容範囲を設定することができます。前に読み込んでいるように、空行列では、デフォルトの相対誤差許容範囲の eps を使います。

実際の問題

最適化問題は、収束するため、数回の繰り返しを行ないます。ほとんどの最適化問題は、良い初期推定から始めることが有利になります。これは、実行効率を改良し、極小値の代わりに、大局的な最小値を求めるのに役に立ちます。

より少ない独立変数数で問題を解きながら進めていく方法を使って問題を解く方法は最良の方法です。低次の問題からの解は、適切な射影を使って、より高次な問題の初期値として使われます。

最適化問題の初期のステージで、より簡単な評価関数を使い、停止基準を緩くすることも、計算時間を軽減することができます。このようなアプローチは、極小値を避けることにより、より良い結果を生じます。

トラブルシュート

典型的な問題や、推薦される取り扱いをまとめます。

問題	推奨される処理
fminbnd、または、fminsearch を使って求められる解が大域的な最小値に思えない。	ユーザの問題が連続で、たった一つの最小値しかもたない限り大域的最小値である保証はありません。種々の初期値で最適化を始めることが、大域的最小値の位置を見つけたり、最小値がたった一つしかないことを確認する方法の手助けになります。可能な部分では、異なる方法で結果を確認してください。
最適化問題は、f を計算できない x の値をもつ可能性があります	非可解領域になると、大きな正の値を与えるためにペナルティ関数を含ませるようにユーザの関数を変更します。
最小化ルーチンは、無限ループになったり、最小でない解 (fzero の場合はゼロにならない) を出力します。	ユーザの目的関数が Inf,NaN または複素数値を出力する可能性があります。最適化ルーチンは、実数値のみ一つ出力されることが期待できます。他の値では、予想できない結果になる可能性があります。ユーザの目的関数 M- ファイルにコードを加えることで、実数のみが出力されることを確認できます (関数 isreal,isfinite を使用)。

最適化コードを MATLAB 5 のシンタックスに変換

ほとんどの関数名と呼び出しシーケンスは、新規の機能に適用するようにVersion 5 において変更され、入出力変数の役割が明確化されています。

つぎの表は、MATLAB が提供する最適化関数と Version 5 で名前が変更された関数を示します。

旧 (Version 4) 名	新規 (Version 5) 名
fmin	fminbnd
fmins	fminsearch
foptions	optimget, optimset
fzero	fzero(関数名は変更なし)
nnls	lsqnonneg

この節では、以下を説明します。

- 新規の関数 optimset と optimget を使ってデフォルトのパラメータ設定を変更する方法を説明します。
- 新規の呼び出しシーケンスの理由とコードの変換方法を説明します。

この節の情報に加えて、新規関数およびそれらの引数に関する情報については、個々の関数のリファレンスページを参照してください。

optimset と optimget の利用

関数 optimset は、デフォルトのパラメータ設定変更のための foptions が変更されています。optimset は最適化ルーチンで利用されるパラメータを含む options 構造体を作成します。最適化ルーチンの最初の呼び出しで、options 構造体が与えられない場合、あるいは空である場合は、デフォルトのパラメータが生成されます。詳細は、optimset のリファレンスページを参照してください。

新規の呼び出しシーケンス

MATLAB Version 5 は、呼び出しシーケンスにおいて以下の点が変更されています。

- 各関数は、パラメータを最適化関数に適合させる options 構造体を取ります (optimset, optimget を参照)。
- 新規のデフォルト出力は、関数が収束しない場合は情報を与えます (Version 4 のデフォルトは出力をもちません。Version 5 はデフォルトとして 'final' を使い、新規のデフォルトは options.display = 'notify' です)。
- 各関数は、終了状態を示す exitflag を出力します。
- 各関数は、その関数に関連する問題の解に関する情報を含む output 構造体をもちます。

以下の節では、旧名の関数と呼び出しシーケンスを新規のものに変換する方法を説明します。以下に示すのは、最も一般的な場合であり、すべての利用可能な入力および出力引数に関するものです。これらの引数の多くはオプションであることに注意してください。詳細は関数リファレンスページを参照してください。

fmin から fminbnd への変換 . Version 4 では、fmin の呼び出しには以下を利用しました。

```
[X,OPTIONS] = fmin('FUN',x1,x2,OPTIONS,P1,P2,...);
```

Version 5 では、fminbnd は以下のように呼び出します。

```
[X,FVAL,EXITFLAG,OUTPUT] = fminbnd(@FUN,x1,x2,...
                                   OPTIONS,P1,P2,...);
```

fmins から fminsearch への変換 . Version 4 では、fmins の呼び出しには以下を利用しました。

```
[X,OPTIONS] = fmins('FUN',x0,OPTIONS,[],P1,P2,...);
```

Version 5 では、fminsearch は以下のように呼び出します。

```
[X,FVAL,EXITFLAG,OUTPUT] = fminsearch(@FUN,x0,...
                                      OPTIONS,P1,P2,...);
```

fzero の新規形式への変換 . Version 4 では、fzero の呼び出しには、以下を利用しました。

```
X = fzero('F',X,TOL,TRACE,P1,P2,...);
```

Version 5 では、引数 TRACE と TOL を以下で置き換えます。

```
if TRACE == 0,
   val = 'none';
elseif TRACE == 1
   val = 'iter';
end
OPTIONS = optimset('Display',val,'TolX',TOL);
```

以下のように fzero を呼び出します。

```
[X,FVAL,EXITFLAG,OUTPUT] = fzero(@F,X,OPTIONS,P1,P2,...);
```

nnls から lsqnonneg への変換． Version 4 では、nnls の呼び出しには、以下を利用しました。

```
[X,LAMBDA] = nnls(A,b,tol);
```

Version 5 では、引数 tol を以下で置き換えます。

```
OPTIONS = optimset('Display','none','TolX',tol);
```

以下のように lsqnonneg を呼び出します。

```
[X,RESNORM,RESIDUAL,EXITFLAG,OUTPUT,LAMBDA] =    lsqnonneg(A,b,X0,OPTIONS);
```

数値積分（求積法）

関数 $F(x)$ の断面の下の部分の面積が $F(x)$ を数値積分することにより求められます。この過程を*求積法*と言います。MATLAB の求積法関数は、以下の通りです。

quad	適応 Simpson 求積法の使用
quadl	適応 Lobatto 求積法の使用
dblquad	数値二重積分

humps.m で定義した関数を、0 から 1 の範囲で定義して積分しましょう。

```
q = quad(@humps,0,1)
```

```
q =
    29.8583
```

quad と quadl は、共に、再帰的に演算します。各方法が、特異性の可能性を検知すると、ワーニングを出力します。

quad や quadl に対して、積分の相対誤差トレランスを設定するために、4 番目の引数を設定することができます。ゼロでない 5 番目の引数が、quad、または、quadl に渡された場合、関数は、処理経過を追跡します。

これらの関数の使用例を示す 2 つの例題を用意しています。

- カーブの長さを計算
- 二重積分

例題：曲線の長さの計算

曲線の長さを計算するのに quad、または、quadl を使います。まず、方程式により曲線をパラメータ化します。

$$x(t) = \sin(2t), \qquad y(t) = \cos(t), \qquad z(t) = t$$

ここで、$t \in [0, 3\pi]$ です。

この曲線の 3 次元プロットを行ないます。

```
t = 0:0.1:3*pi;
plot3(sin(2*t),cos(t),t)
```

弧の長さの式から、曲線の長さはパラメータ化された方程式の微係数のノルムの積分に等しいことがわかります。

$$\int_0^{3\pi} \sqrt{4\cos(2t)^2 + \sin(t)^2 + 1}\ dt$$

関数 hcurve で曲線を計算します。

```
function f = hcurve(t)
f = sqrt(4*cos(2*t).^2 + sin(t).^2 + 1);
```

この関数の積分は、quad で行ないます。

```
len = quad(@hcurve,0,3*pi)

len =
   1.7222e+01
```

そして、この曲線の長さは、約 17.2 になります。

例題： 二重積分

つぎの数値積分を考えましょう。

$$\int_{ymin}^{ymax} \int_{xmin}^{xmax} f(x,y)\ dxdy$$

たとえば、 $f(x,y) = y\sin(x) + x\cos(y)$ を考えます。最初のステップは、計算する関数を作成することです。関数は、ベクトル入力が与えられたとき、ベクトル出力を返すものでなければなりません。ユーザは、変数が内側になっているか、外側になっているかを考えなければなりません。この例題では、内側の変数は x で、外側の変数は y （ 積分の順番は、*dxdy*) です。この場合、積分関数は、つぎのようになります。

```
function out = integrnd(x,y)
out = y*sin(x) + x*cos(y);
```

積分を行なうために、2 つの関数が funfun ディレクトリの中で使用可能になります。まず、dblquad が、コマンドラインから直接読み込まれます。この M- ファイルは quad を使って外側のループを計算します。各繰り返しで、quad は内側ループを計算する 2 番目の関数を読み込みます。

2 重積分を計算するため、つぎのステートメントを使います。

```
result = dblquad(@integrnd,xmin,xmax,ymin,ymax);
```

最初の引数は、積分される関数名です。2 番目から 5 番目までの引数は、つぎのようになります。

xmin	内側積分の下限
xmax	内側積分の上限
ymin	外側積分の下限
ymax	外側積分の上限

ここでは、dblquad の使い方の例を示します。

```
xmin = pi;
xmax = 2*pi;
ymin = 0;
ymax = pi;
result = dblquad(@integrnd,xmin,xmax,ymin,ymax)
```

結果は、-9.8698 になります。

デフォルトにより、dblquad は、quad を使います。quadl を使って前の例を積分するため、つぎのステートメントを使います。

```
result = dblquad(@integrnd,xmin,xmax,ymin,ymax,[],@quadl);
```

また、quad と同じ読み込み方、求積関数が引数をもっている場合は、ユーザ定義の求積法関数を dblquad に渡すこともできます。

15

微分方程式

ODE 問題と DAE 問題に対する初期値問題 15-3
ODE 関数のまとめ 15-3
初期値 ODE 問題の紹介 15-5
初期値問題のソルバ 15-6
ODE 問題の表現 15-9
ODE ソルバ性能の改良 15-15
例題：ODE 初期値問題ソルバの適用 15-29
質問と回答とトラブルシュート 15-48

ODEs に対する境界値問題 15-56
BVP 関数のまとめ 15-56
境界値 ODE 問題の紹介 15-57
境界値問題ソルバ 15-59
BVP 問題の表現 15-62
良い初期推定の作成法 15-66
BVP ソルバ性能の改良 15-69

偏微分方程式 15-75
PDE 関数のまとめ 15-75
PDE 問題の紹介 15-76
MATLAB 偏微分方程式ソルバ 15-77
PDE 問題の表現 15-80
PDE ソルバ性能の改良 15-86
例題： 電気力学問題 15-86

参考文献 15-91

この章は、MATLAB を使った微分方程式の数値解を取り扱います。

ODE 問題や DAE 問題に対する初期値問題

常微分方程式 (ODEs) と微分代数方程式 (DAEs) の解について説明します。ここで、興味のある解は、独立変数の与えられた初期値で初期条件を満足します。

ODE 問題に対する境界値問題

ODE 問題の解について説明します。ここで、興味のある解は、ある境界条件を満足します。境界条件は、独立変数の初期値と最終値での解の値の間の関係を満足します。

偏微分方程式

1 つの空間変数と時間変数からなる双曲線偏微分方程式と楕円偏微分方程式に対する初期境界値問題の解の説明を行ないます。

参考文献

この章で記述されている概念を示す参考文献を一覧表示しています。

注意　関数表の中で、共通に使用する関数が、まず最初にリストされ、より高度なものがそれに続きます。同じものが、プロパティテーブルにも示されます。

ODE 問題と DAE 問題に対する初期値問題

この節は、常微分方程式 (ODE) と微分代数方程式 (DAE) の初期値問題 (IVP) を解くために、どのように MATLAB を使用するかを記述しています。つぎの事柄を説明しています。

- MATLAB の ODE ソルバ、関連関数、デモのまとめ
- ODE の紹介と初期値問題
- ODE ソルバの記述と、基本的なシンタックス
- MATLAB での IVP の表現に関する一般的なサンプル
- ソルバの性能を向上させるために、デフォルトの積分プロパティを変更することに関する議論
- MATLAB で解くことのできる IPV 問題のいくつかの例題
- MATLAB の ODE ソルバに関して、良く聞かれる問題の解といくつかのトラブルに関する助言

ODE 関数のまとめ

初期値 ODE 問題のソルバ

つぎのような、MATLAB 初期値問題のソルバがあります。表には、各ソルバを使って解くことのできる問題の種類と各ソルバが使用している手法をリストしています。

ソルバ	解くことのできる問題の種類	手法
ode45	ノンスティッフな微分方程式	Runge-Kutta
ode23	ノンスティッフな微分方程式	Runge-Kutta
ode113	ノンスティッフな微分方程式	Adams
ode15s	スティッフな微分方程式と代数方程式	NDFs(BDFs)
ode23s	スティッフな微分方程式	Rosenbrock
ode23t	中程度のスティッフな微分方程式と代数方程式	Trapezoidal rule
ode23tb	スティッフな微分方程式	TR-BDF2

ODE オプションの取り扱い

積分プロパティ名と値を含む Options 構造体がソルバに渡されて、問題の解に影響を与えます。つぎの関数を使って、Options 構造体を作成したり、変更したり、アクセスすることができます。

関数	詳細
odeset	ODE ソルバへの入力に対する options 構造体の作成と変更
odeget	odeset を使って作成した Options 構造体からプロパティを取得

ODE ソルバ出力関数

出力関数が設定されると、ソルバは、連続的に行なう積分ステップ毎に指定した関数をコールします。ユーザは、odeset を使って、OutputFcn プロパティのように、これらのサンプル関数の一つを設定でき、また、これらを変更して、ユーザ自身の関数を作成できます。

関数	詳細
odeplot	時系列プロット
odephas2	2 次元の位相面プロット
odephas3	3 次元の位相面プロット
odeprint	コマンドウインドウにプリント

ODE 初期値問題のデモ

これらのデモは、MATLAB の中で解くことのできる問題の種類を示しています。MATLAB の Help Browser から、デモ名をクリックして、エディタでデモコードを見ることができます。デモを実行するには、コマンドラインで、demoname と入力してください。

デモ	詳細
amp1dae	スティッフな DAE - 電子回路
ballode	簡単なイベントの生じる位置 - バウンドするボール

デモ	詳細
batonode	時変、状態に依存した質量行列をもつ ODE - バトンの運動
brussode	スティッフな大規模問題 - 化学反応の拡散作用 (Brusselator)
burgersode	状態量に非常に依存した質量行列をもつODE - メッシュを変更する手法を使って、Burger の方程式を解く
fem1ode	時変の質量行列をもつスティッフな問題 - 有限要素法
fem2ode	定質量行列をもつスティッフな問題 - 有限要素法
hb1dae	保存則下でのスティッフな DAE 問題
hb1ode	非常に長い区間でスティッフな問題を解く - Robertson 化学反応
orbitode	より進んだイベントの生じる位置の検出 - 制約付きの 3 体問題
rigidode	ノンスティッフな問題 - 外力のない剛体の Euler 方程式
vdpode	パラメータ化した van der Pol 方程式 (大きなμについては、スティッフ)

初期値 ODE 問題の導入

常微分方程式とは

MATLAB ODE ソルバは、*常微分方程式*を取り扱うように設計されています。常微分方程式は、単一の独立変数t , 通常は、*時間*で、それに関する従属変数yの一回、または、複数回の微分係数を含んでいるものです。t に関するy の微分は、y' として定義され、2 階微分はy'' 等、定義されます。多くの場合、$y(t)$ はベクトルで、その要素は $y_1, y_2, \ldots, y_n$ です。

初期条件を使って、興味のある解を指定

一般的に、与えられた ODE を満足する多くの関数$y(t)$ があり、対象になる解を指定するには、付加的な情報を必要とします。*初期値問題* で、対象の解は、固有の *初期条件* を満足します。すなわち、y は、与えられた初期時間t_0 でy_0 を満足します。初期値問題は、つぎのようになります。

$$y' = f(t, y)$$
$$y(t_0) = y_0 \qquad \textbf{(15-1)}$$

関数 $f(t, y)$ が十分に平滑な場合、この問題は、ただ一つの解をもちます。一般に、解に対する解析的な表現はありません。それで、MATLAB ODE ソルバの一つを使って、数値手法により、$y(t)$ を近似する必要があります。

初期値問題のソルバ

MATLAB ODE ソルバ関数は、ODEs(式 15-1) に対する IVP を解くために、数値的な積分手法を実行します。初期条件をもつ初期時間から始め、設定した時間区間で、各時間ステップで、解を計算します。時間ステップに関する解が、ソルバの誤差のトレランスの規範を満足している場合、このステップは、うまく処理されています。規範を満足していない場合は、処理が失敗したことになります。そして、ソルバは、時間間隔を小さくして、再度、計算を試みます。

この節では、つぎの事柄を記述しています。

- ノンスティッフな ODE 問題やスティッフな ODE 問題を解く MATLAB ソルバ
- ODE ソルバの基本的シンタックス
- 付加的な ODE ソルバの引数

15-25 ページの " 質量行列と DAE プロパティ " では、より一般的な問題の解法を説明します。

これらのソルバに対するデモファイルに関する情報と同様に、任意の MATLAB 関数に対するシンタックスの情報を、MATLAB Help ブラウザを使って、得ることができます。

ノンスティッフな問題に対するソルバ

ノンスティッフな問題用に 3 つのソルバが用意されています。

ode45　陽的 Runge-Kutta(4,5) 式をベースにした Dormand-Prince 法です。これは、*1 段階法* ソルバです。$y(t_n)$ の計算は、すぐ前の点 $y(t_{n-1})$ での解のみが必要になります。一般に、ode45 は、大部分の問題に対して、最初の試みとして適用するのに適切な関数です。

ode23　陽的 Runge-Kutta(2,3) 式をベースにした Bogacki and Shampine による手法です。粗い許容誤差で計算するときや少しスティッフなシステムの場合、ode45 より有効になることがあります。ode45 と同様に ode23 は、1 段階法ソルバです。

ode113　可変次数の Adams-Bashforth-Moulton PECE 法ソルバです。厳しい許容誤差設定された問題や ODE 関数が計算するのに特に時間のかかるような問題では、ode45 より効率的です。ode113 は、*多段階法ソルバ*です。すなわち、解を求めるために、以前の数点を使います。

スティッフ問題に対するソルバ

すべての困難な問題は、必ずスティッフであると云うことではなく、すべてのスティッフな問題は、そのために専用に設計されていないソルバにとっては困難なものになります。スティッフソルバは、他のソルバと全く同じように使うことができます。しかし、問題に対する付加的な情報を与えることで、スティッフなソルバの効率をかなり高めることができます (15-15 ページの "ODE ソルバ性能の改良 " を参照)。

スティッフな問題に対して、設計されたソルバが 4 種類あります。

ode15s　可変次数の多段階法に基づく数値微分公式 (NDFs) です。オプションとして、通常効率的には下がりますが、後退差分方程式 (BDF、Gear 法として有名) を使うこともできます。ode113 用に ode15s も、多段階法ソルバです。問題がスティッフかどうか、または ode45 がうまく機能しないか、または効率が悪いなら、ode15s を使ってください。

ode23s　2 次の Rosenbrock の公式を改良したものをベースにしています。これは、単段階法なので、粗い許容範囲では ode15s よりも効率的です。ode15s では、効率的でないいくつかの種類のスティッフな問題を解くことができます。

ode23t　" フリー " の内挿を使って、台形則を実行します。問題が中程度のスティッフさをもち、数値的な減衰をもたない解を必要とする場合、このソルバを使ってください。

ode23tb　最初の段階で、台形則ステップをもつ暗示的な Runge-Kutta 法を、二番目の段階で、2 次の後退微分方程式を使う、TR-BDF2 を実行します。ode23s と同様に、このソルバは、急激なトレランスで、ode15s よりも、より効率的に機能します。

ODE ソルバの書式設定法

すべての ODE ソルバ関数は、最も適切な手法がはっきりしていなくても種々の数値法を使い易くするために、書式を同じにしています。種々の手法を同じ問題に適用するためには、ODE ソルバ関数名を単に変えるだけ十分です。すべてのソルバ関数で共通の最も簡単な記法は、

```
[t,y] = solver(odefun,tspan,y0)
```

ここで、*solver* は、ここまでに説明した ODE ソルバのうちの一つです。

基本的な入力引数は、つぎのものです。

odefun　ODE システムを記述するファイルの名前を表す文字列

```
dydt = odefun(t,y)
```

ここで、t はスカラ、dydt と y は列ベクトル

tspan　積分区間を設定するベクトル。ソルバは、tspan(1) で初期条件を設定し、tspan(1) から tspan(end) まで積分します。

2 要素ベクトル tspan = [t0 tf] に対して、ソルバは、各積分ステップで計算した解を出力します。2 要素以上の要素をもつ tspan ベクトルに対して、ソルバは、つぎに記述するように、与えられた時刻で解を出力します。時間ベクトルは、増加、または、減少のどちらかの順に配置されている必要があります。

2 つ以上の要素をもつ tspan を設定すると、ソルバが、tspan(1) から tspan(end) までの区間に渡り使用する内部時間ステップに影響を与えません。MATLAB ODE suite の中のすべてのソルバは、基本式を連続的に拡張することにより、出力値を得ます。ソルバは、tspan で設定した時刻点を必ずしも正確にステップを処理しませんが、指定した時刻点で作成された解は、内部の時刻点で計算された解と同じ度合いの精度をもっています。

2 要素以上をもつ tspan を設定することは、計算の効率化に影響を与えませんが、大きなシステムに対しては、メモリの管理に影響を与えます。

y0　問題の初期条件ベクトル。

15-5 ページの " 初期値 ODE 問題の導入 " を参照してください。

出力引数は、つぎのものです。

t　　計算された時間ステップの列ベクトル

y　　解を表す計算結果の配列。y の中の各々の行は、t の対応する行の中で出力される時刻での解に対応します。

ODE ソルバの付加的な引数

よりアドバンストなアプリケーションに関して、ODE ファイルに対して、入力引数ソルバオプションと付加的なパラメータを設定することもできます。

options　　オプションのパラメータ構造体を使って、デフォルトの積分パラメータを変更します。4 つの入力引数を使います。

```
[t,y] = solver(odefun,tspan,y0,options)
```

15-15 ページの "ODE ソルバ性能の改良 " は、ユーザが設定できる構造体の作り方やプロパティの記述法を説明しています。

p1,p2...　　ソルバが、odefun に渡すパラメータ。

```
[t,y] = solver(odefun,tspan,y0,options,p1,p2...)
```

ソルバは、任意の入力パラメータを、options 引数の後に設定して、odefun や options の中に設定した関数に渡します。オプションを設定しない場合は、options = [] を使ってください。関数 odefun は、つぎの型をしています。

```
dydt = odefun(t,y,p1,p2,...)
```

例題は、15-12 ページの " 付加的なパラメータを ODE 関数に渡す " を参照してください。

ODE 問題の表現

この節は、MATLAB ODE ソルバの一つを使って、初期値 ODE 問題を解く過程を記述します。van der Pol 方程式を使います。

- MATLAB を使って、ODE 問題を解くために必要なステップを記述します。
- スティッフな ODE 問題の例題として

注意 詳細は、15-8 ページの "ODE ソルバの書式設定法 " を参照してください。

例題：MATLAB で IVP ODE 問題を解く (van der Pol 方程式、ノンスティッフ)

この例題は、MATLAB を使って、初期値 ODE 問題を解くために必要なステップを示し、説明します。

1 問題を 1 階の ODE 問題として書き換えます． ODEs は、1 階以上の微係数と複数の従属変数を含んでいます。MATLAB ODE ソルバを使って、このような方程式をつぎの型をした一階微分方程式の等価なシステムとして表現します。

$$y' = f(t,y)$$

ユーザは、任意の常微分方程式を

$$y^{(n)} = f(t,y,y',\ldots,y^{(n-1)})$$

つぎのように代入して、1 階の微分方程式として、記述できます。

$$y_1 = y,\quad y_2 = y',\quad \ldots,\quad y_n = y^{(n-1)}$$

結果は、n 個の 1 階の ODEs の等価なシステムになります。

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_3 \\ &\vdots \\ y_n' &= f(t,y_1,y_2,\ldots,y_n) \end{aligned}$$

たとえば、van der Pol 方程式 (二階微分) を書き換えます。

$$y_1'' - \mu(1-y_1^2)y_1' + y_1 = 0$$

ここで、$\mu > 0$ は、スカラパラメータです。そして、$y'_1 = y_2$ を代入して、つぎの 1 階 ODEs の結果を得ます。

$$y'_1 = y_2$$
$$y'_2 = \mu(1-y_1^2)y_2 - y_1$$

2 MATLAB で、1 階の ODEs のシステムをコード化． 1 階の ODEs のシステムとして方程式を表現すると、MATLAB ODE ソルバが使用できる型にそれをコード化します。関数は、つぎの型になります。

```
dydt = odefun(t,y)
```

t と y が、関数の最初の 2 つの引数になる必要がありますが、関数は、それらを必ず使用するとは限りません。出力 dydt は、列ベクトルで、y の微係数です。

下のコードは、MATLAB 関数 vdp1 で表した van der Pol システムです。関数 vdp1 は、$\mu = 1$ を仮定しています。y_1 と y_2 は、2要素ベクトルの y(1) と y(2) になります。

```
function dydt = vdp1(t,y)
dydt = [y(2); (1-y(1)^2)*y(2)-y(1)];
```

vdp1 は、引数 t と y を受け入れますが、計算では、t を使いません。

3 ソルバを問題に適用． Decide 問題を解くために使用するソルバを決定します。そして、ソルバをコールし、1 階の ODEs システムを記述するために作成した関数を渡し、それと共に、問題を解くための時間間隔と初期条件ベクトルを渡します。ODE ソルバの記述に関しては、15-6 ページの " 初期値問題のソルバ " と ODE ソルバリファレンスページを参照してください。

van der Pol システムに対して、初期値 y(1) = 2 と y(2) = 0 を使って、時間区間 [0 20] で、ode45 を使用します。

```
[t,y] = ode45(@vdp1,[0 20],[2; 0]);
```

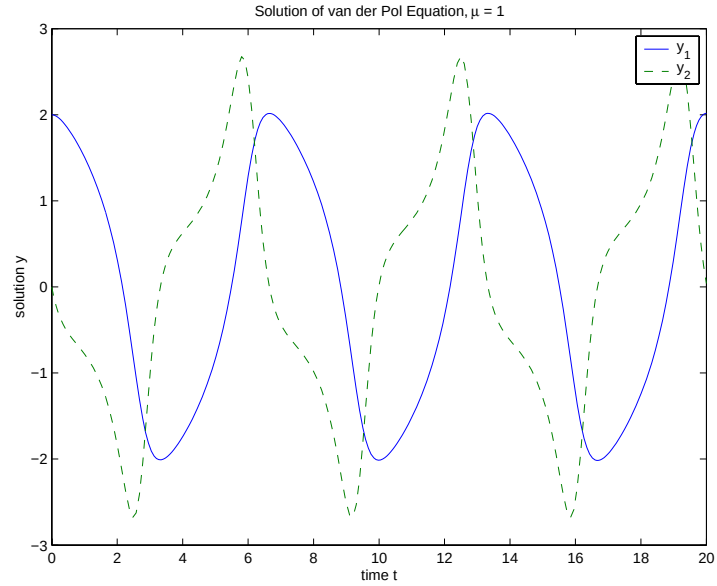
この例題は、ode45 への関数ハンドルとして、@ を使って、vdp1 に渡します。出力結果は、時間点 t を表す列ベクトルと解の配列 y です。計算結果の行列 y の各列は、列ベクトル t で出力される時刻に対応します。y の最初の列は、y_1 に対応し、2 番目の列は、y_2 に対応します。

注意 関数ハンドルに関する情報に関しては、MATLAB ドキュメントの中の function_handle (@), func2str, str2func のリファレンスページと "Programming with MATLAB" の M-Files Programming の節を参照してください。

---

4 ソルバ出力の可視化． plot コマンドを使って、ソルバ出力を可視化することができます。

```
plot(t,y(:,1),'-',t,y(:,2),'--')
title('Solution of van der Pol Equation, \mu = 1');
xlabel('time t');
ylabel('solution y');
legend('y_1','y_2')
```



また、ソルバの出力関数を使って、出力を処理することもできます。ソルバは、連続的な時間ステップの各ステップの後で、積分プロパティ OutputFcn の中に設定し、関数をコールします。odeset を使って、希望する関数を OutputFcn に設定します。詳細は、15-4 ページの "ODE ソルバ出力関数 " を参照してください。

付加的なパラメータを ODE 関数に渡す． 関数は、options 引数の後に任意の入力引数を設定して、ODE 関数やユーザが options に設定した関数に渡します。たとえば、

1 コードの中にmuを明示的に設定するのではなく、muパラメータを渡すことにより、van der Pol 関数を一般化します。

```
function dydt = vdp1(t,y,mu)
dydt = [y(2); mu*(1-y(1)^2)*y(2)-y(1)];
```

2 ソルバへのコールの中の options 引数の後に、設定することで、パラメータ mu を関数 vdp1 に渡します。この例題では、options = [] を使っています。

```
[t,y] = ode45(@vdp1,tspan,y0,[],mu)
```

calls

```
vdp1(t,y,mu)
```

これらの関数をベースにした例題は、vdpode を参照してください。

例題 : van der Pol 方程式 , m = 1000(スティッフ)

注意 MATLAB の中で、初期値 ODE 問題を解くための詳細な例は、15-10 ページの " 例題 : MATLAB で IVP ODE 問題を解く (van der Pol 方程式、ノンスティッフ)" を参照してください。

この例題は、*スティッフ*な問題を取り扱います。スティッフな問題では、シミュレーション時間に比べ、非常に短い時間スケールで解は変化しますが、興味のある解 (適切な精度を保った数値解) は、はるかに大きい時間スケールでしか変化しません。スティッフ問題用に設計されていない方法では、最も速い変化を解くために十分な細かさの時間間隔を使うために、時間変化のゆっくりしている部分では、効率の悪いものになります。

μ を大きくして 1000 にすると、van der Pol 方程式の解は、急激に変化し、より長い時間スケールで振動を示します。初期値問題の解を近似することは、非常に難しい問題です。この特別な問題はスティッフなので、ode45 のようなスティッフに対応しないソルバは、非現実的な効率の悪いものになります。スティッフなソルバ ode15s は、このような問題に対して作られたものです。

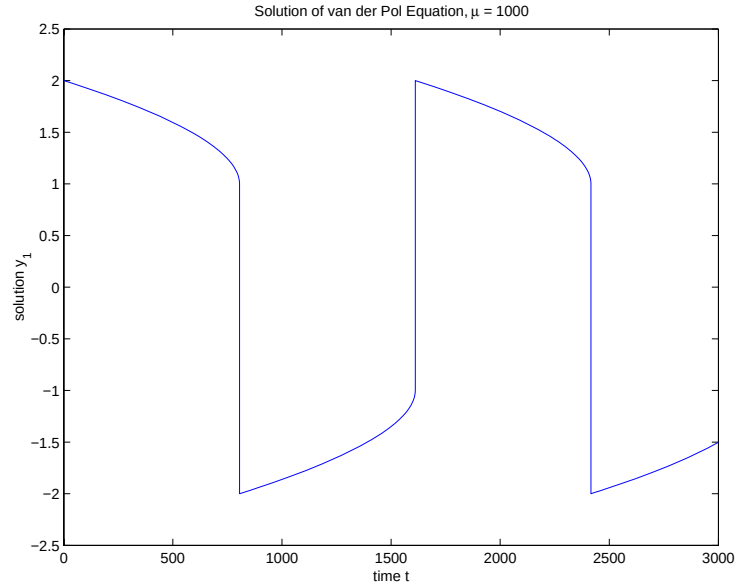
つぎのコード vdp1000 は、μ = 1000 のとき、ODE ファイルの中に van del Pol システムを、どのように表現するかを示したものです。

```
function dydt = vdp1000(t,y)
dydt = [y(2); 1000*(1-y(1)^2)*y(2)-y(1)];
```

注意　この例題は、ODE 関数の中に、μ をハードコードしています。vdpode デモは、同じ問題を解いていますが、ODE 関数への付加的な引数として、ユーザ定義のμを渡しています。詳細は、15-9 ページの "ODE ソルバの付加的な引数 " を参照してください。

さて、関数 ode15s を使って、vdp1000 を解きましょう。[2; 0] の初期条件ベクトルはそのままですが、計算時間を [0 3000] にします。スケーリングの目的 (積分ステップ間隔とシステムの振幅周期を調べる) で、y(t) の最初の成分をプロットしましょう。

```
[t,y] = ode15s(@vdp1000,[0 3000],[2; 0]);
plot(t,y(:,1),'-');
title('Solution of van der Pol Equation, \mu = 1000');
xlabel('time t');
ylabel('solution y_1');
```



ODE ソルバ性能の改良

ODE ソルバのデフォルトのプロパティは、一般的な問題を取り扱うために設定されています。ある場合においては、これらのデフォルトを書き換えて、ODE ソルバの性能を改良することができます。これを行うために、引数 options に含まれる 1 つまたは複数のプロパティをソルバに設定します。

```
[t,y] = solver(odefun,tspan,y0,options)
```

この節では、つぎのことを記述します。

- options 構造体を作成するための create, modify, query の使い方
- options 構造体の中で、使用可能なプロパティの記述。

つぎと、それに続くプロパティ表の中で、最も一般的に使われているプロパティが最初にリストされ、より高度なものがそれに続いて表示されています。

ODE プロパティカテゴリ

プロパティカテゴリ	プロパティ名
誤差のコントロール	RelTol, AbsTol, NormControl
ソルバ出力	OutputFcn, OutputSel, Refine, Stats
ヤコビアン行列	Jacobian, JPattern, Vectorized
ステップサイズ	InitialStep, MaxStep
質量行列と DAE 問題	Mass, MStateDependence, MvPattern, MassSingular, InitialSlope
イベントの生じる位置	Events
ode15s-specific	MaxOrder, BDF

ODE Options 構造体の作成と取り扱い

Option 構造体の作成． 関数 odeset は、ODE ソルバに引数として渡すことのできる options 構造体を作成します。odeset は、つぎのような書式で与えたいプロパティの名前とその値とを組にして設定します。

```
options = odeset('name1',value1,'name2',value2,...)
```

このようにして、指定された名前のプロパティに特定の値を設定した options を作成します。いくつかの設定されていないプロパティは、ソルバの中ではデフォルトを使います。すべてのプロパティに対して、プロパティ名をユニークに識別できる最初の数文字のみの入力でも構いません。odeset は、プロパティ名として、大文字、小文字の区別を行いません。

入力引数がないとき、odeset は全てのプロパティ名とその取り得る値を示し、設定しているデフォルト値には { } を付けています。

設定されている options の構成の変更． つぎの書式で、すでに設定されている引数 options を変更します。

```
options = odeset(oldopts,'name1',value1,...)
```

これは、既に存在している構造体 oldopts の値を、プロパティ名 / 値の組で設定したもので、書き直したり、新しい組を構造体に加えて、構造体 options を書き換えることができます。出力引数として、書き換えられた構造体が出力されます。同様にして、つぎのコマンド

```
options = odeset(oldopts,newopts)
```

では、oldopts と newopts を組み合わせます。2 番目の引数に含まれるプロパティが、最初の引数に含まれるプロパティと一致している場合、2 番目の引数に含まれるプロパティで書き換えられて、出力引数に、新しくなったプロパティを含め、構造体全体が出力されます。

Options の確認． 関数 odeget で設定した options 構造体から、あるプロパティの値を取り出すために、関数 odeset を使います。

```
o = odeget(options,'name')
```

これによって、options の中で設定していたプロパティ値を取り出します。設定されていない値に対しては空行列 [] が出力されます。

odeset と同様に、プロパティ名はそれを特定できる最初の何文字かの設定で十分で、プロパティ名は、大文字、小文字の区別を行いません。

誤差のコントロールプロパティ

各積分ステップにおいて、ソルバは、解の i 番目の要素に、局所的な誤差を計算します。この誤差が、指定した相対許容誤差 RelTol、絶対許容誤差 AbsTol を比べて、小さいか、等しくなければなりません。

```
|e(i)| 」 max(RelTol*abs(y(i)),AbsTol(i))
```

ルーチン問題に対して、ODE ソルバは、ユーザが要求した精度におおまかに等しい精度にします。これは、長い区間に渡って積分する問題に対しては、精度を低くしたものを使い、また、不安定な問題についても精度を低くしたものを使います。難しい問題は、デフォルトの値よりも条件の厳しい許容範囲を使うことを必要とします。相対精度に対しては、RelTol を調整します。絶対誤差許容範囲に対しては、解の要素のスケーリングが重要になります。|y| が、AbsTol よりやや小さい場合、ソルバは、y の中の正しい桁数を保証できません。解の要素のスケールを見つけるために、何回か問題を解く必要があります。

おおまかに言って、スレッシュホールド AbsTol(i) より小さいものを除いて、すべての解の要素の中で、RelTol の桁数を必要とすることを意味しています。小さくて、要素 y(i) に興味がない場合、y(i) の中にいくつかの正しい桁数を得るのに十分小さな AbsTol(i) を設定する必要があります。それで、興味のある要素を正確に計算することができます。

つぎの表は、誤差のコントロールプロパティを記述しています。odeset を使って、プロパティを設定してください。

ODE 誤差コントロールプロパティ

プロパティ	値	詳細
RelTol	正のスカラ {1e-3}	解ベクトル y のすべての成分に適用する相対的な誤差許容範囲。これは、解の各要素のサイズに対するエラーの尺度になります。大雑把に言って、スレッシュホールド AbsTol(i) より小さいものを除いて、解のすべての要素の中での正しい桁数をコントロールするものです。 デフォルトは、1e-3 で、精度 0.1% に対応します。
AbsTol	正のスカラ、または、ベクトル {1e-6}	解ベクトルの各成分に適用される絶対誤差許容範囲。AbsTol(i) は、i 番目の解の要素の値が、スレッシュホールド以下の場合は、それほど重要ではありません。絶対的許容誤差値は、解が 0 に近付いたときの精度を決定します。要素 y(i) が、小さくて、興味がない場合でさえ、AbsTol(i) を y(i) の中で正しい桁数を得るように十分に小さくする必要があります。そして、ユーザは、より興味のある要素を十分な精度で計算することができます。 AbsTol が、ベクトルの場合、AbsTol の長さは、解ベクトル y の長さと等しくなる必要があります。AbsTol が、スカラの場合、y のすべての要素にその値が適用されます。

ODE 誤差コントロールプロパティ（続き）

プロパティ	値	詳細
NormControl	on | {off}	解のノルムに関する誤差のコントロール。このプロパティを on に設定すると、ソルバは、各シミュレーションステップで、norm(e) max(RelTol*norm(y),AbsTol) になるように誤差をコントロールすることを要求します。デフォルトでは、ソルバは、より厳密に、要素単位でのエラーのコントロールを使います。

ソルバの出力プロパティ

つぎのテーブルは、ソルバが生成する出力をコントロールするプロパティをまとめたものです。これらのプロパティを設定するには、odeset を使います。

ODE ソルバの出力プロパティ

プロパティ	値	詳細

OutputFcn	関数 {odeplot}	設定する出力関数。ソルバは、連続的な積分ステップの後で、この関数をコールします。
		
		たとえば、
		
		`options = odeset('OutputFcn',@myfun)`
		
		は、OutputFcn プロパティを出力関数 myfun に設定し、ODE ソルバに渡すことができます。
		
		出力関数は、つぎの型をしている必要があります。
		
		`status = myfun(t,y,flag)`
		
		ソルバは、つぎのフラグと共に指定した出力関数をコールします。コールする書式は、フラグによって変更することに注意してください。関数は、適切に対応します。
		
		init　シミュレーションを始める前に、出力関数を初期化するために、ソルバは、myfun(tspan,y0,'init') を呼び出します。tspan と y0 は、ODE ソルバへの入力引数です。

ODE ソルバの出力プロパティ（続き）

プロパティ	値	詳細
		{none} myfun が意図した機能を実行するように、各ステップの後で、ソルバは、status = myfun(t,y) をコールします。t は、区間 [a,b] の間の単点で、y は、その点での数値解です。myfun は、0 か、1 のいずれかからなる status を出力します。status = 1 の場合、ソルバはシミュレーションを停止します。ユーザは、この機能を使って、**Stop** ボタンを実行する仕組みを作ることができます。 done 積分が終了して、出力関数がクリーンアップ作業を実行することが可能となるときに、ソルバは myfun([],[],'done') を呼び出します。
		ユーザは、一般的な目的出力関数を使用でき、または、それをエデットして、ユーザ自身のものを作成できます。 • odeplot - 時系列プロット (出力引数なしでコールしたり、出力関数を設定しないでコールするときのデフォルト) • odephas2 - 2 次元の位相面プロット • odephas3 - 3 次元の位相面プロット • odeprint - 計算しながら、解を印刷 **注意** 出力引数なしで、ソルバをコールすると、ソルバは、解すべての履歴を保持するためのストレージを確保しません。
OutputSel	インデックスベクトル	プロパティ OutputSel は、解ベクトルのどの要素を出力関数に受け渡すかを特定したインデックスベクトルです。たとえば、出力関数 odeplot を使って、1 番目と 3 番目の要素のみをプロットしたいときに、これを使うことができます。 options = odeset('OutputFcn',@odeplot,'OutputSel',[1 3]); デフォルトでは、ソルバは、解の要素すべてを出力関数に渡します。

ODE ソルバの出力プロパティ（続き）

プロパティ	値	詳細
Refine	正の整数	出力点数を、Refine 倍します。Refine が 1 の場合、ソルバは、各時間ステップの最後でのみ解を出力します。整数 n >1 で与えられるプロパティ Refine は、出力点の数を n 倍に増やして、滑らかな出力を作ります。Refine は、length(tspan)>2 の場合、適用されません。 **注意** すべてのソルバで、Refine のデフォルト値は 1 です。ode45 では、デフォルトはソルバのステップサイズを大きくするために 4 とします。これを変更して ode45 で選択される時間ステップのみを見るには、Refine を 1 に設定してください。 Refine 用に作成されるエクストラの値は、連続に拡張した方程式を使って計算されます。これらは、計算時間を極端に増やさないで、計算される時間ステップ間で正確な解を得るために、ODE ソルバで使われる方程式を設定します。
Stats	on | {off}	Stats プロパティで積分計算に関する統計量などの情報について表示するかどうかを設定します。デフォルトでは、Stats は off です。これが、on の場合、計算終了後、つぎのような項目が表示されます。 • 成功したステップ回数 • 失敗した回数 • $f(t, y)$ を計算するためにコールされた ODE 関数の回数 • 偏微分行列 $\partial f/\partial y$ が作成された回数 • LU 分解の回数 • 線形システムの解の数

Jacobian 行列プロパティ

スティッフな ODE ソルバは、ユーザが Jacobian 行列、$\partial f/\partial y$ すなわち微分方程式を定義する関数の偏微分行列についての情報を与えると、実行が速くなります。

$$\frac{\partial f}{\partial y} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \cdots \\ \vdots & \vdots & \end{bmatrix}$$

Jacobian 行列$\partial f/\partial y$は、信頼性と効率性に敏感なので、スティッフなソルバ (ode15s, ode23s, ode23t, ode23tb) にのみ、Jacobian 行列プロパティを使います。Jacobian を計算する関数を与えない場合、これらのスティッフなソルバは、有限差分を使って、数値的に Jacobian を近似します。この場合、Vectorized、または、JPattern プロパティを使うこともできます。

つぎのテーブルは、Jacobian 行列プロパティをまとめたもので、詳細も記述しています。これらのプロパティを設定するには、odeset を使ってください。

ODEJacobian 行列プロパティ

プロパティ	値	詳細

Jacobian	関数 / 定数行列	スティッフな問題に対して、解の算出スピードや信頼性の改善のため、Jacobian 関数、または、定数行列を使用します。このプロパティを関数 FJac に設定します。ここで、FJac(t,y) は、$\partial f/\partial y$ を計算するか、$\partial f/\partial y$ の定数を計算します。
		
		上に示したスティッフな van der Pol 問題に対して、Jacobian は、つぎのようにコード化されます。
		
		`function J = vdp1000jac(t,y)`
		`J = [ 0 1`
		`(-2000*y(1)*y(2)-1) (1000*(1-y(1)^2)) ];`

JPattern	{0,1} のスパース行列	Jacobian の中で、ゼロでない要素に 1 をもつスパース行列。これは、数値的なスパースな Jacobian 行列を作成するために使われます。 $f(t, y)$ の成分 i が y の成分 j に依存している場合、$S(i, j) = 1$ で、他の場合、0 であるスパース行列 S にこのプロパティを設定します。ソルバは、このプロパティを使って、数値的に、スパース Jacobian 行列を作成します。Jacobian 行列が大きくて、スパースの場合、実行速度が著しく向上します。Jacobian プロパティを使った例題として、15-36 ページの " 例題 : 大規模、スティッフなスパース行列 " (brussode) を参照してください。

ODEJacobian 行列プロパティ（続き）

プロパティ	値	詳細

Vectorized	on | {off}	onに設定すると、F(t,[y1 y2 ...])が、ベクトル[F(t,y1) F(t,y2) ...] を出力するように、ODE 関数 F がコード化されているか否かをスティッフソルバに知らせます。これは、Jacobian 行列のすべての列を計算するのに必要とされる関数計算の回数を減らします。そして、解を計算する時間を実質的に減らすことになります。
		
		MATLAB 配列記法を使って、ODE 関数をベクトル化することは、簡単です。たとえば、スティッフな van der Pol の例題は、サブスクリプトにコロン記号を使い、配列パワー (.^) と配列乗算 (.*) を使って、ベクトル化されます。
		
		`function dydt = vdp1000(t,y)`
		`dydt = [y(2,:); 1000*(1-y(1,:).^2).*y(2,:)-y(1,:)];`

ステップサイズプロパティ

ステップサイズプロパティは、ソルバにより使われる最初のステップサイズを設定し、それを使って問題のスケールを良く認識するのに役に立ちます。それに加え、続いて行なう処理の大きさに範囲を設定できます。

つぎのテーブルは、サテップサイズプロパティをまとめたものです。詳細な説明は、テーブルに従って、その設定については、odeset を使用してください。

ODE ステップサイズプロパティ

プロパティ	値	詳細
InitialStep	正のスカラ	初期ステップサイズ。InitialStep は、ソルバが最初に計算するステップサイズに上限を設定できます。InitialStep を設定していない場合、初期のステップサイズは、初期時間 tspan(1) での解の勾配をベースにし、すべての解の要素の勾配がゼロの場合、非常に大き過ぎる値がステップサイズとして利用されることがあります。このような現象が起こることがわかっているときや積分計算を始めるときの重要な振舞いを確実に解きたい場合に適切な InitialStep を設定してください。
MaxStep	正のスカラ値 {0.1*abs(t0-tf)}	このプロパティは、ソルバが使うステップサイズの大きさに上限を設定します。微分方程式が周期性をもつ係数か、または、解をもっている場合、MaxStep を周期の数分の 1(例えば 1/4 倍) に設定することが良い場合があります。これは、ソルバが時間ステップを大きく引き伸ばしたりしなく、対象の区間全体をカバーすることを保証します。MaxStep は小さくしないでください。 • MaxStep を小さくして、出力点数を多くしないでください。これは、解の算出に時間がかかります。代わりに、Refine を使って、非常に低いコスト (計算時間をあまり使わない) で連続的に拡張することにより付加的な出力を計算します。 • 数値解の精度が十分でないように思えても、MaxStep を小さくしないでください。代わりに、相対誤差許容範囲 RelTol を減らし、絶対誤差許容ベクトル AbsTol に対して適切な値を決定するようにして計算される解を使います (誤差の許容範囲に関するプロパティの記述は、15-16 ページの " 誤差のコントロールプロパティ " を参照)。

ODE ステップサイズプロパティ（続き）

プロパティ	値	詳細
		• シミュレーション時間全体を通して1回だけしか起こらない現象を計算ステップが飛び越えてしまわないように、MaxStep を小さく設定するべきではありません。その変化が起こる時間が分かっている場合、2 つにシミュレーション時間を分けて、ソルバを 2 回起動してください。その時間が分からないときは、許容誤差値 RelTol と AbsTol を小さくして計算させてください。MaxStep を小さくするのは、最終手段として使ってください。

質量行列と DAE プロパティ

ODE suite のソルバは、つぎの型をしています。

$$M(t,y)\,y' = f(t,y) \tag{15-2}$$

ここで、質量行列 $M(t,y)$ はスパースです。

$M(t,y)$ が正則な場合、上の方程式は、$y' = M^{-1}f(t,y)$ と等価になり、ODE は、t_0 で任意の初期値 y_0 に対する解をもちます。より一般的な型 (式 15-2) は、質量行列の項でモデルを一般的に表現する場合、便利です。大規模な、スパース $M(t,y)$ に対して、式 15-2 を直接解くことは、ストレージの節約や、必要な実行時間の節約になります。

$M(t,y)$ が正則でない場合、$M(t,y)\,y' = f(t,y)$ は、微分代数方程式 (DAE) になります。DAE は、y_0 に矛盾がない場合、すなわち、$M(t_0,y_0)yp_0 = f(t_0,y_0)$ のような初期勾配 yp_0 が存在する場合のみ解をもちます。y_0 と yp_0 が矛盾する場合、ソルバは、それらを推定として使い、推定に近い矛盾のない値を計算し、問題を解くことを続けます。インデックス 1 の DAEs に対して、矛盾のない初期条件をもつ初期値問題を解くことは、ODE を解くことと同じです。

関数 ode15s と ode23t は、インデックス 1 の DAEs を解くことができます。DAE 問題の例として、hb1dae(15-45 ページの " 例題： 微分代数方程式 ") と amp1dae を参照してください。

つぎのテーブルは、質量行列と DAE プロパティをまとめたものです。詳細な説明は、テーブルに従ってください。これらのプロパティを設定するには、odeset を使います。

ODE 質量行列と DAE プロパティ

プロパティ	値	詳細
Mass	定数行列 | 関数	質量行列 $M(t,y)$ を計算する関数、または、定数質量行列です。問題 $My' = f(t,y)$ に対して、このプロパティを M の値に設定します。問題 $M(t,y)\,y' = f(t,y)$ に対して、このプロパティを関数 Mfun へのこのプロパティに設定します。ここで、Mfun(t,y) は、質量行列 $M(t,y)$ を計算する関数です。DAEs を解く場合、M が対角行列 (半陽的な DAE) であるように問題を定式化して利用します。関数 ode23s は、定数質量行列 M をもつ問題のみを解くことができます。 例題の問題は、fem1ode(15-33 ページの " 例題 : 有限要素離散化 "),fem2ode, batonode を参照してください。
MStateDependence	none | {weak} | strong	質量行列の y への依存性。問題 $M(t)\,y' = f(t,y)$ に対して、このプロパティに none を設定します。weak と strong は共に、$M(t,y)$ を示しますが、weak は、代数方程式を解く場合、近時を使って、インプリシットに解きます。
MvPattern	スパース行列	$\partial(M(t,y)v)/\partial y$ スパース行列。任意の k に対して、$M(t,y)$ の成分 (i,k) が、y の成分 j に依存している場合、$S(i,j) = 1$ であるスパース行列 S で、そうでない場合は 0 となります。MStateDependence が、strong の場合、ソルバ ode15s, ode23t, ode23tb を使います。例題としては burgersode を参照してください。

ODE 質量行列と DAE プロパティ（続き）

プロパティ	値	詳細
MassSingular	yes | no | {maybe}	質量行列が特異であるかどうかを示します。このプロパティは、質量行列が特異でなく、ode15s あるいは ode23t ソルバを利用している場合には no に設定されます。maybe のデフォルト値により、ソルバは問題が DAE であるかどうかを調べます。すなわち、$M(t_0, y_0)$ が特異であるかどうかです。 質量行列に関する問題の例題は、15-33 ページの " 例題 : 有限要素離散化 " (fem1ode) を参照してください。
InitialSlope	ベクトル | { ゼロ ベクトル }	矛盾のない初期勾配yp_0を表すベクトル。ここで、yp_0 は、$M(t_0, y_0)\, yp_0 = f(t_0, y_0)$ を満足します。デフォルトは、ゼロベクトルです。 このプロパティは、DAEs を解く場合、関数 ode15s と ode23t と共に使います。

イベントの発生個所に関するプロパティ

ODE 問題の中で、ボールが地面に達する時刻や宇宙船が地球に帰ってくる時刻、ODE の解がある値に達する時刻等のような特別のイベントの時刻が重要になることがあります。問題を解いている間、MATLAB の ODE ソルバはユーザが定義した関数のベクトルがどこからどこまでゼロになるのか、その移り変わりの場所を求めます。

つぎのテーブルは、Events プロパティに関する説明です。このプロパティを設定するには、odeset を使ってください。

ODE イベントプロパティ

文字列	値	詳細

Events	関数	MATLAB は、一つ、または、複数のイベント関数を含んでいます。MATLAB は、つぎの型で使います。
		`[value,isterminal,direction] = events(t,y)`
		value, isterminal, direction は、ベクトルで、その i 番目の要素は、i 番目のイベント関数に対応します。
		• value(i) は、i 番目のイベント関数の値です。
		• sterminal(i) = 1 は、積分がイベント関数のゼロ点で終了する場合 1 で、その他の場合は 0 です。
		• すべてのゼロが計算で求まる場合、direction(i) = 0(デフォルト)、イベント関数が増加する部分だけで求まる場合 +1、減少する部分でだけ求まる場合 -1 です。
		イベント関数を指定した場合、ソルバは、3 つの付加的な出力、TE、YE、IE を戻します。
		`options = odeset('Events',@events)`
		`[T,Y,TE,YE,IE] = solver(odefun,tspan,y0,options)`
		• TE は、各イベントが起こった時刻の列ベクトル
		• YE の行は、TE の中の時刻に対応する解
		• インデックスベクトル IE は、TE で示された時刻でどのイベントが起こったかを特定します。
		イベント関数を使った例題については、15-39 ページの " 例題 : 単純イベント位置検出問題 " (ballode) と 15-42 ページの " 例題 : アドバンスなイベント位置検出問題 " (orbitode) を参照してください。

ode15s のプロパティ

ode15s は、可変次数のスティッフソルバで数値微分公式 (NDFs) に基づく手法です。NDFs は、Gear 法としても知られている後退微分公式 (BDFs) と非常に良く似

た手法ですが、より効率的なものです。ode15s のプロパティでは、NDFs か BDFs の選択と公式の最大次数の設定が行えます。

つぎのテーブルは、ode15s プロパティを記述しています。これらのプロパティを設定するには、odeset を使ってください。

ode15s プロパティ

プロパティ	値	詳細
MaxOrder	1 | 2 | 3 | 4 | {5}	解を計算するために使用する最大次数
BDF	on | {off}	デフォルトの NDFs の代わりに BDFs を使用する場合に設定。BDF を on に設定することは、ode15s に、BDF を使わせることになります。 NDFs も BDFs も 1 または 2 次では、A- 安定です。(安定領域は、複素平面の左半面全てを含みます) これより高い次数は、同様の安定性をもたず、次数が高いほど不安定さが増します。もっとも安定な公式が使われるようにするために、MaxOrder の値を小さくし (例えば 2 にする)、より効果的に解くことのできるステイッフな問題 (スティッフなときの解の振動) のクラスがあります。

例題 :ODE 初期値問題ソルバの適用

この節では、MATLAB の中で解くことのできる種々の種類の例題を示しています。

- 単純なノンスティッフな問題 (rigidode)
- スティッフな問題 (vdpode)
- 有限要素離散化 (fem1ode)
- 大規模、スティッフなスパース問題 (brussode)
- 単純イベント位置検出問題 (ballode)
- アドバンスなイベント位置検出問題 (orbitode)
- 微分代数問題 (hb1dae)

例題： 単純なノンスティッフ問題

rigidode は、ノンスティッフソルバに対して、Krogh [6] により提唱された標準テスト問題の解を示します。

ODEs は、外力のない剛体の Euler 方程式です。

$$y_1' = y_2 y_3$$

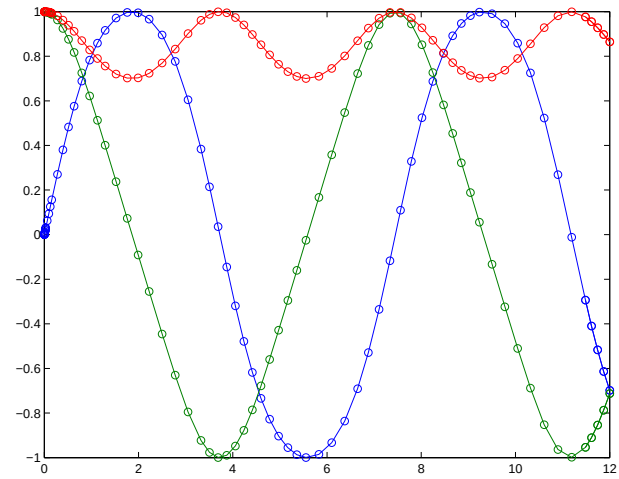
$$y_2' = -y_1 y_3$$

$$y_3' = -0.51 y_1 y_2$$

ユーザの便利さのため、すべての問題は、一つの M ファイルで定義され、解かれます。微分方程式は、サブ関数 f としてコード化されています。ode45 ソルバは、出力引数なしでコールすると、デフォルトの出力関数 odeplot が使われて、解成分をプロットします。この例題を実行するには、コマンドラインで、単に rigidode をタイプしてください。

この例題を MATLAB Help ブラウザから実行するには、デモ名をクリックしてください。他の方法としては、コマンドラインで、rigidode と入力してください。

```
function rigidode
%RIGIDODE 外力のない剛体の Euler 方程式
tspan = [0 12];
y0 = [0; 1; 1];

% ode45 を使って、問題を解く
ode45(@f,tspan,y0);
% --------------------------------------------------------------
function dydt = f(t,y)
dydt = [ y(2)*y(3)
        -y(1)*y(3)
        -0.51*y(1)*y(2) ];
```



例題： スティッフな問題 (van der Pol 方程式)

vdpode は、15-13 ページの " 例題： van der Pol 方程式 , m = 1000(スティッフ)" の中で記述されている van der Pol 方程式を示します。微分方程式

$$y_1' = y_2$$

$$y_2' = \mu(1 - y_1^2)y_2 - y_1$$

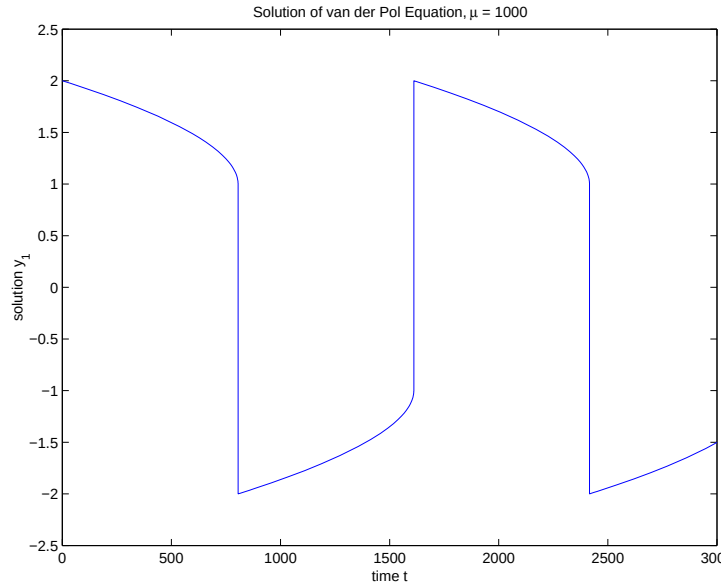
は、定数パラメータ μ を含んでいます。

この問題は、μ が増加すると、よりスティッフになり、振動の周期が長くなります。μ が 1000 を超える場合、方程式は、過減衰になり、問題は強いスティッフ性を示します。リミットサイクルは、解の要素がゆっくり変化する部分と問題が非常にスティッフな部分を含んでいます。すなわち、スティッフでない部分と急激に変化する部分が交互に生じます。

デフォルトでは、スティッフな問題に対して使用する ODE suite の中のソルバは、Jacobian行列を数値的に近似します。しかし、ここの例題では、サブ関数J(t,y,mu) を与えて、μ = mu に対して、(t,y) で、Jacobian 行列 $\partial f/\partial y$ を解析的に計算します。解析的な Jacobian の使用は、積分の信頼性と効率を向上します。

MATLAB Help ブラウザから、この例題を実行するには、デモ名をクリックする、コマンドラインで、vdpode と入力してください。コマンドラインで、vdpode への引数として、μ の値を指定することができます。デフォルトは、μ = 1000 です。

```
function vdpode(MU)
%VDPODE パラメータ化した van der Pol 方程式 ( 大きな MU に対して、スティッフ )
if nargin < 1
  MU = 1000;     % デフォルト
end

tspan = [0; max(20,3*MU)];              % 数周期
y0 = [2; 0];
options = odeset('Jacobian',@J);

[t,y] = ode15s(@f,tspan,y0,options,MU);

plot(t,y(:,1));
title(['Solution of van der Pol Equation, \mu = ' num2str(MU)]);
xlabel('time t');
ylabel('solution y_1');

axis([tspan(1) tspan(end) -2.5 2.5]);
------------------------------------------------------------
function dydt = f(t,y,mu)
dydt = [            y(2)
         mu*(1-y(1)^2)*y(2)-y(1) ];
------------------------------------------------------------
function dfdy = J(t,y,mu)
dfdy = [         0                  1
         -2*mu*y(1)*y(2)-1    mu*(1-y(1)^2) ];
```



例題： 有限要素離散化

fem1ode は、偏微分方程式の有限要素離散からの結果の ODE 問題の解を示します。fem1ode(N) の中の N の値は、離散化をコントロールするもので、求まる結果の方程式は、N 個の方程式から構成されます。デフォルトは、N = 19 です。

この例題は、質量行列を含んでいます。ODEs のシステムは、偏微分方程式のライン解法から求まります。

$$e^{-t}\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

初期条件 $u(0,x) = \sin(x)$ と境界条件 $u(t,0) = u(t,\pi) = 0$ を使います。整数 N が選択され、h は $\pi/(N+1)$ と定義されます。そして、偏微分方程式の解は、$k = 0, 1, \ldots, N+1$ に対して、$x_k = kh$ で、次の式

$$u(t,x_k) \approx \sum_{k=1}^{N} c_k(t)\phi_k(x)$$

によって、近似されます。ここで、$\phi_k(x)$ は x_k で 1、それ以外の x_j で 0 である区分的線形関数です。Galerkin 離散化は、つぎの ODE システムを導きます。

$$M(t)c' = Jc \text{ where } c(t) = \begin{bmatrix} c_1(t) \\ \\ c_N(t) \end{bmatrix}$$

三角行列 $M(t)$ と J は、つぎの関係があります。

$$M_{ij} = \begin{cases} 2h/3 \exp(-t) & \text{if } i = j \\ h/6 \exp(-t) & \text{if } i = j \pm 1 \\ 0 & \text{otherwise} \end{cases}$$

と

$$J_{ij} = \begin{cases} -2/h & \text{if } i = j \\ 1/h & \text{if } i = j \pm 1 \\ 0 & \text{otherwise} \end{cases}$$

初期値 $c(0)$ は、偏微分方程式に対して、初期条件から求まります。問題は、時間間隔 $[0, \pi]$ で解かれます。

femlode の中で、プロパティ

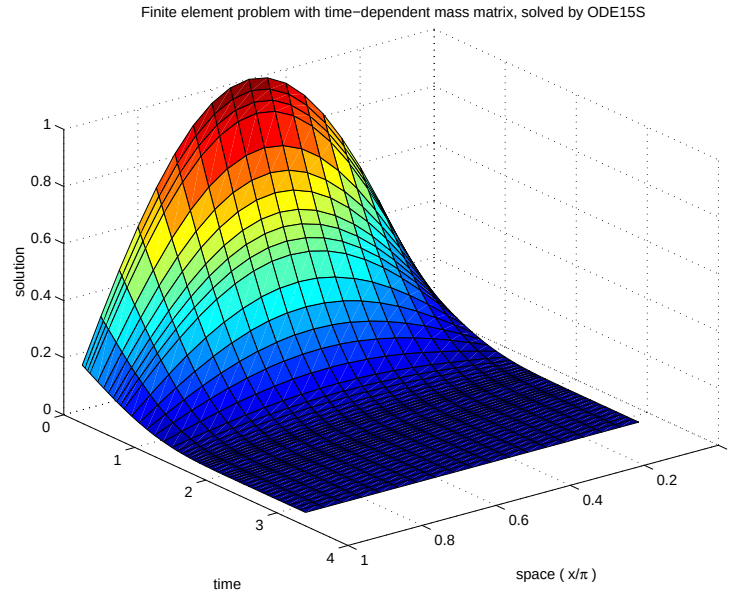
```
options = odeset('Mass',@mass,'MStateDep','none','Jacobian',J)
```

は、問題が $M(t)y' = Jy$ の型をしていることを示しています。サブ関数 A(t,N) は、時変質量行列 $M(t)$ を計算し、J は定数の Jacobian になります。

MATLAB Help ブラウザからこの例題を実行するには、デモ名をクリックするか、コマンドラインで、femlode と入力してください。コマンドラインで、femlode の引数として、N の値を設定することができます。デフォルトは、$N = 19$ です。

```
function femlode(N)
%FEMlODE  時変質量行列をもつスティッフな問題

if nargin < 1
```

```
  N = 19;
end
h = pi/(N+1);
y0 = sin(h*(1:N)');
tspan = [0; pi];

% ヤコビアンは定数
e = repmat(1/h,N,1);    %  e=[(1/h) ... (1/h)];
d = repmat(-2/h,N,1);   %  d=[(-2/h) ... (-2/h)];
J = spdiags([e d e], -1:1, N, N);

options = odeset('Mass',@mass,'MStateDependence','none', ...
                 'Jacobian',J);

[t,y] = ode15s(@f,tspan,y0,options,N);

surf((1:N)/(N+1),t,y);
set(gca,'ZLim',[0 1]);
view(142.5,30);
title(['Finite element problem with time-dependent mass ' ...
       'matrix, solved by ODE15S']);
xlabel('space ( x/\pi )');
ylabel('time');
zlabel('solution');
%-----------------------------------------------------------------
function out = f(t,y,N)
h = pi/(N+1);
e = repmat(1/h,N,1);    %  e=[(1/h) ... (1/h)];
d = repmat(-2/h,N,1);   %  d=[(-2/h) ... (-2/h)];
J = spdiags([e d e], -1:1, N, N);
out = J*y;
%-----------------------------------------------------------------
function M = mass(t,N)
h = pi/(N+1);
e = repmat(exp(-t)*h/6,N,1);  % e(i)=exp(-t)*h/6
e4 = repmat(4*exp(-t)*h/6,N,1);
M = spdiags([e e4 e], -1:1, N, N);
```



例題： 大規模、スティッフなスパース行列

brussode は、(ポテンシャル的に) 大規模なスティッフスパース問題の解を示します。問題は、古典的な"Brusselator"システム[2]で、化学反応の拡散モデルです。

$$u_i' = 1 + u_i^2 v_i - 4u_i + \alpha(N+1)^2(u_{i-1} - 2u_i + u_{i+1})$$
$$v_i' = 3u_i - u_i^2 v_i + \alpha(N+1)^2(v_{i-1} - 2v_i + v_{i+1})$$

は、時間間隔 $[0,10]$ で、$\alpha = 1/50$ とで

$$\left.\begin{aligned} u_i(0) &= 1 + \sin(2\pi x_i) \\ v_i(0) &= 3 \end{aligned}\right\} \quad \text{with } x_i = i/(N+1), \text{ for } i = 1, \ldots, N$$

と共に解かれます。システム内には、$2N$ 個の方程式が存在し、方程式が、$u_1, v_1, u_2, v_2, \ldots$ であるように順序付けられている場合、Jacobian は、制約幅 5 をもつ帯域になります。

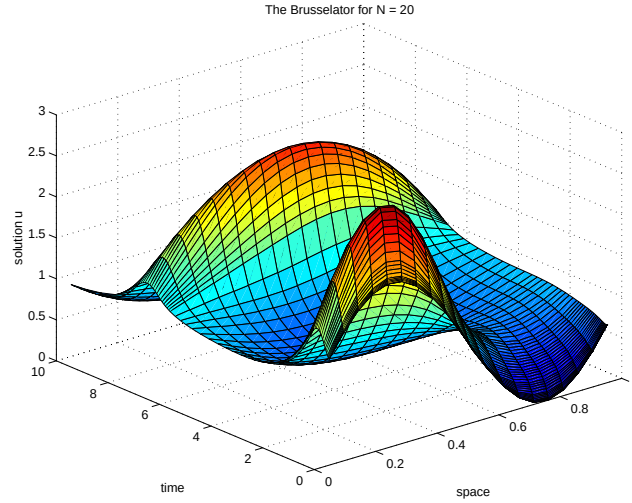
brussode(N) をコール中、パラメータ N ≥ 2 がグリッド点の数を設定するのに使います。ここで、N は*N* に対応します。結果のシステムは、2N 個の方程式から構成されます。デフォルトでは、N は 20 です。問題は、N が大きくなると、スティッフ性が強くなり、Jacobian のスパース性が増大することです。

サブ関数 F(t,y,N) は、Brusselator 問題用の微係数ベクトルを出力します。サブ関数 jpattern(N) は、Jacobian $\partial f/\partial y$ の中の非ゼロの位置を示す 1 と 0 のスパース行列を出力します。デフォルトでは、スティッフな ODE ソルバは、フル行列として、数値的な Jacobian を生成します。プロパティ JPattern は、ソルバが Jacobian をスパース行列として、数値的に生成するために使われるスパースパターンをもったソルバを与えるために使われます。スパースパターンを与えると、Jacobian を作成するために必要な関数計算の回数が著しく減り、シミュレーションが速くなります。Brusselator 問題に対して、スパース性が与えられなければ、関数の 2N 回の計算が、2N 行 2N 列の Jacobian 行列を計算するために必要となります。スパース性が、与えられた場合、N の値に関わらず、四つの計算のみが必要です。

MATLAB Help ブラウザからこの例題を実行するには、デモ名をクリックするか、コマンドラインで、brussode と入力してください。コマンドラインで、brussode への引数として、*N* の値を設定することができます。デフォルトは、*N* = 20 です。

```
function brussode(N)
%BRUSSODE 化学反応をモデル化したスティッフな問題

if nargin<1
  N = 20;
end

tspan = [0; 10];
y0 = [1+sin((2*pi/(N+1))*(1:N)); repmat(3,1,N)];

options = odeset('Vectorized','on','JPattern',jpattern(N));

[t,y] = ode15s(@f,tspan,y0,options,N);

u = y(:,1:2:end);
x = (1:N)/(N+1);
surf(x,t,u);
view(-40,30);
xlabel('space');
ylabel('time');
```

```
zlabel('solution u');
title(['The Brusselator for N = ' num2str(N)]);
% -----------------------------------------------------------------
function dydt = f(t,y,N)
c = 0.02 * (N+1)^2;
dydt = zeros(2*N,size(y,2));       % dy/dt を前もって設定
% グリッドの片端で、関数の 2 成分を計算
i = 1;
dydt(i,:) = 1 + y(i+1,:).*y(i,:).^2 - 4*y(i,:) + ...
            c*(1-2*y(i,:)+y(i+2,:));
dydt(i+1,:) = 3*y(i,:) - y(i+1,:).*y(i,:).^2 + ...
              c*(3-2*y(i+1,:)+y(i+3,:));
% すべての内部グリッド点の関数の 2 成分を計算
i = 3:2:2*N-3;
dydt(i,:) = 1 + y(i+1,:).*y(i,:).^2 - 4*y(i,:) + ...
            c*(y(i-2,:)-2*y(i,:)+y(i+2,:));
dydt(i+1,:) = 3*y(i,:) - y(i+1,:).*y(i,:).^2 + ...
              c*(y(i-1,:)-2*y(i+1,:)+y(i+3,:));
% グリッドの他の片端で、関数の 2 成分を計算
i = 2*N-1;
dydt(i,:) = 1 + y(i+1,:).*y(i,:).^2 - 4*y(i,:) + ...
            c*(y(i-2,:)-2*y(i,:)+1);
dydt(i+1,:) = 3*y(i,:) - y(i+1,:).*y(i,:).^2 + ...
              c*(y(i-1,:)-2*y(i+1,:)+3);
% -----------------------------------------------------------------
function S = jpattern(N)
B = ones(2*N,5);
B(2:2:2*N,2) = zeros(N,1);
B(1:2:2*N-1,4) = zeros(N,1);
S = spdiags(B,-2:2,2*N,2*N);
```



例題： 単純イベント位置検出問題

ballode は、バウンドするボールの運動をモデル化します。この例題は、ODE ソルバがイベントの生じる位置を検出する機能を示すものです。

バウンドするボールの方程式は、つぎのように表せます。

$$y_1' = y_2$$
$$y_2' = -9.8$$

この例題で、イベント関数は、サブ関数 events でコード化されています。

```
[value,isterminal,direction] = events(t,y)
```

ここで、つぎの値を出力します。

- イベント関数の値
- value = 0 の場合、シミュレーションを停止すべきか否かの情報 (isterminal = 1, または ,0)
- ゼロ交点の希望する方向

-1 負の方向のみのゼロ交点を検出

0 すべてのゼロ交点の検出

1 正の方向のみのゼロ交点を検出

value, isterminal, direction の長さは、event 関数の数と同じになります。各ベクトルの i 番目の要素は、i 番目の関数に対応します。より進んだイベントの発生位置を検出する例題は、orbitode(15-42 ページの " 例題 : アドバンスなイベント位置検出問題 ") を参照してください。

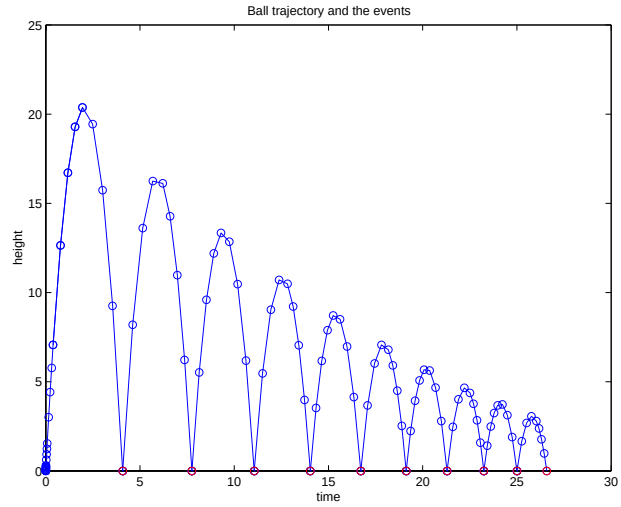
ballode の中で、Events プロパティを @events に設定することは、ボールが落ちる途中 (direction = -1) で、ボールが地面にヒットした (高さ y(1) は 0 になる) とき、シミュレーションを停止 (isterminal = 1) することになります。それで、例題では、バウンドするボールに対応する初期条件を付けて、積分を再スタートします。

この例題を、MATLAB Help ブラウザから実行するには、デモ名をクリックするか、コマンドラインで、ballode と入力してください。

```
function ballode
%BALLODE バウンドするボールのデモを実行

tstart = 0;
tfinal = 30;
y0 = [0; 20];
refine = 4;
options = odeset('Events',@events,'OutputFcn', @odeplot,...
                 'OutputSel',1,'Refine',refine);

set(gca,'xlim',[0 30],'ylim',[0 25]);
box on
hold on;

tout = tstart;
yout = y0.';
teout = [];
yeout = [];
ieout = [];
for i = 1:10
  % 最初のターミナルイベントまで解析
```

```
  [t,y,te,ye,ie] = ode23(@f,[tstart tfinal],y0,options);
  if ~ishold
    hold on
  end
  % 出力をまとめる
  nt = length(t);
  tout = [tout; t(2:nt)];
  yout = [yout; y(2:nt,:)];
  teout = [teout; te];    % Etstart でのイベントをレポートしない
  yeout = [yeout; ye];
  ieout = [ieout; ie];

  ud = get(gcf,'UserData');
  if ud.stop
    break;
  end

  % 9 の減衰をもつ新しい初期条件を設定
  y0(1) = 0;
  y0(2) = -.9*y(nt,2);

  % 最初の timestep の良い推定は、最後の timestep の長さになり、それを
  % 使うことで、計算は速くなります。
  options = odeset(options,'InitialStep',t(nt)-t(nt-refine),...
                           'MaxStep',t(nt)-t(1));
  tstart = t(nt);
end

plot(teout,yeout(:,1),'ro')
xlabel('time');
ylabel('height');
title('Ball trajectory and the events');
hold off
odeplot([],[],'done');
% --------------------------------------------------------------
function dydt = f(t,y)
dydt = [y(2); -9.8];
% --------------------------------------------------------------
function [value,isterminal,direction] = events(t,y)
% 高さが、減じる方向でゼロを通る時刻を求め、シミュレーションを
```

```
% 停止する。
value = y(1);      % 高さ = 0 を検出
isterminal = 1;    % シミュレーションを停止
direction = -1;    % 負の方向のみ
```



例題： アドバンスなイベント位置検出問題

orbitode は、ノンスティッフな問題に対して使用するソルバに対する標準テストの解を示します。これは、月の周りを回り、地球に戻ってくる宇宙船の軌跡を示すものです (Shampine and Gordon [6], p.246)。

orbitode 問題は、4 つの方程式を解きます。

$$y_1' = y_3$$

$$y_2' = y_4$$

$$y_3' = 2y_4 + y_1 - \frac{\mu^*(y_1+\mu)}{r_1^3} - \frac{\mu(y_1-\mu^*)}{r_2^3}$$

$$y_4' = -2y_3 + y_2 - \frac{\mu^* y_2}{r_1^3} - \frac{\mu y_2}{r_2^3}$$

ここで、つぎのようにします。

$$\mu = 1/82.45$$

$$\mu^* = 1-\mu$$

$$r_1 = \sqrt{(y_1+\mu)^2 + y_2^2}$$

$$r_2 = \sqrt{(y_1-\mu^*)^2 + y_2^2}$$

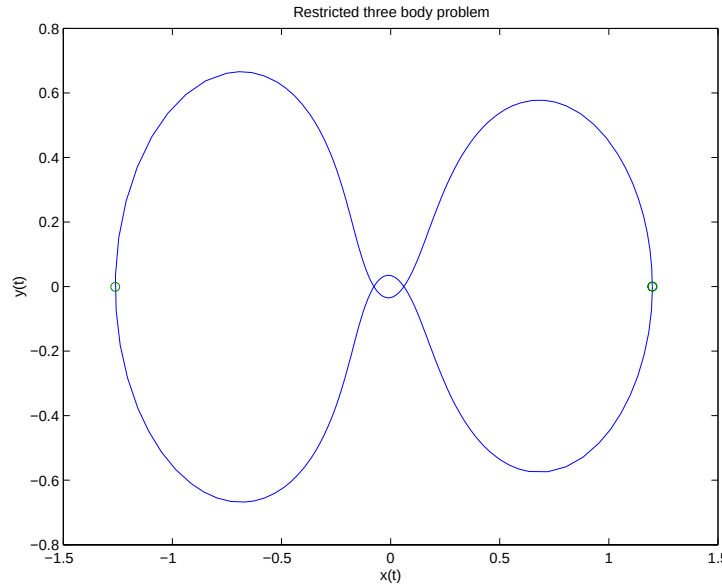
最初の 2 つの解要素は、無限小の質量の座標系で、他のものに対してプロットしたものが、他の 2 つの物体の周りを回る軌跡になります。初期条件は、軌跡が周期性をもつように選択されます。μ の値は、宇宙船が月の周りを回り、地球に戻るように設定します。中程度の厳密さをもったトレランスが、軌跡の質的に高い挙動を再生するために必要となります。適切な値は、RelTol が 1e-5 で、AbsTol が 1e-4 です。

events サブ関数は、スタート点から最大の距離をもつ点の位置を示すイベント関数と宇宙船が、スタート点に戻る時刻を含むものです。積分に使われるステップサイズが大きくて、イベントの位置を決定できない場合でさえ、イベントの正確な位置を割り出すことに注意してください。この例題で、ゼロクロッシングの方向を決定する機能は、クリティカルなものです。初期の点と最大距離の点として出力される点は共に、同じイベント関数値をもち、クロッシングの方向は、それらを区別するために使われます。

MATLAB Help ブラウザからこの例題を実行するには、デモ名をクリックするか、コマンドラインで、orbitode と入力してください。この例題は、出力関数 odephase2 を使って、2 次元の位相平面プロットを作成し、積分の進行状況を見ることができます。

```
function orbitode
%ORBITODE 制約付き 3 体問題

tspan = [0 7];
y0 = [1.2; 0; 0; -1.04935750983031990726];
options = odeset('RelTol',1e-5,'AbsTol',1e-4,...
                 'OutputFcn',@odephas2,'Events',@events);

[t,y,te,ye,ie] = ode45(@f,tspan,y0,options);

plot(y(:,1),y(:,2),ye(:,1),ye(:,2),'o');
title ('Restricted three body problem')
ylabel ('y(t)')
xlabel ('x(t)')
% ------------------------------------------------------------
function dydt = f(t,y)
mu = 1 / 82.45;
mustar = 1 - mu;
r13 = ((y(1) + mu)^2 + y(2)^2) ^ 1.5;
r23 = ((y(1) - mustar)^2 + y(2)^2) ^ 1.5;
dydt = [ y(3)
         y(4)
         2*y(4) + y(1) - mustar*((y(1)+mu)/r13) - ...
                         mu*((y(1)-mustar)/r23)
        -2*y(3) + y(2) - mustar*(y(2)/r13) - mu*(y(2)/r23) ];
% ------------------------------------------------------------
function [value,isterminal,direction] = events(t,y)
% Locate the time when the object returns closest to the
% initial point y0 and starts to move away, and stop integration.
% Also locate the time when the object is farthest from the
% initial point y0 and starts to move closer.
%
% The current distance of the body is
%
%   DSQ = (y(1)-y0(1))^2 + (y(2)-y0(2))^2
%       = <y(1:2)-y0,y(1:2)-y0>
%
% A local minimum of DSQ occurs when d/dt DSQ crosses zero
% heading in the positive direction.  We can compute d(DSQ)/dt as
%
```

```
%  d(DSQ)/dt = 2*(y(1:2)-y0)'*dy(1:2)/dt = 2*(y(1:2)-y0)'*y(3:4)
%
y0 = [1.2; 0];
dDSQdt = 2 * ((y(1:2)-y0)' * y(3:4));
value = [dDSQdt; dDSQdt];
isterminal = [1; 0];            % Stop at local minimum
direction = [1; -1];            % [local minimum, local maximum]
```



例題： 微分代数方程式

hb1dae は、hb1ode デモを*微分代数方程式* (DAE) 問題として作り替えたものです。hb1ode デモの中でコード化されている Robertson 問題は、スティッフな ODE 問題を解くコードに対する古典的なテスト問題です。

$$y_1' = -0.04y_1 + 10^4 y_2 y_3$$

$$y_2' = 0.04y_1 - 10^4 y_2 y_3 - 3 \cdot 10^7 y_2^2$$

$$y_3' = 3 \cdot 10^7 y_2^2$$

注意 Robertson 問題は、LSODI [3] へのプロローグへの例題です。

---

hb1ode で、問題は、$y_1(0) = 1$, $y_2(0) = 0$, $y_3(0) = 0$ をもち、定常状態で解かれます。これらの微分方程式は、線形保存則を満足し。DAE として問題を再定式化するのに使います。

$$y_1' = -0.04y_1 + 10^4 y_2 y_3$$

$$y_2' = 0.04y_1 - 10^4 y_2 y_3 - 3 \cdot 10^7 y_2^2$$

$$0 = y_1 + y_2 + y_3 - 1$$

明らかに、これらの方程式は、和が 1 にならない要素をもつ$y(0)$に対する解をもっていません。問題は、

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

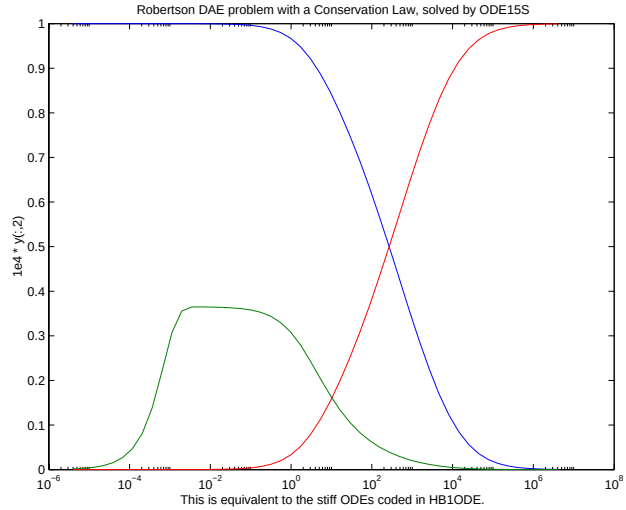
である $My' = f(t, y)$ をもちます。M は明らかに正則でなく、hb1dae は、ソルバに伝えません。ソルバは、問題が ODE ではなく、DAE であることを識別します。同様に、整合性をもつ初期条件は明らかになりますが、例題では、値 $y_3(0) = 10^{-3}$ を使い、整合性のある初期条件の計算を示しています。

MATLAB Help ブラウザからこの例題を実行するには、デモ名をクリックするか、コマンドラインに、hb1dae と入力してください。hb1dae は、つぎの特徴をもっています。

- 他の成分と比べ、y_2 により小さな絶対誤差のトレランスを課します。これは、y_2 が他の要素より小さく、大きな変化が、相対的に短い時間で生じるためです。

- y_2 の挙動をうまく示すために、付加的な点を加えます。
- y_2 に 10^4 乗算して、y_2 をプロットしたとき、見えるようにします。
- 対数スケールを使って、長時間でも解のプロットを見えるようにします。

```
function hb1dae
%HB1DAE スティッフな微分代数方程式 (DAE)

% 定数、特異質量行列
M = [1 0 0
     0 1 0
     0 0 0];

% 初期化をテストするため、矛盾のある初期条件を使用
y0 = [1; 0; 1e-3];
tspan = [0 4*logspace(-6,6)];

% LSODI 例題の許容範囲を使用。'MassSingular' プロパティ
% は、DAE の自動検出をテストするデフォルト 'maybe' のまま
% にします。
options = odeset('Mass',M,'RelTol',1e-4,...
                 'AbsTol',[1e-6 1e-10 1e-6],'Vectorized','on');

[t,y] = ode15s(@f,tspan,y0,options);

y(:,2) = 1e4*y(:,2);

semilogx(t,y);
ylabel('1e4 * y(:,2)');
title(['Robertson DAE problem with a Conservation Law, '...
       'solved by ODE15S']);
xlabel('This is equivalent to the stiff ODEs coded in HB1ODE.');
% --------------------------------------------------------------
function out = f(t,y)
out = [ -0.04*y(1,:) + 1e4*y(2,:).*y(3,:)
         0.04*y(1,:) - 1e4*y(2,:).*y(3,:) - 3e7*y(2,:).^2
         y(1,:) + y(2,:) + y(3,:) - 1 ];
```



質問と回答とトラブルシュート

この節では、MATLAB の ODE ソルバの使用法やオペレーションについての質問と回答のテーブルをいくつか用意しています。

- 一般的な ODE ソルバに関する質問
- 問題のサイズ、使用するメモリサイズ、計算時間
- 積分計算の時間ステップ
- 許容誤差とその他のパラメータ
- 異なるシステムを解く場合
- トラブルシューティング

一般的な ODE ソルバの問題

問題	回答
ODE ソルバは、quad または quadl とどのように異なりますか。	quad と quad8 は、$y' = f(t)$ の型の問題を解きます。ODE ソルバは、さらに一般的な、$y' = f(t,y)$ または、質量行列 $M(t,y)\,y' = f(t,y)$ を含む問題を解きます。
未知数の数より方程式が多いときやその逆のとき (未知数の数より方程式が少ない時)、ODE システムを解くことができますか。	できません。

プログラムのサイズ、使用するメモリサイズ計算時間

質問	回答
ODE suite でどのくらいの大きさの問題を解くことができますか。	基本的な制限は、メモリサイズとシミュレーション時間です。各ステップ毎に、ノンスティッフな問題に対するソルバは、長さ n のベクトルを割り当てます。ここで、n は、システムの方程式の数です。スティッフな問題に対するソルバは、長さ n のベクトルを割り当てるだけでなく、n 行 n 列のヤコビアン行列も割り当てます。これらのソルバでは、スパース行列を利用するのが有効です。 問題がスティッフでないか、スパースオプションを使った場合は、数千の未知数をもつ問題を解くことができる可能性があります。しかしながら、この場合も結果を維持するためのストレージが問題になります。ソルバに特別な点のみで解を計算させたり、または、出力引数を設定しないでソルバをコールしたり、解をモニタするために出力関数を使ったりすることを試みてください。

プログラムのサイズ、使用するメモリサイズ計算時間（続き）

質問	回答
非常に大きなシステムを解いている時でも、y の中のほんの 2、3 の要素のみしか必要でない場合があります。不要な要素をストアしない方法が何かありますか。?	あります。ユーザが設定する出力関数を作成することで、この問題に対応できます。ソルバの読み込みに出力引数を設定しなかった場合、ソルバは、全ての要素の履歴を保存するための領域を割り当てません。その代わりに、ソルバは、OutputFcn(t,y) を各時間ステップ毎にコールします。特定の要素の履歴を保持するためには、対象となる要素だけをストアする出力関数を記述し、それをプロットするように記述します。
積分計算の立ち上がり部分でどのくらいの CPU 時間がかかりますか。また、それはどのようにすれば軽減できますか。	立ち上がりで、最も時間のかかるのは、問題のスケールに合ったステップサイズを見つけようとすることに依るものです。もし、適切なステップサイズが分かっている場合、プロパティ InitialStep を使ってください。例えば、イベントの発生位置を含むループで積分器を繰り返し読み込むとき、イベントの前の最後のステップサイズは、おそらく、つぎの積分計算にとっても適切なスケールと考えられます。例題としては、ballode を参照してください。

積分計算の時間ステップ

質問	回答
積分が採用した初めのステップサイズが大きすぎて、重要な振る舞いを見失いませんか。	プロパティ InitialStep で最初のステップサイズは設定できます。積分器は、この値を、まず利用し、必要ならばさらに小さくします。
固定ステップは場で積分計算できますか。	いいえ。

許容誤差とその他のオプションパラメータ

質問	回答
どのようにして、RelTol と AbsTol を決めれば良いでしょうか。	RelTol は、相対的な精度を示す許容誤差値で解の正しい桁数をコントロールします。AbsTol は、絶対的な許容誤差値で真の解と計算された値との差をコントロールします。各時間ステップで、解の中の i 番目の要素のエラー e が次の条件を満たす必要があります。 `|e(i)| <= max(RelTol*abs(y(i)),AbsTol(i))` 大雑把に言って、このことは、スレッシュホールド AbsTol(i) より小さいものを除いて、すべての解成分で、RelTol の正しい桁をもとうとすることを意味しています。すなわち、ソルバは、AbsTol(i) よりも小さい解に対して、正しい桁数を保証します。もし、問題がスレッシュホールドをもつならば、それを AbsTol として使ってください。y(i) の値が、小さくて、興味がもてない場合でさえ、y(i) の中のある正しい桁を得るために、十分小さい AbsTol(i) に設定しなければなりません。そして、より興味のある要素を精度良く計算できます。
コンピュータの計算の精度と同等の精度の答えが欲しいのですが、単純に RelTol を eps にしてはいけないのでしょうか。	マシンの精度に近い値を得ることはできますが、その方法では近い値ということだけです。連続な関数を近似しようとするため、ソルバでは eps に近い値を RelTol として利用できません。許容誤差値を eps にすると、マシンの計算は、全ての関数が不連続であるとみなすことになります。
解のある 1 つの要素の答えの精度には興味が無いことを設定するには、どのようにすれば良いでしょうか。	その要素に対応する絶対許容誤差値を大きくすることができます。許容誤差値がその要素よりも大きいならば、その要素は、正しい桁数には設定されません。他の要素を正確に計算するために、この要素から正しい桁数を設定しなければなりませんが、一般的に、このような操作は自動的に行なわれません。

異なるシステムを解く場合

質問	回答
ライン手法で離散化して PDE を取り扱うことができますか。	できます。離散化された PDE は、ODE のシステムになります。離散化によって、質量行列を含んだ形になるかもしれません。そして、それに対する MATLAB ODE ソルバが用意されています。しばしば、システムはスティッフになります。これは、ODE が放物線型や、流体内での化学反応のような非常に異なる時間スケールで生じる現象の場合に期待されるようなものです。このような場合、4 つのソルバ、ode15s，ode23s，ode23t，ode23tb が用意されています。このシステムはしばしばスティッフになります。PDE が放物線の型であるときや液体の流動による化学反応のように異なる時間スケールで起こる現象があるときに、この事が予測されます。このような場合、ode15s か ode23s を使ってください。普通、多くの方程式がある場合、プロパティ JPattern を設定してください。これは簡単なことですが、非常に計算時間がかかるため上手く行くときと失敗するときで大きな差が出ます。システムがスティッフでないときあるいはそれほどスティッフではないときには、ode23 や ode45 が、ode15s，ode23s，ode23t，ode23tb より有効です。JPattern を使う例題は、15-36 ページの " 例題： 大規模、スティッフなスパース行列 " を参照してください。 1 次元の放物線 - 楕円偏微分方程式は、MATLAB PDE ソルバ pdepe で直接解くことができます。詳細は、15-75 ページの " 偏微分方程式 " を参照してください。
微分代数方程式 (DAE) を解けますか	はい、ソルバ ode15s と ode23t は、$M(t,y)y' = f(t,y)$ 型の DAE 問題を解くことができます。ここで、$M(t,y)$ は、正則ではありません (DAE 問題は、インデックス 1 でなければなりません)。たとえば、amp1dae や hb1dae を参照してください。

異なるシステムを解く場合（続き）

質問	回答
サンプリングされたデータによるシステムを積分できますか	直接的には無理です。補間やその他のデータフィッティングの手法を使って関数としてデータを表現しなければなりません。この関数の平滑化には危険があります。スプラインのような区分的多項式で置き換えた場合、一見滑らかにみえますが、ソルバにとっては不連続になっています。微分値が不連続になる点で、ソルバは非常に細かいステップを要求されます。データを表現する滑らかな関数を使用するか、感度的に落ちる低次のソルバ (ode23, ode23s, ode23t, ode23tb) を使用するかのどちらかを行ってください。
遅れを持った微分方程式を解けますか	直接的には無理です。いくつかのケースでは、1 回の遅れ時間より小さな時間ステップで分割してシミュレーションを行うことによって、遅れをもった微分方程式を解くために、初期値問題ソルバを使うことができます。詳細は、[5] を参照してください。
最終時間での値は分かっているが、初期値が分からない場合どうしますか。	このバージョンの MATLAB ode suite の ode45 や他のソルバでは時間について後ろ向きにもにも、前向きにも解くことができるので可能です。ソルバの書式は、つぎのようになります。 `[t,y] = ode45(odefun,[t0 tf],y0);` シンタックスは、t0 > tf です。

トラブルシューティング

質問	回答
計算結果が予想していたものと異なります。	予想が正しいのであれば、許容誤差値をデフォルトより小さくする必要があります。長すぎる時間間隔で積分した問題や、同様に少し不安定な問題の解を正確に計算するためには、より小さな相対的許容誤差値で計算する必要があります。ある時間において、求めた解のある要素がそれに対応する絶対的許容誤差値よりも小さくなっていないかチェックすべきです。その場合は、これらの要素について何桁の精度をもっているかを調べる必要はありません。これらの要素については、この現象は当然で、正確に計算できていない要素のために、それらに依存している他の要素の精度も落ちているかもしれません。
プロットが十分滑らかではありません。	デフォルトでは、ode45 で 4、その他で 1 となっている Refine の値を増やして下さい。Refine を大きくするほど、出力点数は多くなります。計算速度には、Refine の値は、ほとんど影響がありません。
計算できたところまでをプロットした結果は良いように見えますが、計算がある値で、止まってしまいました。	初めに、計算が止まる点の近くで ODE 関数が滑らかかであるかを確認して下さい。そうでなければ、その部分でもっと小さな時間刻みが必要になっています。tspan の値を ODE ファイルの滑らかなところでいくつかに分割することも良いかもしれません。 関数は、滑らかで計算が小さなステップ幅で進むのであれば、この問題を解こうとしたのとは別のスティッフソルバを使ってみてください。ode15s, ode23s, ode23t, ode23tb に変えてシミュレーションしてください。

トラブルシューティング （続き）

質問	回答
大変多くの時間ステップを使うので積分計算が非常に遅くなります。	初めに、tspan にあまり多くの点を設定していないかどうかを調べて下さい。ソルバは、なめらかな計算結果を出力するために必要なだけ多くの時間ステップで計算します。tspan に比べて非常に小さい時間スケールで ODE 関数が変化している場合、多くの時間ステップが必要になります。積分計算に長い時間がかかることは大きな問題です。tspan を小さくして計算してみて下さい。 tspan の上で、ODE 関数が目立って変化していないようであれば、その問題がスティッフであろうと考えられます。ode15s，ode23s，ode23t，ode23tb を使ってみてください。 最後に、ODE 関数が効率よく記述されているかを確かめて下さい。ODE 関数で微係数が何回も計算されます。数値積分に要する時間は、ODE 関数の計算に非常に大きく関わっています。複雑な計算になっている定数値を ODE 関数の計算のたびに計算し直すより、むしろ、これらをグローバルに値をストアしておくか、関数の外部で計算しておき、追加の引数として関数に引き渡した方が良いでしょう。
時間 t で解が急変することが分かっているのに、この点を見失って積分計算を行ってしまいました。ここで t0 」t 」tf です。	時間 t で急変することが分かっているのならば、tspan の間隔を [t0 t] と [t tf] の 2 つに分けて、2 回計算するのが良いかもしれません。 微分方程式が、周期的な係数、または、解をもっている場合、周期長に最大ステップサイズを設定し、ステップが周期長より大きくならないようにしてください。

ODE に対する境界値問題

この節は、常微分方程式 (ODEs) の境界値問題 (BVPs) で MATLAB をどのように使用するかを記述しています。つぎの事柄を説明しています。

- MATLAB BVP 関数とデモのまとめ
- BVPs の紹介
- BVP ソルバの記述とそのシンタックス
- MATLAB 内での BVP 表現の一般的な例
- 難しい問題を解くために使用する方法に関する議論と例題
- ソルバの性能の改良のためのデフォルトの積分プロパティの変更について

BVP 関数のまとめ

ODE 境界値問題のソルバ

ソルバ	詳細
bvp4c	常微分方程式に対する 2 点境界値問題を解きます。

BVP 補助関数

関数	詳細
bvpinit	bvp4c 用の初期推定値の作成
bvpval	bvp4c の出力を使って、数値解を計算します。

BVP オプションの取り扱い

options 構造体は、名前の付いたプロパティの値をソルバに送ります。そして、問題の解を与えます。これらの関数を使って、構造体を作成し、変更し、アクセスします。

関数	小差
bvpset	BVP options 構造体を作成 / 変更
bvpget	bvpset を使って、作成された options 構造体からプロパティを抽出します。

ODE 境界値問題のデモ

これらのデモは、MATLAB BVP ソルバを使って、解くことのできる種々の問題を示しています。MATLAB Help ブラウザから、デモ名をクリックして、エディッタを使って、デモコードを見ることができます。コマンドラインで、demoname と入力することで、デモを実行することができます。

デモ	詳細
mat4bvp	Mathieu 方程式の 4 番目の固有関数
shockbvp	解が、x = 0 近傍で、衝撃層をもつもの
twobvp	厳密な 2 つの解をもつ BVP

Shampine, Reichelt, and Kierzenka による "Solving Boundary Value Problems for Ordinary Differential Equations in MATLAB with bvp4c" に他の例題が記述されています、チュートリアルと例題は、ftp://ftp.mathworks.com/pub/doc/papers/bvp/ を参照しています。このチュートリアルは、手法を示しているもので、現実の問題としては、つまらないものです。

境界値 ODE 問題の紹介

BVP ソルバは常微分方程式を取り扱うために設計されています。

$$y' = f(x, y)$$

ここで、x は独立変数、y は従属変数、y' は dy/dx を表わします。

ODE の一般的な情報については、15-5 ページの " 常微分方程式とは " を参照してください。

境界条件を使って、興味のある解を指定

*境界値問題*で、対象の解は、境界条件を満足します。これらの条件は、一つのxでなく複数の値で、解の値の間の関係を記述しています。bvp4c は、2 点の BVP 問題を解くように設計されています。これは、区間$[a, b]$で求まった解が、つぎの境界条件を満足しなければならない問題であることを意味しています。

$$g(y(a), y(b)) = 0$$

初期値問題と異なり、境界値問題は、解をもたない可能性があり、また、無限個の解をもつ可能性もあります。BVP を解くプロセスの積分部として、必要な解に対する推定を与えることが必要になります。この推定の質は、ソルバの性能にクリティカルになり、連続計算さえもそのようになります。

無限大の区間の制約をもつ問題や、特異係数を含む問題を含む BVP 問題を解くときに、他の困難が生じます。しばしば、BVP 問題は、問題を解く部分で決定されなければならない未知のパラメータpを含んでいます。

$$y' = f(x, y, p)$$

$$g(y(a), y(b), p) = 0$$

境界条件は、pの値を決定するために十分なものです。

境界値問題ソルバ

この節では、つぎの事柄を記述します。

- BVP ソルバ、bvp4c
- BVP ソルバの基本的なシンタックス
- 付加的な BVP ソルバ引数

MATLAB Help ブラウザを使って、MATLAB 関数に対するシンタックスに関する情報を得ることができます。

BVP ソルバ

関数 bvp4c は、常微分方程式 (ODEs) に関する 2 点境界値問題を解きます。つぎの型をした 1 階の常微分方程式、

$$y' = f(x, y)$$

を、区間 $[a, b]$ で、一般的な 2 点の境界条件

$$bc(y(a), y(b)) = 0$$

の基で、シミュレーションします。これは、つぎの型をした問題に対して、未知パラメータを含ませることができます。

$$y' = f(x, y, p)$$
$$bc(y(a), y(b), p) = 0$$

この場合、境界条件の数は、解と未知パラメータを決定するのに十分になります。

bvp4c は、$[a, b]$ の区間で、解が連続で、また、一階微分も連続なものになります。関数 bvpval と bvp4c の出力を使って、シミュレーション区間での指定した点で、解を計算します。

bvp4c は、3 ステージ Lobatto IIIa 公式を実現する有限差分コードです。これは、選点法の方程式で、選点多項式は、シミュレーション区間で、一様な 4 次の精度である C1 連続解を与えます。メッシュの選択と、エラーコントロールは、連続解の残差をベースにしています。

選点法は、メッシュ点を使って、積分区間をサブ区間に分割します。ソルバは、境界条件から求まる代数方程式とサブ区間に分けた選点条件を使って、グローバルシステムを解くことにより、数値解を決定します。そして、ソルバは、各サブ区間に関して、数値解のエラーを計算します。解が、許容範囲の規範を満たさな

い場合、メッシュを使って、再度、処理を繰り返します。ユーザは、初期メッシュの点を与え、その点に解の初期近似を与える必要があります。

BVP ソルバの基本的なシンタックス

BVP ソルバの基本的なシンタックスは、つぎのようになります。

```
sol = bvp4c(odefun,bcfun,solinit)
```

基本的な入力引数は、つぎのものです。

odefun　微分方程式を計算する関数。これは、

```
dydx = odefun(x,y)
```

wher の型になります。x はスカラ、dydx と y は、列ベクトルです。odefun は、未知のパラメータのベクトルと既知のパラメータ変数数を受け入れることができます。

bcfun　境界条件の中での残差を計算する関数。これは、

```
res = bcfun(ya,yb)
```

の型になります。ここで、ya と yb は、y(a) と y(b) を表す列ベクトルです。そして、res は、境界条件を満足する中での残差の列ベクトルです。bcfun は、未知のパラメータのベクトルと既知のパラメータ変数数を受け入れることができます。

solinit　フィールド x と y をもつ構造体

- x　初期メッシュの順番付けられたノード。境界条件は、a = solinit.x(1) と b = solinit.x(end) になります。
- y　ノード solinit.x(i) での解に対する推定 solinit.y(:,i) をもつ解に対する初期推定

構造体は、任意の名前をもつことができ、フィールドは、x と y で名前付けられます。これは、未知パラメータに対する初期推定を与えるベクトルを含んでいます。補助関数 bvpinit を使って、solinit を作成できます。詳細は、bvpinit を参照してください。

出力引数 sol は、ソルバで作成される構造体です。基本的な場合、構造体は、フィールド x, y, yp をもちます。

sol.x	bvp4c で選択されるメッシュノード
sol.y	sol.x のメッシュ点で、$y(x)$ を近似
sol.yp	sol.x のメッシュ点で、$y'(x)$ を近似
sol.parameters	存在すると、ソルバで検出される未知パラメータ値

関数bvpvalは、出力構造体solを使って、[a,b]の任意の点で、数値解を計算します。

Ad 付加的な BVP ソルバ引数

よりアドバンストなアプリケーション用に、入力引数として、ソルバオプションと付加的な既知パラメータを指定できます。

options　デフォルトの積分プロパティを変更するオプションパラメータの構造体。これは、４番目の入力引数になります。

```
sol = bvp4c(odefun,bcfun,solinit,options)
```

15-70ページの"BVP Options構造体の作成と取り扱い" では、ユーザが設定できる構造体の作成法やプロパティの記述法を説明しています。

p1,p2...　*既知*のパラメータで、ソルバは、odefun や bcfun に渡します。

```
sol = bvp4c(odefun,bcfun,solinit,options,p1,p2...)
```

ソルバは、options 引数の後に、任意の数だけ設定した入力パラメータを、コールする度に、odefun や bcfun に、渡します。options が設定されていない場合は、options = [] を使います。odefun 引数リストの中で、既知のパラメータは、x,y に、存在するならば、未知のパラメータ (parameters) に続きます。

```
dydx = odefun(x,y,p1,p2,...)
dydx = odefun(x,y,parameters,p1,p2,...)
```

bcfun 引数リストの中で、既知のパラメータは、ya、yb、また、存在するならば、未知パラメータベクトルに続きます。

```
res = bcfun(ya,yb,p1,p2,...)
res = bcfun(ya,yb,parameters,p1,p2,...)
```

例題は、15-67 ページの " 例題： 連続性を使って、難しい BVP を解く " を参照してください。

BVP 問題の表現

この節は、つぎの事柄を記述します。

- MATLAB ソルバ、bvp4c を使って、境界値問題 (BVPs) を解くプロセス
- 未知パラメータパラメータの検出
- 指定した点での解の計算

例題： Mathieu の方程式

この例題は、Mathieu の方程式の４番目の固有値を決定します。ここでは、２階の微分方程式を、１階の２つの微分方程式として記述する方法を示し、そして、未知のパラメータλを bvp4c を使って決定する方法を示します。

ここでは、まず、Mathieu の方程式の４番目 ($q = 5$) の固有値λを計算します。

$$y'' + (\lambda - 2q\cos 2x)y = 0$$

未知のパラメータλが存在する場合、つぎの２階微分方程式は、３つの境界条件に従います。

$$y(0) = 1$$
$$y'(0) = 0$$
$$y'(\pi) = 0$$

注意　デモ mat4bvp は、この例題のコード全体を示すものです。デモは、サブ関数を使って、一つの M- ファイルの中に、bvp4c で必要とされるすべての関数を配置します。コマンドラインで、この例題を実行するには、mat4bvp と入力してください。詳細は、15-60 ページの "BVP ソルバの基本的なシンタックス " を参照してください。

---

1 1次のシステムへの書き換え． bvp4c を使って、方程式を、1次の微分方程式の等価なシステムとして書き換えます。$y_1 = y$と$y_2 = y'$を代入して、微分方程式は、2つの1次方程式として記述されます。

$$y_1' = y_2$$
$$y_2' = -(\lambda - 2q\cos 2x)y_1$$

微分方程式は、未知パラメータλに依存していることに注意してください。境界条件は、つぎのようになります。

$$y_1(0) - 1 = 0$$
$$y_2(0) = 0$$
$$y_2(\pi) = 0$$

2 MATLAB で1次の ODEs システムをコード化． ユーザは、方程式を1次のシステムとして表現すると、bvp4c が使用できる関数として、それをコード化できます。未知のパラメータが存在するので、関数は、つぎの型になります。

```
dydx = odefun(x,y,parameters)
```

つぎのコードは、MATLAB 関数 mat4ode のシステムを表わします。

```
function dydx = mat4ode(x,y,lambda)
q = 5;
dydx = [   y(2)
         -(lambda - 2*q*cos(2*x))*y(1) ];
```

vp4c での未知パラメータの使用の詳細は、15-65 ページの " 未知パラメータの検出 " を参照してください。

3 境界条件関数のコード化． ユーザは、MATLAB 関数の中に境界条件をコード化する必要があります。未知パラメータが存在するので、関数は、つぎの型で表します。

```
res = bcfun(ya,yb,parameters)
```

つぎのコードは、MATLAB 関数 mat4bc 内に表現した境界条件です。

```
function res = mat4bc(ya,yb,lambda)
res = [  ya(2)
         yb(2)
```

```
        ya(1)-1 ];
```

4 初期推定の作成． 推定構造体 solinit は、bvpinit を使って作成されます。解と未知パラメータに対する初期推定が必要になります。

解に対する初期推定は、関数 mat4init の型で与えられます。この関数として、境界条件を満足し、正しい量的な挙動 (符号の変化回数) をもっているので、$y = \cos 4x$ を選択します。

```
function yinit = mat4init(x)
yinit = [  cos(4*x)
          -4*sin(4*x) ];
```

bvpinit へのコールの中で、3 番目の引数 (lambda) は、未知パラメータλとして初期推定を与えます。

```
lambda = 15;
solinit = bvpinit(linspace(0,pi,10),@mat4init,lambda);
```

この例題は、@ を使って、bvpinit への関数ハンドルとして、mat4init に渡します。

注意 関数ハンドルに関する情報については、function_handle (@), func2str, str2func のリファレンスページ、および MATLAB のドキュメントの中の "Programming with MATLAB" の "M-Files Programming" を参照してください。

---

5 BVP ソルバの適用． mat4bvp 例題は、関数 mat4ode と mat4bc をもつ関数と共に bvp4c をコールし、構造体 solinit が、bvpinit と共に作成できます。

```
sol = bvp4c(@mat4ode,@mat4bc,solinit);
```

6 結果の可視化． 例題では、結果をすべて表示します。

- bvp4c で求まった、未知パラメータλの値を印刷します。

  ```
  fprintf('The fourth eigenvalue is approximately %7.3f.\n',...
          sol.parameters)
  ```

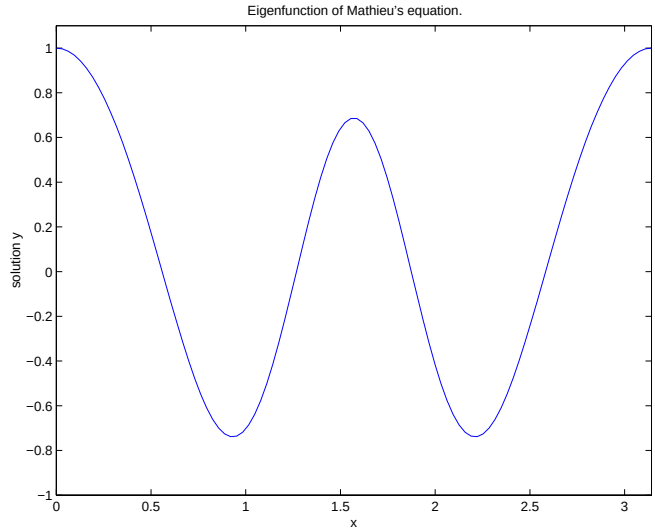
- bvpval を使って、[0, π] の区間で、等間隔に分布している 100 点で数値解を計算します。そして、最初の要素をプロットします。この要素は、$y(x)$ の近似です。

  ```
  xint = linspace(0,pi);
  Sxint = bvpval(sol,xint);
  ```

```
plot(xint,Sxint(1,:))
axis([0 pi -1 1.1])
title('Eigenfunction of Mathieu''s equation.')
xlabel('x')
ylabel('solution y')
```

bvpval の使い方に関する情報は、15-66 ページの " 指定した点の解の計算 " を参照してください。

つぎのプロットは、最終固有値λ = 17.097 に関連した固有関数を示しています。



未知パラメータの検出

bvp4c ソルバは、つぎの型の問題に対して、未知パラメータ p を求めることができます。

$$y' = f(x, y, p)$$
$$bc(y(a), y(b), p) = 0$$

ユーザは、ベクトル solinit.parameters の中に未知パラメータに対する初期推定を bvp4c に与えます。solinit を作成するために、bvpinit をコールすると、付加的な引数 parameters の中にベクトルとして、初期推定を設定します。

```
solinit = bvpinit(x,v,parameters)
```

bvp4c 関数引数 odefun と bcfun は、各々 3 番目の引数として設定します。

```
dydx = odefun(x,y,parameters)
res = bcfun(ya,yb,parameters)
```

bvp4c ソルバは、各繰り返しで、未知パラメータの中間的な値を計算し、最終値を odefun や bcfun の引数 parameters に渡します。ソルバは、これらの未知パラメータの最終値を sol.parameters に出力します。

指定した点の解の計算

bvp4c で実行する選点法は、積分区間 $[a, b]$ 全体に渡り、C1 の連続性を示します。ユーザは、bvp4c と補助関数 bvpval を使って、戻される構造体 sol を使って、$[a, b]$ の中の任意の点で近似解 $S(x)$ を計算することができます。

```
Sxint = bvpval(sol,xint)
```

関数 bvpval は、ベクトル化されています。ベクトル xint に対して、Sxint の i 番目の列は、解 y(xint(i)) を近似します。

良い初期推定の作成法

境界値問題を解くために、解に対する初期推定を与える必要があります。ユーザの設定する推定の質が、ソルバの性能に影響を与え、問題全体を解くこともできます。しかし、うまい推定に達することは、境界値問題を解くために必要な事柄の大部分を占めることになります。確かに、ユーザは、問題の物理的な根源の知識を適用する必要があります。

良い推定を行う一つの方法は、比較的簡単な問題、すなわち、連続として、問題を解くことです。問題の解は、つぎの問題を解くために良い推定を与えることになります。

つぎの例題は、連続を使って、難しい問題をどのように解くかを示しています。

注意 デモ shockbvp は、つぎの例題のすべてのコードを示しています。デモは、サブ関数を使い、一つの M- ファイルの中にすべての必要な関数を配置しています。この例題を実行するためには、コマンドラインで、shockbvp と入力してください。詳細は、15-60 ページの "BVP ソルバの基本的なシンタックス " と 15-62 ページの "BVP 問題の表現 " を参照してください。

例題： 連続性を使って、難しい BVP を解く

つぎの例題は、境界条件$y(-1) = -2$と$y(1) = 0$を使って区間[-1 1]で $\varepsilon = 10^{-4}$ に対して、つぎの微分方程式を解きます。

$$\varepsilon y'' + xy' = \varepsilon\pi^2\cos(\pi x) - \pi x\sin(\pi x)$$

$0 < \varepsilon < 1$ に対して、解は、$x = 0$ で転位層をもっています。ε の小さな値に対して、解の中の急激な変化のために、問題は、数値的に解くことが困難になります。

注意　この問題は、[1] に記述され、うまく作成した BVP コード COLSYS のメッシュ選択機能を示すものです。

---

1 ODE と境界条件関数のコード化． bvp4c が使用できる関数として、微分方程式と境界条件をコード化します。付加的な既知パラメータεが存在するので、関数は、つぎのような型になる必要があります。

```
dydx = odefun(x,y,p1)
res = bcfun(ya,yb,p1)
```

MATLAB 関数 shockODE と shockBC に、微分方程式と境界条件を、それぞれ、コード化しています。付加的なパラメータεを、e で表します。

```
function dydx = shockODE(x,y,e)
pix = pi*x;
dydx = [  y(2)
         -x/e*y(2) - pi^2*cos(pix) - pix/e*sin(pix) ];

function res = shockBC(ya,yb,e)
res = [ ya(1)+2
        yb(1)   ];
```

例題は、付加的な引数として、e を bvp4c に渡します。

```
sol = bvp4c(@shockODE,@shockBC,sol,options,e);
```

その後、bvp4c は、これらを計算するときに、この引数を関数 shockODE と shockBC に渡します。詳細は、15-61 ページの "Ad 付加的な BVP ソルバ引数 " を参照してください。

2 解析的な偏微分係数を与える． この問題に対して、ソルバは、解析的な偏微分係数を利用します。つぎのコードは、関数 shockJac と shockBCJac に、微係数を表しています。

```
function jac = shockJac(x,y,e)
jac = [ 0   1
        0 -x/e ];

function [dBCdya,dBCdyb] = shockBCJac(ya,yb,e)
dBCdya = [ 1 0
           0 0 ];
dBCdyb = [ 0 0
           1 0 ];
```

bvp4c は、付加的な引数をユーザが与えるすべての関数に渡すので、shockJac と shockBCJac は、付加的な引数 e を受け入れなければなりません。

オプションの FJacobian と BCJacobian を設定することで、偏微分係数を計算するために、これらの関数を使うことを bvp4c に知らせます。

```
options = bvpset('FJacobian',@shockJac,...
                 'BCJacobian',@shockBCJac);
```

3 初期推定の作成． 初期メッシュとメッシュ点での解の値に対する推定を含む推定構造体を、bvp4c に与える必要があります。$y(x) \equiv 1$ と $y'(x) \equiv 0$, の定数推定と [-1 1] 上を５等分に分割した点からなるメッシュを使って、$\varepsilon = 10^{-2}$ に対する問題を解きます。bvpinit を使って、推定構造体を作成します。

```
sol = bvpinit([-1 -0.5 0 0.5 1],[1 0]);
```

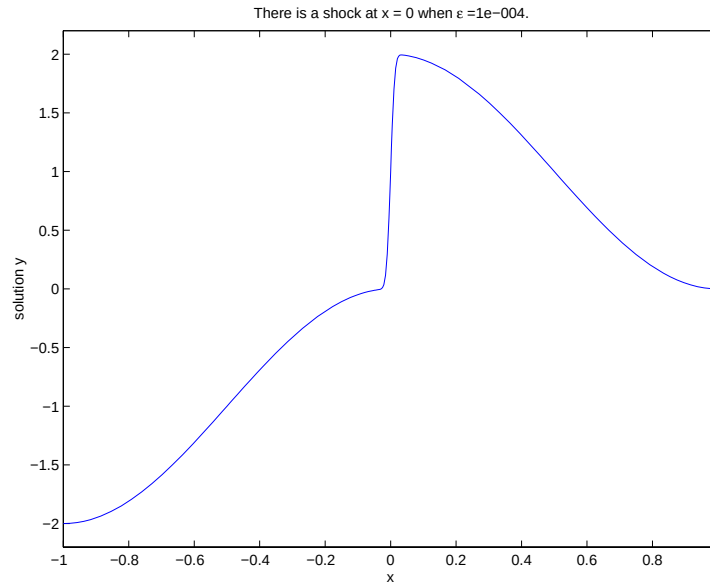
4 連続性を使って、問題を解きます． パラメータ $\varepsilon = 10^{-4}$ を使って、解を得るために、この例題では、$\varepsilon = 10^{-2}, 10^{-3}, 10^{-4}$ をパラメータとして、一連の問題を作成し、それを連続的 (その結果を利用して) に解くことを示します。ソルバ bvp4c は、自動的には、このような操作を行ないませんが、コード化したユーザインタフェースは、このような処理を簡単に行なえるように設計されています。bvp4c が、つぎの繰り返しの推定として、e の中の一つの値に対して作成した出力 sol を使います。

```
e = 0.1;
for i=2:4
    e = e/10;
    sol = bvp4c(@shockODE,@shockBC,sol,options,e);
```

```
end
```

5 結果の可視化． 最終解を表示します。

```
plot(sol.x,sol.y(1,:))
axis([-1 1 -2.2 2.2])
title(['There is a shock at x = 0 when \epsilon ='...       sprintf('%.e',e)
'.'])
xlabel('x')
ylabel('solution y')
```



BVP ソルバ性能の改良

BVP ソルバ bvp4c の中のデフォルトの積分プロパティは、一般的な問題を取り扱うことができるように選択されています。いくつかの場合、これらのデフォルト値を変更することで、ソルバの性能を改良することができます。これを行うには、Options 構造体の中の一つ、または、複数のプロパティ値を変更して、bvp4c と共に使います。

```
sol = bvp4c(odefun,bcfun,solinit,options)
```

この節では、つぎの事柄を扱います。

- Options 構造体の作成、変更、引用方法について
- ユーザが Options 構造体の中で使用できるプロパティの記述

つぎのテーブル、また、それに続く、プロパティテーブルで、最も一般的に使われるプロパティカテゴリを表示し、その後に、アドバンスなものを表示します。

BVP プロパティカテゴリ

プロパティカテゴリ	プロパティ名
誤差の許容範囲	RelTol, AbsTol
解析的な偏微分係数	FJacobian, BCJacobian
メッシュサイズ	NMax
出力表示	Stats

注意 ソルバの効率を向上させる他の方法については、ページ 66 の良い初期推定の作成法 と、ftp://ftp.mathworks.com/pub/doc/papers/bvp/ で利用できるチュートリアル "Solving Boundary Value Problems for Ordinary Differential Equations in MATLAB with bvp4c" を参照してください。

BVP Options 構造体の作成と取り扱い

関数 bvpset は、bvp4c に提供する options 構造体を作成します。bvpget を使って、指定したプロパティの値を、options 構造体から取得することができます。

Options 構造体の作成. 関数 bvpset は、つぎのシンタックスを使って、プロパティ名 / 値を組として与えます。

```
options = bvpset('name1',value1,'name2',value2,...)
```

これは、名前を設定したプロパティに、値を指定した options 構造体を作成します。指定していないプロパティは、デフォルト値が使われます。すべてのプロパ

ティに対して、プロパティ名を判断できるユニークな名前を設定することができ、大文字、小文字の区別も行ないません。

入力引数を設定しない場合、bvpset は、すべてのプロパティ名と設定可能な値を表示し、デフォルト値を { } で示します。

既存の options 構造体の修正． 既に存在する options 構造体を修正するには、つぎのシンタックスを使います。

```
options = bvpset(oldopts,'name1',value1,...)
```

これは、oldopts の中の値を、名前 / 値の組を使って設定したもので書き換えます。修正した構造体は、出力引数として出力されます。同様に、つぎのコマンド

```
options = bvpset(oldopts,newopts)
```

は、構造体 oldopts と newopts を組み合わせます。出力引数 options の中で、いくつかの値は、oldopts のものを newopts で書き換えた結果になります。

Options 構造体の取得． 関数 bvpget は、bvpset で作成した options 構造体からプロパティ値を抽出します。

```
o = bvpget(options,'name')
```

これは、指定したプロパティ値を出力します。また、options 構造体に指定したものが存在しない場合は、空行列 [] を出力します。

bvpset と同様に、一意的にプロパティ名を識別できる先頭の文字を入力するだけで十分です。プロパティ名では、大文字と小文字の区別は行いません。

誤差許容範囲のプロパティ

bvp4c は、選点法の方程式を使うので、数値的な解は、選点方程式を満足するメッシュ点をベースにしています。メッシュ選択と誤差のコントロールは、計算した解 $S(x)$ が、擾乱問題 $S'(x) = f(x, S(x)) + res(x)$ の厳密解であるような解の残差をベースにしています。メッシュのサブ区間上で、解の i 番目の残差のノルム、res(i) が計算され、許容範囲より小さくすべきか、等しくすべきかを設定します。この許容範囲は、ユーザが設定できる相対許容範囲 RelTol と絶対許容範囲 AbsTol の関数になります。

$$\| (\mathrm{res(i)} / \max(\mathrm{abs(f(i))}, \mathrm{AbsTol(i)} / \mathrm{RelTol})) \| \le \mathrm{RelTol}$$

つぎのテーブルに、誤差の許容範囲のプロパティを記述します。bvpset を使って、許容範囲を設定します。

BVP 誤差許容範囲のプロパティ

プロパティ	値	詳細
RelTol	正のスカラ {1e-3}	残差ベクトルのすべての要素に適用する相対誤差許容範囲。$f(x,y)$ のサイズに対する残差の尺度になります。デフォルト値は、10^(-3) (0.1% の精度)
AbsTol	正のスカラ、または、ベクトル {1e-6}	残差ベクトルの対応する要素に適用する絶対誤差許容範囲。AbsTol(i) は、対応する成分の値が重要でないと判断するスレッシュホールドです。スカラ値が設定されている場合、すべての要素に同一のものが適用されます。

解析的な偏微分係数

デフォルトで、bvp4c で使用するすべての偏微分係数は、有限差分で近似されます。微分方程式の解析的な偏微分係数 $\partial f/\partial y$ と境界条件の解析的な偏微分係数 $\partial bc/\partial ya$ と $\partial bc/\partial yb$ を与えることで、bvp4c は、効率良くなります。問題に未知パラメータが含まれている場合、パラメータに関する偏微分係数 $\partial f/\partial p$ と $\partial bc/\partial p$ も与えなければなりません。

つぎのテーブルは、解析的な偏微分プロパティを記述しています。bvpset を使って、これらのプロパティを設定します。

BVP の解析的な偏微分方程式プロパティ

プロパティ	値	詳細
FJacobian	関数	この関数は、$f(x,y)$ の解析的な偏微分係数を計算します。$y' = f(x,y)$ を解く場合、dfdy = fjac(x,y) が、ヤコビアン $\partial f/\partial y$ を計算する場合、このプロパティを @fjac に設定してください。問題に未知パラメータ p を含む場合、[dfdy,dfdp] = fjac(x,y,p) は、偏微分係数 $\partial f/\partial p$ を戻します。
BCJacobian	関数	関数は、$bc(ya,yb)$ の解析的な偏微分係数を計算します。境界条件 $bc(ya,yb)$ に対して、[dbcdya,dbcdyb] = bcjac(ya,yb) は、偏微分 $\partial bc/\partial ya$ と $\partial bc/\partial yb$ を計算する場合、このプロパティを @bcjac に設定します。問題が、未知パラメータ p を含んでいる場合、[dbcdya,dbcdyb,dbcdp] = bcjac(ya,yb,p) は、偏微分係数 $\partial bc/\partial p$ も戻します。

メッシュサイズプロパティ

bvp4c は、代数方程式を解いて、メッシュの各点で、BVP の数値解を決定します。代数システムのサイズは、微分方程式の数 (n) とカレントメッシュのメッシュ点数 (N) に依存します。許されるメッシュ点数が非常に多い場合、計算は停止し、bvp4c は、ワーニングメッセージを表示し、その時点で求まっている解を戻します。この解は、誤差の許容範囲を満足しているものではありませんが、誤差の許容範囲を緩くするか、または、NMax の数を増やすか、どちらかを行って、計算を再スタートするための非常に良い初期推定になります。

つぎのテーブルは、メッシュサイズプロパティを記述しています。このプロパティを設定するには、bvpset を使ってください。

BVP メッシュサイズプロパティ

プロパティ	値	詳細
NMax	正の整数 {floor(1000/n)}	BVP を解く場合、使用可能なメッシュ点の最大数。ここで、n は、問題の中の微分方程式の数です。NMax のデフォルト値は、代数方程式のサイズを約 1000 の方程式に制限します。2、3 の微分方程式システムに対して、NMax のデフォルト値は、正確な解を得るのに十分と考えられます。

解の統計的なプロパティ

Stats プロパティは、解の統計に関する量を示します。

つぎのテーブルは、解の統計的なプロパティを記述しています。bvpset を使って、このプロパティを設定します。

BVP 解の統計的なプロパティ

プロパティ	値	詳細
Stats	on | {off}	計算に関する統計量を表示するか否かを設定。stats プロパティが on の場合、問題を解いた後で、bvp4c は、プロットします。 • メッシュ内の点数 • 解の残差の最大値 • 微分方程式関数が、$f(x,y)$ を計算するためにコールされる最大回数 • $bc(y(a),y(b))$ を計算するために、境界条件関数 bcfun をコールする最大回数

偏微分方程式

この節は、MATLAB を使って、偏微分方程式 (PDEs) の初期値 - 境界値問題を解く方法を記述します。つぎの項目を説明します。

- MATLAB 関数とデモのまとめ
- PDEs の紹介
- PDE ソルバとそのシンタックスの記述
- 例題を含んで、MATLAB の中での PDE の表現の基本的なもの
- ソルバ性能の改良のためのデフォルトの積分プロパティの変更に関する議論
- 実社会の問題の解法例題

PDE 関数のまとめ

MATLAB PDE ソルバ

ここでは、MATLAB PDE ソルバについて記述します。

PDE 初期 - 境界値問題のソルバ	
pdepe	一つの空間変数と時刻から構成される放物線型と楕円型、偏微分方程式 (PDEs) の初期 - 境界値問題を解きます。

PDE 補助関数

PDE 補助関数	
pdeval	pdepe の出力を使って、PDE の数値解を計算します。

PDE デモ

つぎのデモは、MATLAB PDE ソルバを使って解くことのできるいくつかの問題を示しています。MATLAB Help ブラウザから、デモ名をクリックして、エディッタにデ

モコードを表示します。コマンドラインで、demoname と入力することで、デモを実行することができます。

デモ	詳細
pdex1	簡単な PDE が、直接的な定式化、計算、解のプロット手法を示します。
pdex2	不連続を含む問題
pdex3	偏微分係数の値の計算を含む
pdex4	短い t で、急激な変化と両端で境界層をもつ2つのPDEsのシステム
pdex5	初期条件としてステップ関数をもつ PDEs システム

PDE 問題の紹介

つぎの型をした、一つの空間変数 x と時刻 t からなる PDEs を pdepe は解きます。

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^{m} f\left(x, t, u, \frac{\partial u}{\partial x}\right)\right) + s\left(x, t, u, \frac{\partial u}{\partial x}\right) \qquad (15\text{-}3)$$

PDE 問題は、$t_0 \le t \le t_f$ と $a \le x \le b$ で成り立ちます。区間 $[a, b]$ は、有限で、m は、0, 1, 2 のいずれかで、スラブ、円筒、球面対称も対応します。$m > 0$ の場合、$a \ge 0$ もまた成り立ちます。

式 15-3 で、$f(x, t, u, \partial u/\partial x)$ は流量の項で、$s(x, t, u, \partial u/\partial x)$ はソース項です。時間に関する偏微係数のカップリングは、対角行列 $c(x, t, u, \partial u/\partial x)$ との乗算に制限されています。この行列の対角要素は、ゼロ、または、正になります。ゼロの要素は、楕円方程式に対応し、その他は、放物線方程式に対応します。少なくとも、一つは、放物線方程式が存在します。放物線方程式に対応する c の要素は、メッシュ点が存在する場合、x の孤立値でゼロになります。メッシュ点が各インタフェースに配置される場合、物質の境界による c と s の中の不連続部も取り扱えます。

初期時刻 $t = t_0$ で、すべての x に対して、解要素は、つぎの型の初期条件を満足します。

$$u(x, t_0) = u_0(x) \qquad (15\text{-}4)$$

$x = a$ または$x = b$の境界で、すべてのtに対して、解要素は、つぎの型をした境界条件を満足します。

$$p(x, t, u) + q(x, t)f\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0 \qquad (15\text{-}5)$$

$q(x, t)$ は、すべてがゼロか、または、ゼロでないかの要素をもつ対角行列です。境界条件が、$\partial u/\partial x$ より、むしろ、流量の項fで表現されることに注意してください。また、2 つのパラメータに関しては、p のみが、u に依存します。

MATLAB 偏微分方程式ソルバ

この章では、つぎのことを記述します。

- PDE ソルバ pdepe
- PDE ソルバの基本的なシンタックス
- PDE ソルバへの付加的な引数

PDE ソルバ

MATLAB PDE ソルバ、pdepe は、一つの空間変数xと時間tを使って、放物線型、楕円型 PDEs システムに対する初期値 - 境界値問題を解きます。システムの中で、少なくとも放物線方程式が一つ必要になります。

ユーザが設定するノードをベースに 2 次の空間的な離散化を使って、pdepe ソルバは、PDEs を ODEs に変換します。離散化手法は、[7] に記述されています。時間積分は、ode15s を使って行ないます。pdepe ソルバは、式 15-3 が、楕円方程式を含む場合に生じる微分代数方程式を解くためと指定したスパースパターンをもつヤコビアンの取り扱いに、ode15s の機能を使っています。ode15s は、時間ステップと定式化をドラマテックに変化させます。

離散化の後、楕円方程式は、代数方程式になります。楕円方程式に対応した初期条件の要素が、離散化と整合性を保っていない場合、pdepe は、時間積分が始まる前にそれらを調整しようとします。このため、最初の時刻に対して出力される解は、他の時間のものと比べて離散化誤差をもっています。メッシュが十分細かい場合、pdepe は、与えられた初期条件に似た整合性のある初期条件を見つけることができます。pdepe が、整合性のある初期条件を見つけることが困難であるメッセージを表示する場合、メッシュをより細かく分割してください。放物線方程式に対応する初期条件ベクトルの要素に対して、調整は必要ありません。

PDE ソルバの基本的なシンタックス

ソルバの基本的なシンタックスは、つぎの型をしています。

```
sol = pdepe(m,pdefun,icfun,bcfun,xmesh,tspan)
```

注意 与えられた対応は、15-76 ページの "PDE 問題の紹介" で使われた項目です。

入力引数は、つぎのものです。

m　問題の対称性を指定します。m が、0 の場合 = slab, 1 の場合 = cylindrical、2 の場合 = spherical です。式 15-3 の中の m に対応します。

pdefun　PDE の要素を定義します。式 15-3 の中の項 c, f, s を計算し、つぎの型をしています。

```
[c,f,s] = pdefun(x,t,u,dudx)
```

ここで、x と t はスカラ、u と dudx はベクトルで、u と x に関するその偏微分係数を近似します。c, f, s は列ベクトルです。c は、行列 c の対角要素を保存します。

icfun　初期条件を計算します。つぎの型をしています。

```
u = icfun(x)
```

引数 x と共にコールすると、icfun は、列ベクトル u の中の x で解要素の初期値を計算し、出力します。

bcfun　境界条件の項 p と q を計算します。つぎの型をしています。

```
[pl,ql,pr,qr] = bcfun(xl,ul,xr,ur,t)
```

ここで、ul は、左境界 xl = a での近似解で、ur は、右境界 xr = b での近似解です。pl と ql は、xl で計算した p と q の対角に対応した列ベクトルです。同様に、pr と qr が、xr に対応します。$m > 0$ と $a = 0$ の場合、$x = 0$ の近傍での解の制約は、$a = 0$ で流量 f がゼロになることを必要とします。pdepe は、この境界条件を自動的に課し、pl と ql の中に戻る値を無視します。

xmesh　tspan の中のすべての値に対して必要となる数値解に位置を設定するベクトル [x0, x1, ..., xn] 。x0 と xn は、それぞれ、*a* と *b* に対応します。

解の 2 次近似は、xmesh の中で指定したメッシュ上で行われます。一般に、解に急激に変化する部分では、非常に密なメッシュ点を使うことがベストです。pdepe は、*x* で、メッシュの選択を自動的には行ないません。ユーザは、適切な固定されたメッシュを xmesh に与える必要があります。コストは、xmesh の長さに強く関連しています。$m > 0$ の場合、座標の特異性のために、$x = 0$ の近傍で細かいメッシュを使う必要はありません。

xmesh の要素は、x0 < x1 < ... < xn を満足している必要があります。xmesh の長さは、≥ 3 でなければなりません。

tspan　xmesh の中のすべての点に対して、必要となる解の点を設定するベクトル [t0, t1, ..., tf] 。t0 と tf は、それぞれ、t_0 と t_f に対応します。

pdepe は、ODE ソルバを使って、時間積分を行うときに、動的に時間ステップと定式化を選択します。tspan の中で設定した点での解は、積分の定式を連続的に拡張する手法を使って得られます。tspan の要素は、ユーザが知りたい位置を指定するもので、コストは、tspan の長さにわずかに影響されます。

tspan の要素は、t0 < t1 < ... < tf を満足します。tspan の長さは、≥ 3 を満足しなければなりません。

出力引数 sol は、つぎのような 3 次元配列になります。

- sol(:,:,k) は、解 *u* の k 番目の要素の近似を行ないます。
- sol(i,:,k) は、時刻 tspan(i) とメッシュ点 xmesh(j) で解の k 番目の近似を行ないます。
- sol(i,j,k) は、時刻 tspan(i) とメッシュ点 xmesh(j) で解の k 番目の近似を行ないます。

PDE ソルバの付加的な引数

よりアドバンスなアプリケーションに対して、入力引数として、ソルバオプションと付加的なパラメータを PDE 関数に渡すことができます。

options　オプションパラメータの構造体は、デフォルトの積分プロパティを変更します。これは、7 番目の入力引数として設定します。

```
sol = pdepe(m,pdefun,icfun,bcfun,...
            xmesh,tspan,options)
```

詳細は、15-86 ページの "PDE ソルバ性能の改良 " を参照してください。

p1,p2...　ソルバが、pdefun, icfun, bcfun に渡すパラメータ

```
sol = pdepe(m,pdefun,icfun,bcfun,xmesh,tspan,...
            options,p1,p2...)
```

ソルバは、options 引数の後に設定した任意の入力引数を、pdefun, icfun, bcfun に、コールする度に渡します。オプションがない場合、options = [] を使います。pdefun 引数リストの中で、パラメータは、x, t, u, dudx の後に設定されます。

```
f = pdefun(x,t,u,dudx,p1,p2,...)
```

icfun 引数リストの中で、パラメータは、x の後に続きます。.

```
res = bcfun(x,p1,p2,...)
```

bcfun 引数リストの中で、パラメータは、xl, ul, xr, ur, t の後に続きます。

```
res = bcfun(xl,ul,xr,ur,t,p1,p2,...)
```

例題のデモ pdex3 を参照してください。

PDE 問題の表現

この節では、つぎの事柄を説明します。

- MATLAB ソルバ pdepe を使って、PDE 問題を解くプロセス
- 指定した点で、解の計算

例題： 単純 PDE

つぎの例題は、直接的な定式化、解、単純な PDE の解のプロットに関する事柄を示します。

$$\pi^2 \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

この方程式は、時刻$t \geq 0$ に関して、$0 \leq x \leq 1$ の区間で成り立ちます。$t = 0$ で、解は、つぎの初期条件を満足します。

$$u(x, 0) = \sin \pi x$$

$x = 0$ と$x = 1$ で、解は、つぎの境界条件を満足します。

$$u(0, t) = 0$$

$$\pi e^{-t} + \frac{\partial u}{\partial x}(1, t) = 0$$

注意 デモ pdex1 は、この例題に対するコードをすべて含んでいます。デモは、単一の M- ファイルの中に、必要なすべてのサブ関数を設定しています。デモを実行するには、コマンドラインで、pdex1 と入力してください。詳細は、15-78 ページの "PDE ソルバの基本的なシンタックス " を参照してください。

---

1 PDE の書き換え． つぎの型に PDE を書き換えます

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right) \frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x}\left(x^m f\left(x, t, u, \frac{\partial u}{\partial x}\right)\right) + s\left(x, t, u, \frac{\partial u}{\partial x}\right)$$

これは、式 15-3 で示される型で、pdepe で使用可能なものです。詳細は、15-76 ページの "PDE 問題の紹介 " を参照してください。この例題では、結果の方程式

$$\pi^2 \frac{\partial u}{\partial t} = x^0 \frac{\partial}{\partial x}\left(x^0 \frac{\partial u}{\partial x}\right) + 0$$

で、パラメータ $m = 0$ と以下の項になります。

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right) = \pi^2$$

$$f\left(x, t, u, \frac{\partial u}{\partial x}\right) = \frac{\partial u}{\partial x}$$

$$s\left(x, t, u, \frac{\partial u}{\partial x}\right) = 0$$

2 MATLAB での PDE のコード化． 上で示される型に PDE を書き直し（ 式 15-3)、項を設定すると、pdepe が使用可能な関数に、PDE をコード化できます。関数は、つぎの型をしています。

```
[c,f,s] = pdefun(x,t,u,dudx)
```

ここで、c，f，s は、それぞれ、*c* ，*f* ，*s* の項に対応します。つぎのコードは、例題の問題に対して、c，f，s を計算します。

```
function [c,f,s] = pdex1pde(x,t,u,DuDx)
c = pi^2;
f = DuDx;
s = 0;
```

3 初期条件関数をコード化． MATLAB の中で、つぎの型の初期条件をコード化します。

```
u = icfun(x)
```

つぎのコードは、MATLAB 関数 pdex1ic に初期条件を表しています。

```
function u0 = pdex1ic(x)
u0 = sin(pi*x);
```

4 境界条件関数をコード化． MATLAB の中で、つぎの型の境界条件をコード化します。

```
[pl,ql,pr,qr] = bcfun(xl,ul,xr,ur,t)
```

式 15-5 と同じ型で記述された境界条件が、つぎのものです。

$$u(0, t) + 0 \cdot \frac{\partial u}{\partial x}(0, t) = 0 \qquad \text{at } x = 0$$

と

$$\pi e^{-t} + 1 \cdot \frac{\partial u}{\partial x}(1,t) = 0 \qquad \text{at } x = 1$$

です。

つぎのコードは、MATLAB 関数 pdex1bc の中の境界条件の要素 $p(x,t,u)$ と $q(x,t)$ を計算するものです。

```
function [pl,ql,pr,qr] = pdex1bc(xl,ul,xr,ur,t)
pl = ul;
ql = 0;
pr = pi * exp(-t);
qr = 1;
```

関数 pdex1bc の中で、pl と ql は、左の境界条件 ($x = 0$) に対応し、pr と qr は右の境界条件 ($x = 1$) に対応します。

5 解に対するメッシュ点の選択 . MATLAB PDE ソルバを使う前に、pdepe を使って、解を計算するメッシュ点 (t,x) を指定する必要があります。これらは、ベクトル t と x で設定できます。

ベクトル t と x は、ソルバの中で、異なる役割を行います (15-77 ページの "MATLAB 偏微分方程式ソルバ " を参照)。特に、解のコストと精度は、ベクトル x の長さに強く依存します。しかし、計算は、ベクトル t の中の要素にはあまり影響を受けません。

この例題は、空間区間 [0,1] を 20 等分のメッシュに分割し、時間間隔 [0,2] を 5 等分した t に関して、解を計算します。

```
x = linspace(0,1,20);
t = linspace(0,2,5);
```

6 PDE ソルバの適用 . 例題は、m = 0 をもつ pdepe、関数 pdex1pde, pdex1ic, pdex1bc をコールし、解を計算する位置を、x と t でメッシュを使って定義します。pdepe 関数は、3 次元配列 sol の中に数値解を出力します。ここで、sol(i,j,k) は、t(i) と x(j) で計算した解の k 番目の要素 u_k です。

```
m = 0;
sol = pdepe(m,@pdex1pde,@pdex1ic,@pdex1bc,x,t);
```

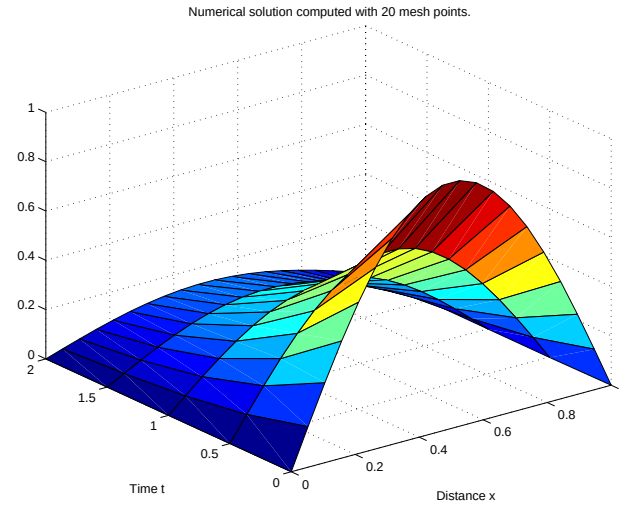
この例題は、@ を使って、pdex1pde, pdex1ic, pdex1bc に、pdepe への関数ハンドルとして、渡します。

注意 関数ハンドルに関する情報は、function_handle (@), func2str, str2func のリファレンスページ、および MATLAB ドキュメントの "Programming with MATLAB" の "M-Files Programming" を参照してください。

7 結果の視覚化 . 結果を表示することで、例題を終了します。

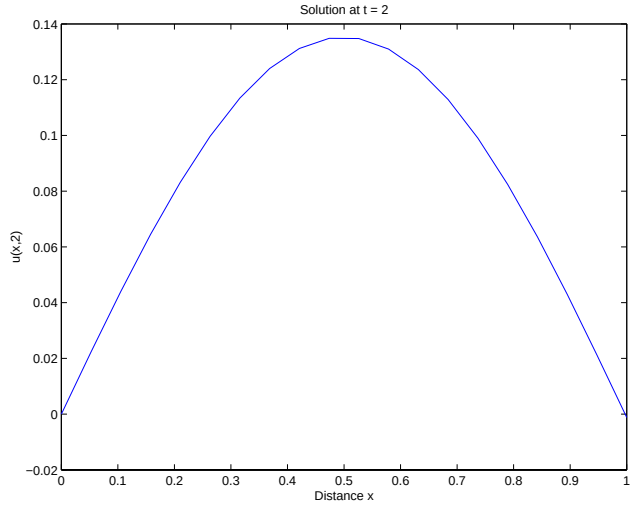
- 最初の解の成分を抽出し、表示します。この例題で、解 u は一成分のみをもっています。しかし、表示目的のために、3 次元配列からそれを抽出してみます。サーフェスプロットは、解の挙動を示しています。

```
u = sol(:,:,1);

surf(x,t,u)
title('Numerical solution computed with 20 mesh points')
xlabel('Distance x')
ylabel('Time t')
```



- つぎの図は、t_f での解のプロファイルと、t の最終値を示しています。この例題で、t_f = t = 2 になります。詳細は、15-85 ページの " 指定した点での解の計算 " を参照してください。

```
figure
plot(x,u(end,:))
title('Solution at t = 2')
xlabel('Distance x')
ylabel('u(x,2)')
```



指定した点での解の計算

上のように解を得て、プロットした後、t の特別な値での解のプロファイルや、特別な点 x での解の時間変化に興味があります。(ステップ 7 の中で抽出した解の)k 番目の列 u(:,k) は、x(k) での解の時系列を含んでいます。j 番目の行 u(j,:) は、t(j) での解のプロファイルを含んでいます。

ベクトル x と u(j,:) と補助関数 pdeval を使って、点 xout の任意の組で、解 u と微係数$\partial u/\partial x$を計算します。

```
[uout,DuoutDx] = pdeval(m,x,u(j,:),xout)
```

例題 pdex3 は、xout = 0 で、解の微係数を pdeval を使って計算します。詳細は、pdeval を参照してください。

PDE ソルバ性能の改良

MATLAB PDE ソルバの中のデフォルトの積分プロパティは、一般的な問題を取り扱うことができるように選択されています。いくつかの場合、これらのデフォルト値を変更することで、ソルバの性能を改良することができます。これを行うには、Options 構造体の中の一つ、または、複数のプロパティ値を変更して、pdepe と共に使います。

```
sol = pdepe(m,pdefun,icfun,bcfun,xmesh,tspan,options)
```

odeset を使って、Options 構造体を作成します。つぎのテーブルの中に示される ODE ソルバがもっているオプションは、pdepe で使用することができます。入力引数 options を使わないで得られるデフォルトは、一般的に満足のいくものです。構造体の作成や、プロパティの記述法については、15-15 ページの "ODE ソルバ性能の改良 " を参照してください。

PDE プロパティカテゴリ

プロパティカテゴリ	プロパティ名
誤差のコントロール	RelTol, AbsTol, NormControl
ステップサイズ	InitialStep, MaxStep

例題：電気力学問題

この例題は、偏微分方程式システムの解を示すものです。問題は、電気力学を使ったものです。これは、区間の両端に境界層があり、解は、短い t で急激に変化するものです。

PDEs は、つぎのようになります。

$$\frac{\partial u_1}{\partial t} = 0.024\,\frac{\partial^2 u_1}{\partial x^2} - F(u_1 - u_2)$$

$$\frac{\partial u_2}{\partial t} = 0.170\,\frac{\partial^2 u_2}{\partial x^2} + F(u_1 - u_2)$$

ここで、$F(y) = \exp(5.73y) - \exp(-11.46y)$ です。この方程式は、時間 $t \geq 0$ に対して、区間 $0 \leq x \leq 1$ で成り立ちます。

解 u は、初期条件

$$u_1(x, 0) \equiv 1$$

$$u_2(x, 0) \equiv 0$$

と境界条件

$$\frac{\partial u_1}{\partial x}(0, t) \equiv 0$$

$$u_2(0, t) \equiv 0$$

$$u_1(1, t) \equiv 1$$

$$\frac{\partial u_2}{\partial x}(1, t) \equiv 0$$

を満足します。

注意　デモ pdex4 は、この例題をすべてコード化されています。デモは、単一の M-ファイルの中に、必要なすべてのサブ関数を設定しています。デモを実行するには、コマンドラインで、pdex4 と入力してください。詳細は、15-78 ページの "PDE ソルバの基本的なシンタックス" と 15-80 ページの "PDE 問題の表現" を参照してください。

---

1 PDE の書き換え． pdepe で期待されている型で、方程式はつぎのようになります。

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} .* \frac{\partial}{\partial t}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \frac{\partial}{\partial x}\begin{bmatrix} 0.024(\partial u_1/\partial x) \\ 0.170(\partial u_2/\partial x) \end{bmatrix} + \begin{bmatrix} -F(u_1 - u_2) \\ F(u_1 - u_2) \end{bmatrix}$$

u の偏微分係数の境界条件は、流量の項で記述される必要があります。pdepe で期待される型で、左境界条件

$$\begin{bmatrix} 0 \\ u_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} .* \begin{bmatrix} 0.024(\partial u_1/\partial x) \\ 0.170(\partial u_2/\partial x) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

と以下の右境界条件を表わします。

$$\begin{bmatrix} u_1 - 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} .* \begin{bmatrix} 0.024(\partial u_1/\partial x) \\ 0.170(\partial u_2/\partial x) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

2 MATLAB の中での PDE のコード化． 上で示した型で、PDE を書き直した後、pdepe で使用可能なような関数として、コード化できます。関数は、つぎの型をしています。

```
[c,f,s] = pdefun(x,t,u,dudx)
```

ここで、c，f，s は、式 15-3 の中で、c，f，s に対応しています。

```
function [c,f,s] = pdex4pde(x,t,u,DuDx)
c = [1; 1];
f = [0.024; 0.17] .* DuDx;
y = u(1) - u(2);
F = exp(5.73*y)-exp(-11.47*y);
s = [-F; F];
```

3 初期条件関数をコード化． 初期条件関数は、つぎの型である必要があります。

```
u = icfun(x)
```

下のコードは、MATLAB 関数 pdex4ic の中の初期条件を表現しています。

```
function u0 = pdex4ic(x);
u0 = [1; 0];
```

4 境界条件関数をコード化． 境界条件関数は、つぎの型である必要があります。

```
[pl,ql,pr,qr] = bcfun(xl,ul,xr,ur,t)
```

下のコードは、MATLAB 関数 pdex4bc の中で境界条件の要素 $p(x,t,u)$ と $q(x,t)$ (式 15-5) を計算します。

```
function [pl,ql,pr,qr] = pdex4bc(xl,ul,xr,ur,t)
pl = [0; ul(2)];
ql = [1; 0];
pr = [ur(1)-1; 0];
qr = [0; 1];
```

5 解に対するメッシュ点を選択． 解は、短い t で急激に変化します。プログラムは、この急激な変化に追従するように時間でのステップサイズを選択します。しかし、プロットの中でも、この挙動を見るには、出力時間をそれに応じて選択する必要があります。[0,1] の両端で、解に境界層があり、メッシュ点は、この急激な変化に追従するように配置する必要があります。解の挙動を表すメッシュを選択するために、いくつかの実験をする必要があります。

```
x = [0 0.005 0.01 0.05 0.1 0.2 0.5 0.7 0.9 0.95 0.99 0.995 1];
```

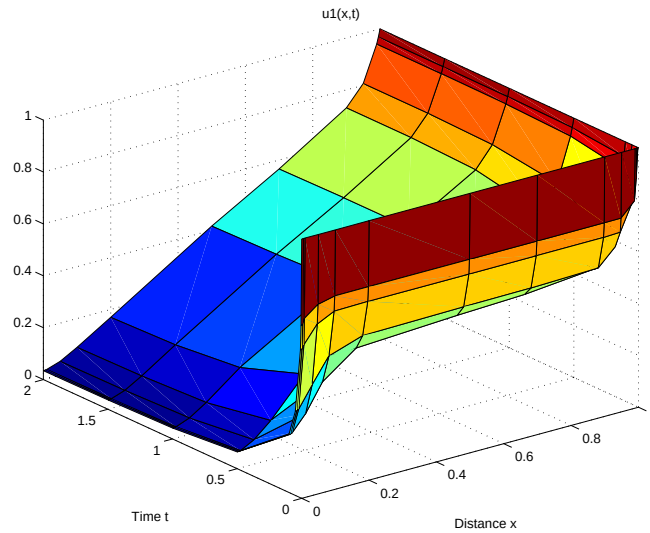
```
t = [0 0.005 0.01 0.05 0.1 0.5 1 1.5 2];
```

6 PDE ソルバの適用． 例題は、m = 0 をもつ pdepe、関数 pdex4pde, pdex4ic, pdex4bc をコールし、pdepe が解を計算する位置を、x と t でメッシュを使って定義します。pdepe 関数は、3 次元配列 sol の中に数値解を出力します。ここで、sol(i,j,k) は、t(i) と x(j) で計算した解の k 番目の要素 u_k です。

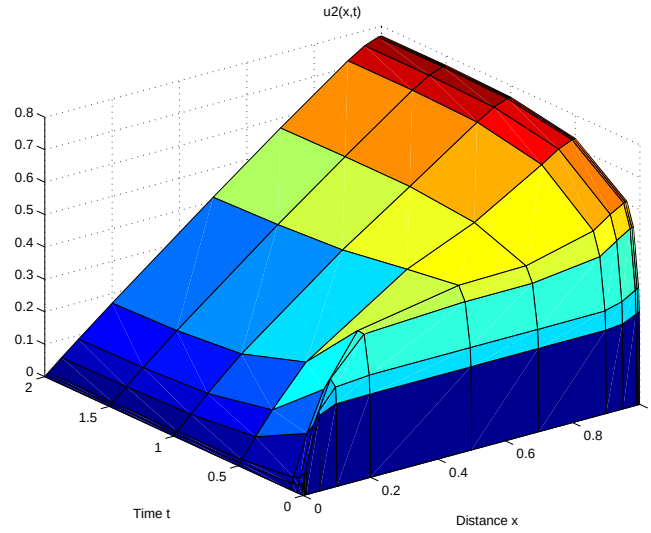
```
m = 0;
sol = pdepe(m,@pdex4pde,@pdex4ic,@pdex4bc,x,t);
```

7 結果の視覚化． サーフェスプロットが、解要素の挙動を表示します。

```
u1 = sol(:,:,1);
u2 = sol(:,:,2);

figure
surf(x,t,u1)
title('u1(x,t)')
xlabel('Distance x')
ylabel('Time t')
```



```
figure
surf(x,t,u2)
title('u2(x,t)')
xlabel('Distance x')
ylabel('Time t')
```



参考文献

[1] Ascher, U., R. Mattheij, and R. Russell, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, SIAM, Philadelphia, PA, 1995, p. 372.

[2] Hairer, E., and G. Wanner, *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*, Springer-Verlag, Berlin, 1991, pp. 5-8.

[3] Hindmarsh, A. C., ″LSODE and LSODI, Two New Initial Value Ordinary Differential Equation Solvers,″ *SIGNUM Newsletter*, Vol. 15, 1980, pp. 10-11.

[4] Hindmarsh, A. C., and G. D. Byrne, ″Applications of EPISODE: An Experimental Package for the Integration of Ordinary Differential Equations,″ *Numerical Methods for Differential Systems*, L. Lapidus and W. E. Schiesser eds., Academic Press, Orlando, FL, 1976, pp 147-166.

[5] Shampine, L. F., *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall Mathematics, 1994.

[6] Shampine, L. F., and M. K. Gordon, *Computer Solution of Ordinary Differential Equations*, W.H. Freeman & Co., 1975.

[7] Skeel, R. D. and M. Berzins, ″A Method for the Spatial Discretization of Parabolic Equations in One Space Variable,″ *SIAM Journal on Scientific and Statistical Computing*, Vol. 11, 1990, pp.1-32.

16

スパース行列

関数のまとめ 16-3

はじめに 16-6
スパース行列ストレージ 16-6
一般的なストレージ情報 16-7
スパース行列の作成 16-7
MATLAB の外部からスパース行列を読み込む 16-12

スパース行列の可視化 16-13
非ゼロ要素に関する情報 16-13
スパース行列のグラフィカルな表示 16-15
関数 find とスパース行列 16-16

例題：隣接行列とグラフ 16-17
隣接行列の紹介 16-17
隣接行列を使ったグラフ表示 16-18
Bucky ボール 16-18
空気の流れに関するモデル 16-23

スパース行列の演算 16-25
計算について 16-25
標準数学演算 16-25
置換と並べ替え 16-26
因子分解 16-30
連立方程式 16-36
固有値と特異値 16-38

参考文献 16-41

MATLAB は、非ゼロの部分が、全体から見た場合少ない*スパース行列*をサポートしています。この特徴は、行列が使用するストレージの節約と計算時間の節約に効果を発揮します。

この章は、MATLAB の中でのスパース行列の作成法、特殊な数学演算、一般的な数学演算の使い方を説明します。つぎの事柄が含まれています。

関数のまとめ
スパース行列関数のまとめ

はじめに
MATLAB の中でのスパース行列の紹介

スパース行列の可視化
スパース行列に関する情報やグラフック情報の取得法

例題：隣接行列とグラフィックス
例題は、スパース行列をデモンストレーションするために、隣接行列を使います。

スパース行列演算
スパース行列固有の演算を行う関数についての説明

参考文献
この章で記述されている概念を説明している文献

関数のまとめ

スパース行列関数は、MATLABのtoolboxディレクトリの中のディレクトリsparfunに格納されています。

関数のまとめ

カテゴリ	関数	詳細
基本スパース行列	speye	スパース単位行列
	sprand	一様分布するスパースランダム行列
	sprandn	正規分布するスパースランダム行列
	sprandsym	ランダムスパース対称行列
	spdiags	対角要素から作成されるスパース行列
フル行列からスパース行列への変換	sparse	スパース行列の作成
	full	スパース行列をフル行列に変換
	find	非ゼロ要素のインデックスの検出
	spconvert	外部書式のスパース行列を内部書式へ変換
スパース行列を使った処理	nnz	非ゼロ行列要素の数
	nonzeros	非ゼロ要素の表示
	nzmax	非ゼロ要素行列に割り当てられるストレージの総量
	spones	非ゼロのスパース行列要素を1で置き換え
	spalloc	スパース行列に対するメモリの割り当て
	issparse	スパース行列に対して真
	spfun	非ゼロ行列要素に関数を適用
	spy	スパース性の構造表示

関数のまとめ（続き）

カテゴリ	関数	詳細
グラフ理論	gplot	" グラフ理論 " 的なグラフ表示
	etree	Elimination tree.
	etreeplot	elimination tree のプロット
	treelayout	Lay out tree or forest.
	treeplot	Plot picture of tree.
並べ替えアルゴリズム	colamd	列の近似最小度合いでの並び替え
	colmmd	列の最小度合いでの並び替え
	symamd	対称な近似最小度合いでの並び替え
	symmmd	対称な最小度合いでの並び替え
	symrcm	対称逆 Cuthill-McKee での並び替え
	colperm	非ゼロ要素数による列の並び替え
	randperm	ランダム置換ベクトル
	dmperm	Dulmage-Mendelsohn 分解
線形代数	eigs	大きい順に任意の個数の固有値取り出し
	svds	大きい順に任意の個数の特異値取り出し
	luinc	不完全 LU 分解
	cholinc	不完全 Cholesky 分解
	normest	行列の 2 ノルムの計算
	condest	1 ノルム条件数の計算
	sprank	構造化ランク

関数のまとめ（続き）

カテゴリ	関数	詳細
線形方程式（繰り返し法）	bicg	双共役勾配法
	bicgstab	双共役勾配安定法
	cgs	共役勾配二乗法
	gmres	一般化最小残差法
	lsqr	正規方程式での共役勾配法での LSQR 実現
	minres	最小残差法
	pcg	条件付き共役勾配法
	qmr	準最小残差法
	symmlq	対称 LQ 法
Miscellaneous	spaugment	最小二乗拡大システムの作成
	spparms	スパース行列ルーチンに対するパラメータ設定
	symbfact	シンボリックな因子解析

はじめに

スパース行列は、大部分がゼロ要素である行列の特別なクラスです。このプロパティは、MATLAB では、

- 行列要素の非ゼロ要素のみをインデックスと共にストアします。
- ゼロ要素への演算を省略して計算時間を軽減します。

この節では、つぎの事柄について説明します。

- スパース行列ストレージ
- 一般的なストレージ情報
- スパース行列の作成
- スパース行列の読み込み

スパース行列ストレージ

フル行列の場合、MATLAB はすべての行列要素を内部にストアします。ゼロの要素の部分も、非ゼロ要素の部分と同じストレージ容量を必要とします。しかし、スパース行列の場合、MATLABは非ゼロ要素とそのインデックスのみをストアします。ゼロ要素が含まれる割合の高い大きな行列では、この方法はデータストレージ容量に対するメモリ総量をかなり少なくします。

MATLAB は、実要素をもつスパース行列を内部にストアするため 3 つの配列をもっています。nnz 個の非ゼロ要素をもつ m 行 n 列のスパース行列を考えましょう。

- 最初の配列は、配列の非ゼロ要素を浮動小数点フォーマットで含んだものです。この配列の長さは、nnz の非ゼロ要素の数になります。
- 2 番目の配列は、非ゼロ要素に対応する整数行インデックスを含んだものです。この配列も nnz の長さと同じです。
- 3 番目の配列は、他の配列の中の各列の最初を示す n 個のポインタとそれらの配列の最後を示すポインタを含むものです。3 番目の配列の長さは、n+1 です。

この行列は、nnz 個の浮動小数点数と nnz+n+1 個の整数をストアするストレージを必要とします。浮動小数点数は 8 バイト、整数は 4 バイトで、スパース行列をストアするためには、

```
8*nzmax + 4*(nzmax+n+1)
```

のストレージが必要です。複素数要素をもつスパース行列もまた取り扱えます。この場合、MATLAB は、非ゼロ要素の虚数部をストアするため nnz の長さの 4 番目

の配列を必要とします。要素は、実数部または虚数部のどちらかが非ゼロであれば、要素としては非ゼロと考えます。

一般的なストレージ情報

whos コマンドを使って、行列のサイズ、クラスを含む行列ストレージに関する高水準の情報を得ることができます。たとえば、つぎの whos リストは、同じ行列をスパースで表したものと、フルで表したものとの情報を示しています。

```
whos
  Name            Size          Bytes  Class

  M_full       1100x1100      9680000  double array
  M_sparse     1100x1100         4404  sparse array

Grand total is 1210000 elements using 9684404 bytes
```

使用されるバイト数は、ゼロ要素の部分を含んでいないために、スパースの場合のほうが、かなり少なくなっています。この場合。スパース行列の密度は、4404/9680000 で、約 0.00045% です。

スパース行列の作成

MATLAB は、自動的にスパース行列を作る機能はまだ用意していません。そのため、行列でゼロ要素の占める割合が大きいかどうかユーザが判断して、スパース手法を使うかどうかを決めなければなりません。

行列の *密度* は、行列要素の総数で非ゼロ要素数を割ったもので表します。非常に低い密度の行列が、スパースフォーマットの使用の良い基準になります。

フル行列からスパース行列への変換

フル行列をスパースストレージに変換するには、関数 sparse を一つの引数 (フル行列名) と共に使ってください。

```
S = sparse(A)
```

たとえば、

```
A = [ 0   0   0   5
      0   2   0   0
      1   3   0   0
      0   0   4   0];

S = sparse(A)
```

により

```
 S =

   (3,1)        1
   (2,2)        2
   (3,2)        3
   (4,3)        4
   (1,4)        5
```

を出力します。プリントされる出力は、S の非ゼロ要素とその行と列のインデックスをリストするものです。要素は、内部データ構造を反映して、その列に従ってソートされます。

行列の次数があまり大きくなければ、スパース行列をフルストレージに関数 full を使って変換できます。例えば、A = full(S) は、例題の行列変換を逆変換します。

フル行列をスパースストレージに変換することは、スパース行列作成法としては、あまり使われるものではありません。行列の大きさがフルストレージとして十分小さいものならば、スパースストレージへの変換はあまり意味をもちません。

直接的にスパース行列を作成

関数 sparse を 5 つの引数と共に使って、非ゼロ要素のリストからスパース行列を作ることができます。

```
S = sparse(i,j,s,m,n)
```

i と j は、行列の非ゼロ要素の行と列のインデックスを表すベクトルです。s は、(i,j) により設定されるインデックスにより示される非ゼロ値のベクトルです。m は、結果の行列に対する行の次元数で、n は列の次元数です。

前の例題の行列 S を直接入力してみましょう。

```
S = sparse([3 2 3 4 1],[1 2 2 3 4],[1 2 3 4 5],4,4)
```

```
S =

   (3,1)        1
   (2,2)        2
   (3,2)        3
   (4,3)        4
   (1,4)        5
```

sparse コマンドは、いろいろな使い方を行ないます。上の例題は、行列の中の非ゼロ要素の最大数を length(s) に設定する型です。最大数よりも大きな値を 6 番目に設定すると、必要なストレージの量を変更しないで、後で非ゼロ要素を加えることができます。

例題： 2 階微分演算子

2 階微分演算子の行列表現は、スパース行列の良い例題です。これは、対角部に -2s をもち､1 つ上対角 (superdiagonal) と 1 つ下対角 (subdiagonal) に 1 をもつ三角対角行列です。これを作るのにいろいろな方法があります。ここで示すものも一つの方法です。.

```
D = sparse(1:n,1:n,-2*ones(1,n),n,n);
E = sparse(2:n,1:n-1,ones(1,n-1),n,n);
S = E+D+E'
```

n = 5 に対して、MATLAB は、つぎの出力を行ないます。

```
S =

   (1,1)       -2
   (2,1)        1
   (1,2)        1
   (2,2)       -2
   (3,2)        1
   (2,3)        1
   (3,3)       -2
   (4,3)        1
   (3,4)        1
   (4,4)       -2
   (5,4)        1
   (4,5)        1
   (5,5)       -2
```

さて、F = full(S) は、対応するフル行列を表示します。

```
F = full(S)

F =

    -2     1     0     0     0
     1    -2     1     0     0
     0     1    -2     1     0
     0     0     1    -2     1
     0     0     0     1    -2
```

行列の対角要素からスパース行列を作成

行列の対角要素をベースにスパース行列を作成するには、関数 spdiags を使って行ないます。このシンタックスは、つぎのとおりです。

```
S = spdiags(B,d,m,n)
```

対角要素が p である *m* 行 *n* 列の出力行列 S を作成するには、

- B は、min(*m*,*n*) 行 *p* 列の大きさの行列です。S の対角要素が B の列になります。
- d は、長さ p のベクトルで、その整数要素は、S の対角要素番号を設定するものです。

B の列 j の中の要素は、d の要素に対応する対角要素番号 j(負は下対角の番号、正は上対角番号) により設定される対角になります。

注意 B の列が置き換わる対角よりも大きい場合は、B の列の下部分から super-diagonal が得られ、sub-diagonal は B の列の上部分から得られます。

例えば、行列 B とベクトル d を考えましょう。

```
B = [ 41    11     0
      52    22     0
      63    33    13
      74    44    24 ];

d = [-3
      0
      2];
```

この行列を使って、7 行 4 列のスパース行列 A を作成しましょう。

```
A = spdiags(B,d,7,4)

A =

   (1,1)        11
   (4,1)        41
   (2,2)        22
   (5,2)        52
   (1,3)        13
   (3,3)        33
   (6,3)        63
   (2,4)        24
   (4,4)        44
   (7,4)        74
```

これに対応するフル行列 A は、つぎのようになります。

```
full(A)

ans =

    11     0    13     0
     0    22     0    24
     0     0    33     0
    41     0     0    44
     0    52     0     0
     0     0    63     0
     0     0     0    74
```

spdiags は、スパース行列から対角要素を取り出すか、または、既に存在する対角要素を新しい値で置き換えることもできます。詳細は、help spdiags とタイプインしてください。

MATLAB の外部からスパース行列を読み込む

MATLAB 以外のツールで計算したスパース行列を読み込むことができます。関数 spconvert は、非ゼロ要素とそのインデックスを含んだ ASCII ファイルを転送するために load コマンドと共に使います。例えば、3 列のテキストファイル T.dat、すなわち、1 列目が行のインデックス、2 列目が列のインデックス、3 列目が非ゼ

ロの値からなるファイルを考えます。つぎのステートメントは、T.dat を MATLAB にロードして、スパース行列 S に変換します。

```
load T.dat
S = spconvert(T)
```

save と load コマンドは、MAT- ファイルにバイナリデータとしてストアしたスパース行列も処理することができます。

スパース行列の可視化

MATLAB は、スパース行列に関する情報を量的にも、グラフィカルにも表示する関数をいくつか用意しています。

この節は、つぎの事柄の情報について説明します。

- 非ゼロ要素に関する情報を得る
- スパース行列のグラフの可視化
- 非ゼロ要素のインデックスと値の検出

非ゼロ要素に関する情報

スパース行列の非ゼロ要素に関する高水準の情報を得る関数がいくつか用意されています。

- nnz は、スパース行列の中の非ゼロ要素の数を表します。
- nonzeros は、スパース行列の非ゼロ要素すべてを含む列ベクトルです。
- nzmax は、スパース行列の非ゼロ要素に対して割り当てられたストレージ総容量を表します。

これらのいくつかを試すため、Harwell-Boeing で記述されているものの一つであるスパース行列 west0479 をロードします。

```
load west0479
whos
  Name            Size          Bytes  Class

  west0479      479x479         24576  sparse array
```

この行列は、8 ステージの化学蒸留過程をモデル化したものです。

つぎのコマンドを試してください。

```
nnz(west0479)

ans =

        1887

format short e
west0479
```

```
west0479 =

  (25,1)       1.0000e+00
  (31,1)      -3.7648e-02
  (87,1)      -3.4424e-01
  (26,2)       1.0000e+00
  (31,2)      -2.4523e-02
  (88,2)      -3.7371e-01
  (27,3)       1.0000e+00
  (31,3)      -3.6613e-02
  (89,3)      -8.3694e-01
  (28,4)       1.3000e+02
     .
     .
     .

nonzeros(west0479);
    ans =

         1.0000e+00
        -3.7648e-02
        -3.4424e-01
         1.0000e+00
        -2.4523e-02
        -3.7371e-01
         1.0000e+00
        -3.6613e-02
        -8.3694e-01
         1.3000e+02
           .
           .
           .
```

注意　nonzeros の一覧表示を停止するには、**Ctrl+C** を使ってください。

最初、nnz はデフォルトにより、nzmax と同じ値であることに注意してください。これは、非ゼロ要素の数が、非ゼロに対して割り当てられるストレージの位置の

数と同じだからです。しかし、その後ある配列要素をゼロにしても MATLAB はメモリを開放しません。いくつかの行列要素の値をゼロにすると、nnz の値が変化しますが、nzmax の値は変化しません。

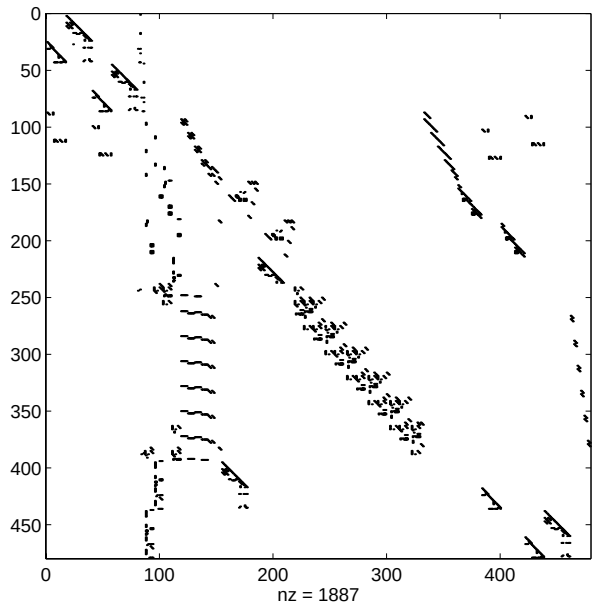
必要なら行列にいくつかの非ゼロ要素を加えることができます。しかし、nzmax のオリジナルな値によって制約を受けるものではありません。

スパース行列のグラフィカルな表示

スパース行列の中で非ゼロ要素の分布を表示するためにグラフィカル書式を使うと有効な場合があります。MATLAB の関数 spy は、スパース構造のテンプレート表示、すなわち、グラフ上の各々の点が非ゼロ配列要素の位置を表す、を作成します。

たとえば、つぎの図です。

```
spy(west0479)
```



関数 find とスパース行列

任意の行列 (フル、スパース行列) に対して、関数 find は、非ゼロ要素のインデックスと値を出力します。構文は、つぎの通りです。

```
[i,j,s] = find(S)
```

find は、ベクトル i に非ゼロ値の行インデックス、ベクトル j に列インデックス、ベクトル s にその値を出力します。つぎに使う例題は、スパース行列の中の非ゼロ要素のインデックスと値を find を使って求めるものです。関数 sparse は、find 出力と新しい行列のサイズの設定値を使って、行列を再作成します。

```
[i,j,s] = find(S)
[m,n] = size(S)
S = sparse(i,j,s,m,n)
```

例題： 隣接行列とグラフ

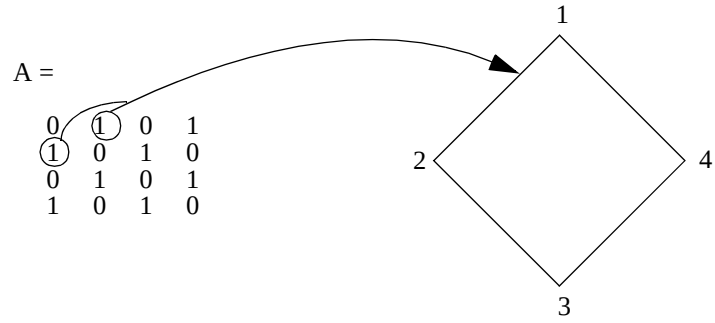
この節では、つぎの事柄の説明を行ないます。

- 隣接行列の紹介
- 隣接行列を gplot を使ってのグラフ化の紹介
- fill-in と distance を示すために、spy プロットの使用に関する情報を含んだ Bucky ボールの例題
- 空気の流れのモデル

隣接行列の紹介

*グラフ*という言葉の正式は数学的定義は、お互いを特別な方法を使って接続した点またはノードの集合です。例えば、ノードとして種々の産業、接続するものとして直接的な経済的な結び付きを示す経済モデルのようなものです。コンピュータソフトウエア産業は、コンピュータハードウエア産業、言い換えれば、半導体産業等と関連しています。

グラフのこの定義は、行列表現を利用することができます。ノード i とノード j が接続されているなら、*方向性をもたないグラフの隣接行列*の (i,j) 番目と (j,i) 番目の要素は 1 で、接続されていなければ 0 になります。例えば、ダイアモンド型のグラフを表す近接行列は、つぎのようになります。



多くのグラフは、1 つのノードに対して比較的結合数が少ないので、ほとんどの隣接行列はスパースになります。非ゼロ要素の実際の位置は、どのようにノードに番号を付けるかに依ります。番号の付け方を変更すると、近接行列の行と列の

置換が生じます。これは、スパース行列計算で時間と必要なストレージ共に重要な影響を与えます。

隣接行列を使ったグラフ表示

MATLAB の関数 gplot は、隣接行列と座標の関連した配列をベースにグラフを作成します。gplot を試すため、ダイヤモンドの例の隣接行列を作成しましょう。

```
A = [0 1 0 1; 1 0 1 0; 0 1 0 1; 1 0 1 0];
```

gplot の座標配列の列は、対応するノードに対する直交座標を含んでいます。この例題では、つぎのように入力します。

```
xy = [1 3; 2 1; 3 3; 2 5];
```

ここでは、最初のノードが (1,3)、2 番目は (2,1)、3 番目は (3,3)、4 番目は (2,5) の位置にあることを意味しています。結果のグラフを見るためには、つぎのように入力します。

```
gplot(A,xy)
```

Bucky ボール

グラフ解析に対する興味あるものとして、*Bucky ボール*があります。これは、球面上に 60 個の点が分布し、かつ各点の最近傍点までの距離がすべて等しい関係を保っているものです。各々の点は、必ず 3 つの隣り合った点をもっています。Bucky ボールは、つぎのように異なる 4 つの物理モデルに使われています。

- Buckminster Fuller により発表された測地線ドーム
- C_{60} 分子、ほぼ球面型をした 60 個の原子をもつ炭素の組成物 n
- 幾何学の中ので、切頭の 20 面体のもの
- スポーツでのサッカーボール

Bucky ボールの隣接行列は 60 行 60 列の対称行列 B です。B は、各行、各列各々 3 つの非ゼロ要素、総数 180 個の非ゼロ要素をもっています。この行列は、前述の物理オブジェクトと関連付られた重要なアプリケーションです。例えば、B の固有値は、C_{60} の化学的な性質を調べるために使います。

Bucky ボールの隣接行列を得るには、つぎのように入力します。

```
B = bucky;
```

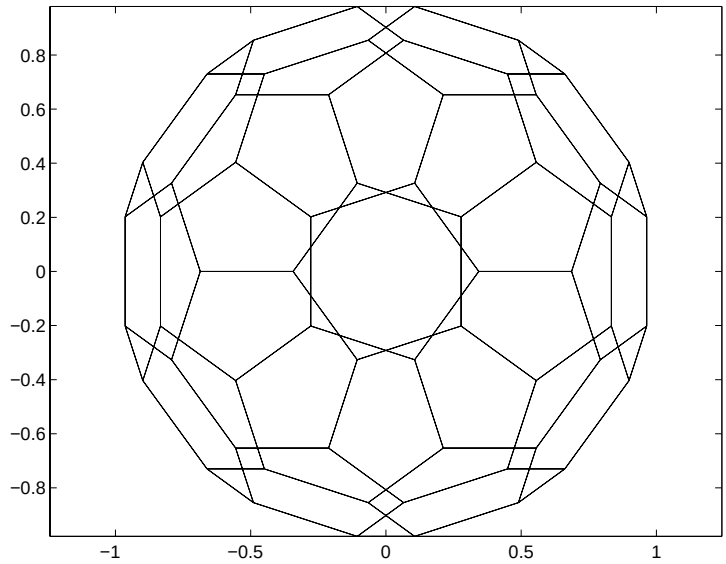
60 次で、5% の密度をもって、この行列はスパース手法を必ずしも必要としていません。しかし、これは、興味ある例題を与えます。

つぎのステートメントで Bucky ボールの座標を得ます。

```
[B,v] = bucky;
```

このステートメントは、3 空間内で単位平面上に等分布する 60 点の *xyz* 座標からなる v を作成します。関数 gplot は、これらの点を使って、Bucky ボールグラフをプロットします。

```
gplot(B,v)
axis equal
```

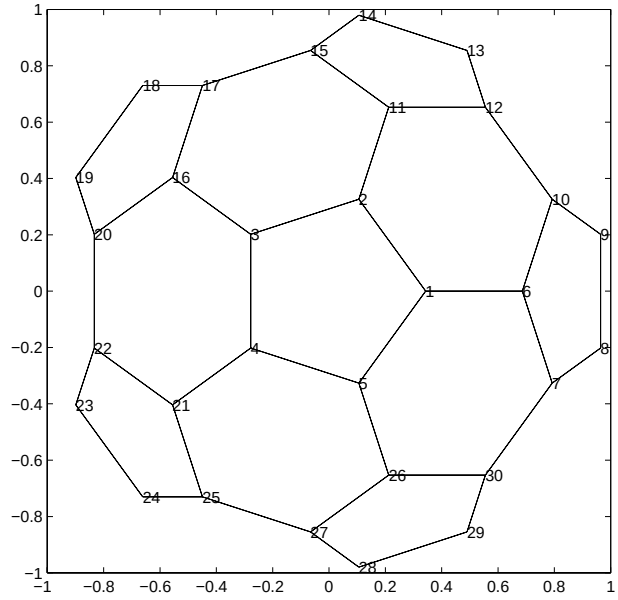


ここでは、Bucky ボールの中のノードをどのように番号付けしているのかわかりません。そのため、結果作成される隣接行列はグラフの表面や結合された対称性が明らかになりません。v で使われている番号付け法は、ボールの構造内の固有の多角形をベースにしています。

一つの多角形の頂点が 1 から 5 で番号付けされ、隣接多角形の頂点は 6 から 10、等々になります。つぎの図は、半分のノード (半平面) に付けられた番号を示し

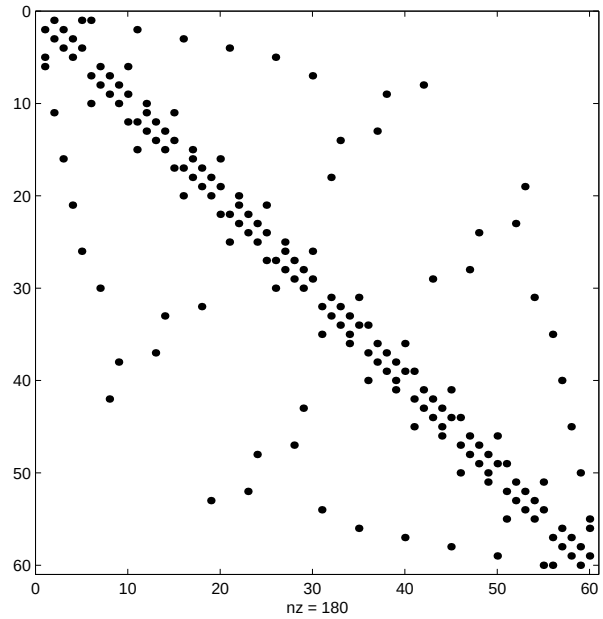
ています。すなわち、残りの半平面の番号付けは、赤道に関して反転して得られます。gplot の使用は、半分のノードを示すグラフを作成します。ユーザは、for ループを使って、ノード番号を加えることもできます。

```
k = 1:30;
gplot(B(k,k),v);
axis square
for j = 1:30, text(v(j,1),v(j,2), int2str(j)); end
```



Bucky ボールの隣接行列の中の非ゼロ要素の位置のテンプレートを見るため、関数 spy を使います。

```
spy(B)
```

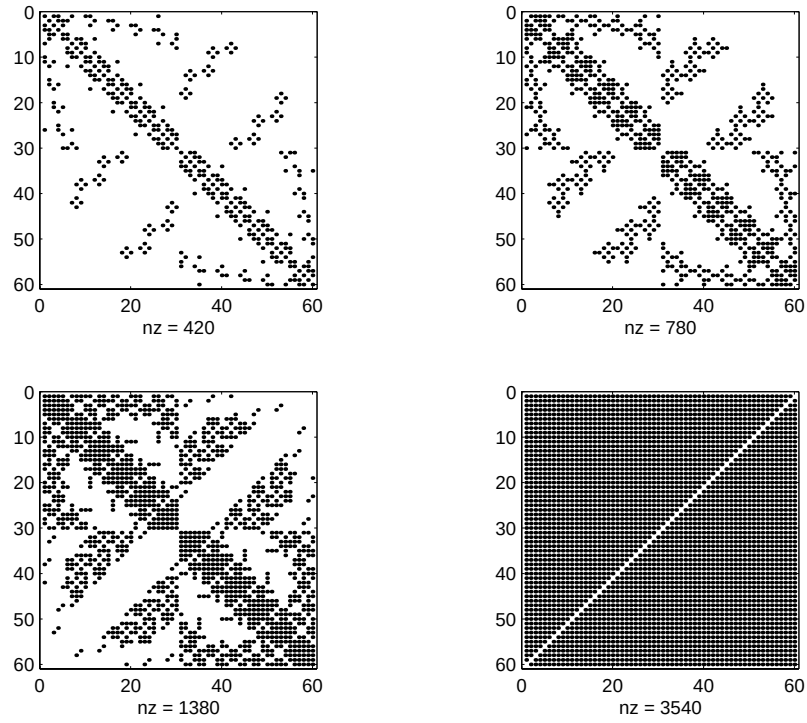


このモデルが使うノード番号付け法は、構造の中の 12 多角形に対応する 5 つの要素をもった 12 グループに分けられる spy プロットを作成します。各々のノードは、自分自身の多角形の中で他の 2 つのノードと別の多角形の中で 1 つのノードと結合しています。一つの多角形の中で、ノードは連続する番号が付けられているので、B の 1 つ上の対角要素と 1 つ下の対角要素のほとんどのものは非ゼロになります。そして、赤道に関しての番号付けの対称性は、反対角線に対して spy プロットが対称性を示すことで明らかになります。

スパース行列のグラフと特性

B の行列のベキ乗の spy プロットは、スパース行列演算に関連した 2 つの重要な考え (full-in と distance) を示します。spy は、これらの考えを表示する手助けになります。

```
spy(B^2)
spy(B^3)
spy(B^4)
spy(B^8)
```

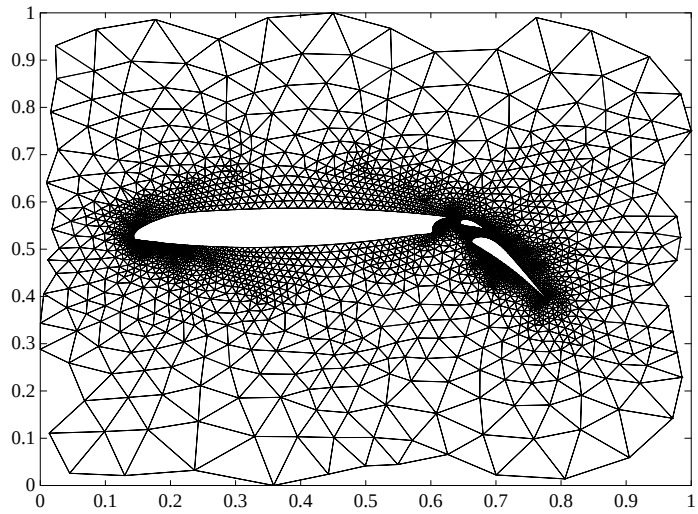


Fill-in は、行列の乗算のような演算で生じます。複数の行列の積は、通常オリジナルのものよりも多くの非ゼロ要素をもちます。それで、より大きなストレージを必要とします。p を増加させると、B^p は、ゼロ要素部分を非ゼロ要素で埋めて、spy(B^p) は、より密になります。

グラフの中の 2 つのノード間の *distance* は、一つのノードから他のノードを得るために必要となるグラフ上でのステップ数です。B の p 乗の spy プロットは、distance 離れたノードを示します。p を増加すると、p ステップでより多くのモードを得ることができます。Bucky ボールに対して、B^8 は、ほぼ完全なフルの型になります。逆対角要素の線対称の部分のみがゼロになり、これより 8 回のステップで球面上の反対側の球面を除いて、あるノードから他のどのノードでも得ることができるということを示しています。

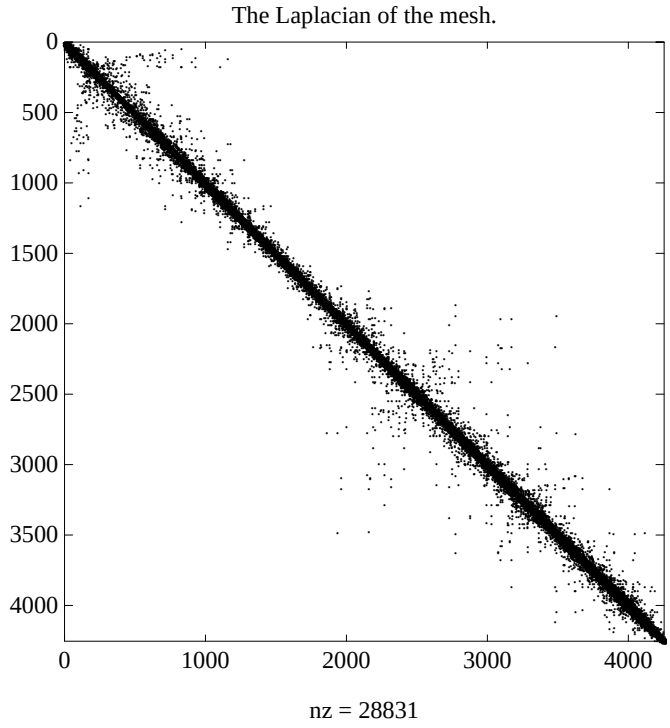
空気の流れに関するモデル

NASA の Research Institute で行われた 2 つの繋げられたフラップをもった飛行機の翼での流れをモデル化する計算が行われました。



2 次元モデルで、翼とフラップの断面図の周りを三角形でグリッド化します。偏微分方程式は非線形で、静水圧や 2 成分の速度を含んだ、いくつかの未知数を含んでいます。非線形繰り返しの各々のステップは、スパース線形システム方程式の解を必要とします。グリッド点の幾何学的な位置と結合が共に既知ならば、gplot 関数は上に示すグラフを作成します。

この例題の中で、4253 グリッド点が存在し、その各々は、3 から 9 個のグリッド点と結合しています。行列の中に 28831 個の非ゼロ要素があり、密度は 0.0016 です。この spy プロットは、ノードの番号付けがはっきりした帯構造を与えることを示しています。


The Laplacian of the mesh.
0
500
1000
1500
2000
2500
3000
3500
4000
0
1000
2000
3000
4000
nz = 28831

スパース行列の演算

MATLAB の標準数学関数の大部分は、フル行列同様、スパース行列にも機能します。加えて、MATLAB はスパース行列に特化した演算を行なういくつかの関数も提供します。この節では、つぎのことを議論します。

- 計算について
- 標準数学演算
- 置換と並べ替え
- 因子分解
- 連立線形方程式
- 固有値と特異値

計算について

スパース演算の計算の複雑さは、行列の中の非ゼロ要素の数 nnz に比例します。計算の複雑さは、行列の行サイズ m と列サイズ n に線形に依存します。しかし、ゼロ要素数と非ゼロ要素数の和、すなわち、全要素数 m*n には独立です。

スパース線形方程式の解のようなかなり込み入った演算の複雑さは、前の節で議論した ordering や fill-in のようなファクタを含んでいます。しかし、一般に、スパース行列演算に必要な計算時間は、非ゼロの量に対して働きかける演算回数に比例します。

標準数学演算

スパース行列は、つぎのルールに乗っ取って計算を進めます。

- 行列を入力して、スカラまたはベクトルを出力する関数は、必ずフルストレージ書式で出力を作成します。例えば、関数 size は、入力がフルまたはスパースどちらでもフルベクトルを必ず出力します。
- スカラまたはベクトルを入力して、行列を出力する zeros, ones, rand, eye のような関数は、必ずフルの結果を出力します。これは、予期できないスパース性を取り込まないようにするために必要です。zeros(m,n) のスパース行列版は、単に sparse(m,n) です。rand と eye のスパース行列版はそれぞれ sprand と speye です。関数 ones のスパース行列版はありません。
- 行列を入力として、行列またはベクトルを出力する単項関数は、オペランドのストレージクラスを保存します。S がスパース行列なら、chol(S) もまたスパース行列で、diag(S) はスパースベクトルです。max や sum のような列方向に機

能する関数は、これらのベクトルがすべて非ゼロとしてもスパースベクトルを出力します。このルールに従わない重要なものは、関数 sparse と full です。

- バイナリ演算子は、両方のオペランドがスパースならばスパースを出力し、両方のオペランドがフルならばフルを出力します。フルとスパースがミックスされたものは、演算がスパース性を保存しない限り、結果はフルになります。S がスパースで、F がフルならば、S+F,S*F,F\S はフルに、一方 S.*F,S&F はスパースになります。多くの場合、結果は、行列にゼロ要素がほとんど含まれないけれどもスパースになります。
- 関数 cat かまたはスパース括弧のどちらかを使って行列の置換を行なうと、混合されたオペランドに対してスパース性をもつ結果を作成します。
- 右辺側のサブ行列インデックス設定法は、オペランドのストレージフォーマットを保存します。T = S(i,j) は、S がスパースで、i,j がスカラまたはベクトルのどちらかであると、スパースな結果を出力します。左辺のサブ行列のインデックス設定法は、左辺の行列のストレージフォーマットを変更しません。

置換と並べ替え

スパース行列 S の行と列の置換は、二つの方法により表現されます。

- 置換行列 P が、P*S のように S の行に、または、S*P' のように S の列に働きます。
- 置換ベクトル p、これは、1:n の置換を含んだフルベクトルですが、これが S(p,:) として S の行に、または、S(p,:) として列に機能します。

例えば、ステートメント

```
p = [1 3 4 2 5]
I = eye(5,5);
P = I(p,:);
e = ones(4,1);
S = diag(11:11:55) + diag(e,1) + diag(e,-1)
```

は、つぎの結果を出力します。

```
p =

     1     3     4     2     5

P =

     1     0     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     1     0     0     0
     0     0     0     0     1

S =

    11     1     0     0     0
     1    22     1     0     0
     0     1    33     1     0
     0     0     1    44     1
     0     0     0     1    55
```

置換ベクトル p と置換行列 P を使って、種々の置換を行なってみましょう。例えば、ステートメント S(p,:) と P*S は、つぎの結果を出力します。

```
ans =

    11     1     0     0     0
     0     1    33     1     0
     0     0     1    44     1
     1    22     1     0     0
     0     0     0     1    55
```

同様に、S(:,p) と S*P' は、つぎの結果を出力します。

```
ans =

    11     0     0     1     0
     1     1     0    22     0
     0    33     1     1     0
     0     1    44     0     1
     0     0     1     0    55
```

P がスパース行列ならば、二つの表現は n に比例したストレージを使い、時間的には nnz(S) に比例して、S のどちら側からでも適用できます。ベクトル表現は、わずかですがコンパクトで、効率的です。それで、種々のスパース行列置換ルーチンは、すべて LU(三角) 分解でピボット置換 (これは、MATLAB の古いバージョンと一致する行列を出力) を例外として、フル行ベクトルを出力します。

2 つの表現間の変換を行なうため、I = speye(n) を適切なサイズの対角が 1 の行列とします。そして、つぎの計算を行ないます。

```
P = I(p,:)
P' = I(:,p)
p = (1:n)*P'
p = (P*(1:n)')'
```

P の逆は、R = P' です。p の逆を r(p) = 1:n で計算することができます。

```
r(p) = 1:5

r =

     1     4     2     3     5
```

スパース性に対する並べ替え

行列の列を並べ替えることは、LU または QR 型のスパースにします。行と列の並べ替えは、Cholesky 型のスパースになります。最も簡単な並べ替えは、非ゼロのカウントにより列をソートすることです。これは、特に、行または列の非ゼロカウントの中で大きな変化があるとき、全く不規則な構造をもつ行列に対して良い並べ替えになります。

関数 p = colperm(S) が、この列カウント置換を計算します。M- ファイル colperm は、たった一つのラインからなるものです。

```
[ignore,p] = sort(full(sum(spones(S))));
```

このラインは、つぎのステップで処理します。

1. 関数spones は、S の中のすべての非ゼロ要素の位置に 1 をもつスパース行列を作ります。
2. 関数 sum は、各々の列の中で非ゼロのカウントを含むベクトルを作成し、行列の列方向に和を計算します。
3. 関数 full は、このベクトルをフルストレージ書式に変換します。

4 関数 sort は、次数の高いものから低いものへと値をソートします。関数 sort からの 2 番目の出力は、このベクトルをソートする置換を設定するものです。

バンド幅を狭くするための並べ替え

逆 Cuthill-McKee 順列は、プロファイラまたは行列のバンド幅を狭くしようとするものです。しかし、この結果が生じる可能性の一番高い最も小さいバンド幅であることを保証していません。しかし、通常は、最小のバンド幅になっています。関数 symrcm(A) は、実際に、対称行列 A + A' の非ゼロ構造に機能します。しかし、非対称行列に対しても有効です。この並びは、1 次元問題やある意味で " 長くて薄い " 問題から生じる行列に対して有効です。"

最小次数の並び

グラフの中のノードの度合いは、隣接行列の対応する行の非ゼロ要素と同数で、このノードに接続している数です。最小次数アルゴリズムは、これらの次数が Gauss 消去法の中で、どのように変更されていくかをベースにした並べ替えを作成しています。列カウンタや逆 Cuthill-McKee を含んで、他のほとんどの並びよりもスパースファクタをもたらす複雑で強力なアルゴリズムです。

MATLAB は、対称行列に対して symamd と symmmd、非対称行列に対して colamd と colmmd の 2 つのバージョンを用意しています。colamd と colmmd は、AA'、または、A'A の型の対称行列に対して、機能します。

最小次数並べ替えアルゴリズムの最も時間を費やす部分は、個々のノードの度合いを保持することです。これを行う、4 つの関数は、すべて、正確な度合いよりも近似的なものを使っています。結果として、

- colmmd と symmmd を使って得られた因子分解は、正確な度合いを使って計算すると、より多くの非ゼロ要素をもつ傾向を示します。
- colamd と symamd は、colmmd や symmmd よりも、より厳密な近似を使います。これらは、正確な度合いを使って得られるものと同等の結果を得ます。
- colamd と symamd は、colmmd と symmmd より、それぞれ、速く処理します。これは、非常に大きい行列に対して、真実です。

関数 spparms を使うアルゴリズムの詳細に関連した種々のパラメータを変更することができます。

colmmd や symmmd で使われる詳細は、[4] を参照してください。colamd と symamd で使われるアルゴリズムの詳細は、[5] を参照してください。colamd と symamd で使われる近似度合いは [1] をベースにしています。

因子分解

この節では、スパース行列に対して 4 つの重要な因子分解手法を議論します。

- LU、または、三角分解
- Cholesky 分解
- QR または直交分解
- 不完全分解

LU 分解

S がスパース行列なら、つぎのステートメントは、3 つのスパース行列 L, U, P を出力します。ここで、この行列には、P*S = L*U の関係があります。

```
[L,U,P] = lu(S)
```

lu は部分ピボットを使った Gauss 消去法で因子を得ます。置換行列 P は、n 個の非ゼロ要素のみを含んでいます。密行列と共に、ステートメント [L,U] = lu(S) は、置換された単位下三角行列と上三角行列でそれらの積が S になる行列を出力します。出力引数なしでは、lu(S) は、ピボット情報なしに単一行列として、L と U を出力します。

スパース LU 分解は、スパースに対してピボットを行ないませんが、数値的に安定なものに対しては、ピボットを行ないます。実際に、時間とストレージの必要量が非常に異なるときでも、スパース分解 (ライン 1) とフルの因子分解 (ライン 2) は、同じ L, U を出力します。

```
[L,U] = lu(S)                  % スパースに対する因子分解

[L,U] = sparse(lu(full(S)))  % フルに対する因子分解
```

つぎのコマンドを使って、スパース行列の中のピボットをコントロールすることができます。

```
lu(S,thresh)
```

ここで、thresh は、[0,1] の中のピボットのスレッシュホールドです。ピボット操作は、ある列の中の対角要素が、その列の中の他のサブ対角要素の大きさの thresh 倍よりも小さい場合に生じます。thresh = 0 は、対角ピボット操作を強制します。thresh = 1 はデフォルトです。

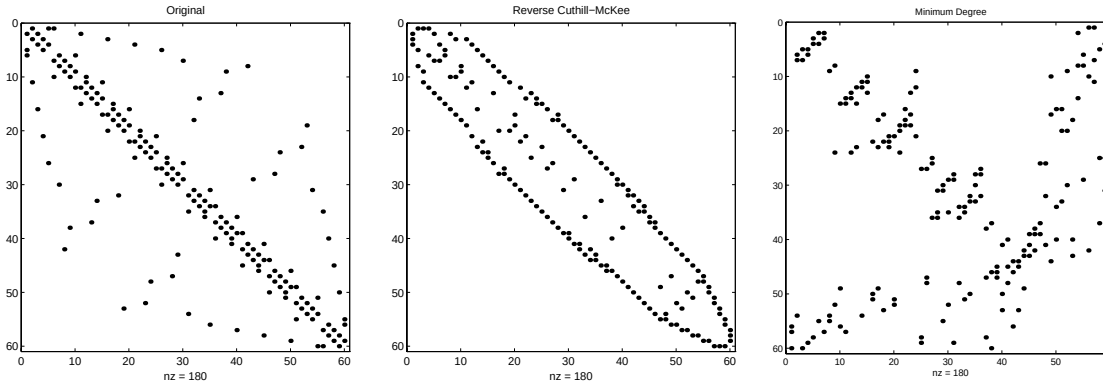
MATLAB は、因子分解の間にスパース L と U ファクタを保持するために必要なメモリを自動的に割り当てます。MATLAB は、必要なメモリを決めたり、より進んだデータ構造を設定するためにシンボリックな LU プリ因子分解を使いません。

並び替えと因子分解． ユーザが、m または colamd から、fill-in 状態を低減する良い列置換 p を得ているなら、lu(S(:,p)) を計算することは、lu(S) を計算するより時間とストレージともに少なくなります。2 つの置換は、対称逆 Cuthill-McKee 並びと対称最小次元順番です。

```
r = symrcm(B);
m = symamd(B);
```

つぎに示すようにラインで作成される 3 つの spy プロットは、これらの 3 つの異なる番号付けによる Bucky ボールの 3 つの隣接行列を示しています。オリジナルの番号付けの局所的な多角形をベースにした構造は、他のものの中では、はっきりしていません。

```
spy(B)
spy(B(r,r))
spy(B(m,m))
```



逆 Cuthill-McGee 並び r は、バンド幅を狭くして、対角部近傍にすべての非ゼロ要素を集中せさます。最小次元の並び m は、大きなゼロのブロックをもつフラクタルに似た構造を作成します。

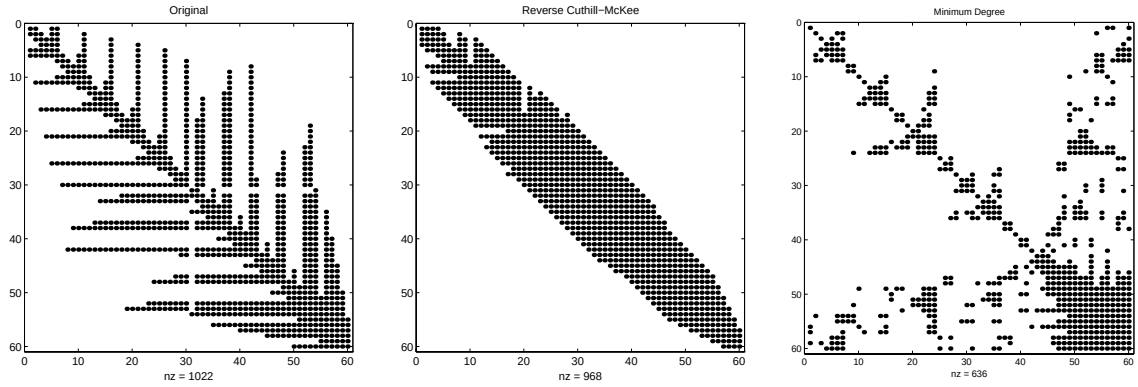
Bucky ボールの LU 分解で生じる fill-in を見るには、スパース単位行列 speye(n,n) を使って、B の対角要素に -3 を設定します。

```
B = B - 3*speye(n,n);
```

各行の和がゼロなので、この新しい B は正則ではありません。しかし、LU 因子分解を計算することはできます。一つの出力引数のみを定義して、コールすると、単一のスパース行列に、lu は 2 つの三角ファクタ L,U を出力します。この行列の中の非ゼロ要素数は、B に含まれる線形システムを解くために必要な時間とストレージの尺度になります。つぎに、3つの置換に対する非ゼロカウントを示します。

オリジナル	lu(B)	1022
逆 Cuthill-McKee	lu(B(r,r))	968
最小度合い	lu(B(m,m))	636

これは小さな規模の例題ですが、結果は典型的なものです。オリジナルの番号付けの方法は、大部分が fill-in になります。逆 Cuthill-McKee 法は、バンド内に集中しますが、最初の 2 つの番号付けとほぼ同じ量になります。最小次数法では、ゼロの数が比較的大きいブロックは消去作業中に保存され、fill-in の量は他の方法に比べ極端に減ります。つぎの spy プロットは、各々の方法の特徴を示しています。



Cholesky 因子分解

S が対称 (またはエルミート)、正定、スパース行列ならば、つぎのステートメント

```
R = chol(S)
```

は、R'*R = S を満たす上三角スパース行列 R を出力します。chol は、スパースに対しては、自動的にピポットを行ないませんが、chol(S(p,p)) の使用に対して最小次数を計算し、制限付置換を探します。

Cholesky アルゴリズムは、スパースに対してピポットを使わず、数値的安定に対してピポットを必要としないので、因子分解の最初にすべてのメモリの割り当てと必要なメモリの総量をすぐ計算できます。

QR 因子分解

MATLAB は、つぎのステートメント

```
[Q,R] = qr(S)
```

でスパース行列 S の完全な QR 分解を行ないます。しかし、実際には、現実的なものではありません。直交行列 Q は、ゼロ要素の割合が高いとき、うまく実行できないことがあります。より実際的な変更法、いわゆる "Q-less QR 分解 " として知られている方法が利用可能になります。

スパース行列を入力引数とし、出力引数を共に 1 個ずつ設定する

```
R = qr(S)
```

と、QR 分解の上三角部を出力します。行列 R は正規方程式 R'*R = S'*S に関連した行列に対する Cholesky 分解を与えます。

しかし、S'*S の計算の中で固有の数値的情報の欠落が避けられます。

同じ行数をもつ 2 つの入力引数と 2 つの出力引数を設定すると、

```
[C,R] = qr(S,B)
```

は、B に直交変換を適用し、Q を実際には計算しませんが、C = Q'*B になる C を作成します。

Q-less QR 分解は、スパース最小二乗問題

$$\text{minimize}\|Ax - b\|$$

の解を、つぎの 2 ステップで出力します。

```
[c,R] = qr(A,b)
x = R\c
```

A がスパースで正方でなければ、MATLAB は、これらのステップをバックスラッシュ演算子を使って解きます。

```
x = A\b
```

または、ユーザは自分自身で因子分解を行ない、ランク落ちに関してはRをチェックしてください。

R = qr(A) が計算されたとき、必ずしも必要ありませんが、異なる右辺bをもつ最小二乗線形連立方程式を解くこともできます。アプローチは、" 準正規方程式 "

```
R'*R*x = A'*b
```

を解きます。ここで、

```
x = R\(R'\(A'*b))
```

そして、丸め誤差を低減するために繰り返し計算を行ないます。

```
r = b - A*x
e = R\(R'\(A'*r))
x = x + e
```

不完全因子分解

関数 luinc と cholinc は、近似的な、不完全因子分解を与えます。これは、スパース繰り返し法で前置条件として利用可能なものです。

関数 luinc は、2 つの異なる不完全 LU 因子分解を作成します。1 つは、drop tolerance を含んでいるもので、他は fill-in レベルを含んでいるものです。A がスパース行列で、tol が小さなトレランスのとき、

```
[L,U] = luinc(A,tol)
```

は、tol と対象の列のノルムとの乗算値よりも小さなすべての要素をゼロに設定する近似的な LU 因子分解を計算します。また、

```
[L,U] = luinc(A,'0')
```

は、近似的な LU 因子分解を計算します。ここで、L+U のスパースパターンは、A のスパースパターンの置換です。

たとえば、

```
load west0479
A = west0479;
nnz(A)
nnz(lu(A))
nnz(luinc(A,1e-6))
```

```
nnz(luinc(A,'0'))
```

は、A が 1887 個の非ゼロ要素をもち、この完全な LU 因子分解は 16777 非ゼロをもち、drop tolerance が 1e-6 を使った不完全 LU 因子分解は 10311 個の非ゼロ要素をち、lu('0') 因子分解は、1886 個の非ゼロ要素をもっています。

関数 luinc は、2,3 のオプションをもっています。詳細は、オンラインヘルプを参照してください。

関数 cholinc は、drop tolerance と対称、正定スパース行列の level 0 fill-in Cholesky 分解を作成します。詳細は、オンラインヘルプを参照してください。

連立方程式

連立方程式システムは、2 つの異なるクラスの方法を使って解かれます。

- 直接法 . これは、Gauss 消去法を改良し、LU または Cholesky 因子分解のような行列因子分解として表現します。含まれているアルゴリズムは、各々の行列要素にアクセスします。
- 繰り返し法：有限回の繰り返しの後、近似解のみが得られます。係数行列は、行列とベクトルの積を通して、または抜き出した線形演算子の結果として間接的に得られます。

直接法

直接法は、実行するコンピュータが十分なストレージをもっていると、より速く、より一般的に使われるアプローチです。繰り返し法は、制約の付いた方程式に対して使われ、対角部が優勢または微分演算子が存在しているようなプロパティに依存します。直接法は、MATLAB のコアの中で実行され、行列の一般的なクラスに対して、できるだけ効果的に機能します。繰り返し法は、通常 MATLAB M- ファイルで実現され、サブ問題またはプリコンデション問題に対して直接的に使用します。

MATLAB の中で直接法にアクセスする通常の方法は、lu または chol を使うのではなく、行列除算演算子 / または \ を使います。A が正方ならば、X = A\B は、線形システム A*X = B の解です。A が正方でなければ、最小二乗解が計算されます。

A が、正方で、フルまたはスパース行列ならば、A\B は、B と同じストレージクラスをもっています。この計算は、いくつかのアルゴリズムの選択が可能です。

- A が三角行列ならば、B の各列に対して三角行列を解きます。
- A が三角行列の置換ならば、それを置換し、B の各列に対して、スパース三角行列を解きます。

- A が対称行列かまたはエルミート行列のどちらかで、正の実数対角要素をもっていると、対称最小次数 p を探し、A(p,p) の Cholesky 因子分解を計算しようとします。うまく行けば、B の各列に対して、2 つのスパース三角行列を解き終了します。
- 他の場合 (A が三角でない、または正の対角要素をもつエルミートでない、または Cholesky 分解がうまくできない)、列最小次数 p を見つけます。A(:,p) の部分ピボットを使った LU 分解を計算し、B の各列に対する 2 つの三角行列を解きます。

正方行列に対して、MATLAB はコストの増加のためにこれらの可能性を試します。三角性と対称性のテストは、より速いコンピュータとより多くのメモリ容量を使えるならば、比較的速くなります。

例えば、つぎのステートメント群を考えます。

```
[L,U] = lu(A);
y = L\b;
x = U\y;
```

この場合、L は三角行列の置換で、U は三角行列なので、MATLAB は右、左割り算に対して三角行列を使います。

異なる順序を使って． A が、三角行列、または、三角行列の置換のどちらでもない場合、バックラッシュ (\) を使って、colmmd と symmmd は、最小次数度合いを決定します。ユーザが、特定の行列に対して、より良い並びを使いたい場合、関数 spparms を使って、最小次数の前もって与えた並びを切り替えます。

A がスパースで、x = A\b が、LU 因子分解を使っている場合、つぎの例題のように、x に対して解くために、colmmd よりも列の並びを使います。

```
spparms('autommd',0);
q = colamd(A);
x = A (:,q) \ b;
x(q) = x;
spparms('autommd',1);
```

A が、Cholesky 因子分解を使って分解する場合、x = A\b は、つぎのものを使って、効率的に計算されます。

```
spparms('autommd',0);
p = symamd(A);
x = A(p,p) \ b(p);
x (p) = x;
```

```
spparms('autommd',1);
```

上の例題の中で、ステートメント spparms('autommd',0 は、自動的に、colmmd、または、symmmd 度合いをオフにします。ステートメント spparms('autommd',1) は、元に戻し、ユーザが後で、適切な度合いを設定しないで、使うことができます。引数をもたないで、spparms を使用すると、スパースパラメータのカレントの設定をレポートします。

繰り返し法

連立線形方程式のスパースシステムを繰り返し法を使って解く 9 つの関数があります。

スパースシステムに対する繰り返し法を使う関数

関数	詳細
bicg	双共役勾配法
bicgstab	安定化された双共役勾配法
cgs	共役勾配二乗法 d
gmres	一般化された最小残差法
lsqr	正規方程式での共役勾配の LSQR 実現
minres	最小残差
pcg	Preconditioned 共役勾配法
qmr	準最小残差
symmlq	対称 LQ 分解

すべての 9 つの方法が、$Ax = b$ または $\min\|b - Ax\|$ を解くために設計されています。preconditioned 共役勾配法 pcg は、行列 A が対称で、正定であることが必要です。minres と symmlq は、対称行列に限られ、行列は、必ずしも正定である必要はありません。lsqr に対して、行列は、正方である必要はありません。他の 5 つの関数は、非対称な正方行列を取り扱うことができます。

すべての 9 つの方法が、preconditioner を使います。線形システム

$$Ax = b$$

は、つぎの等価な

$$M^{-1}x = M^{-1}b$$

preconditioner M は、繰り返し法の収束を速めるために選択されます。多くの場合、preconditioner は、数学的なモデルの中で自然に生じます。変数係数をもつ偏微分方程式は、定数係数をもつ偏微分方程式で近似されます。不完全な行列因子分解は、自然の preconditioner がなくても使うことができます。

正方形上で Laplace 方程式に 5 点の有限差分近似を例題で示します。つぎのステートメントは、共役勾配法の preconditioner M = $R'*R$ を使います。ここで、R は A の不完全 Cholesky 因子です。

```
A = delsq(numgrid('S',50));
b = ones(size(A,1),1);
tol = 1.e-3;
maxit = 10;
R = cholinc(A,tol);
[x,flag,err,iter,res] = pcg(A,b,tol,maxit,R',R);
```

たった 4 回の繰り返しで、前もって設定した精度に達します。

繰り返し法と不完全因子分解に関する背景は、つぎの文献 [2] と [7] を参照してください。

固有値と特異値

つぎの 2 つの関数は、2,3 の特殊な固有値または特異値を計算するのに利用可能です。svds は、ARPACK [6] を利用する eigs を基本にしています。

固有値または特異値を計算する関数

関数	小差
eigs	複数の固有値
svds	複数の特異値

これらの関数は、スパース行列と共に頻繁に使われるものです。しかし、これらはフル行列でも可能で、M- ファイルで定義される線形演算子と共にも使うことができます。

ステートメント

```
[V,lambda] = eigs(A,k,sigma)
```

は、" シフト "sigma の近傍に存在する行列 A の k 個の固有値と対応する固有ベクトルを探します。sigma を省略すると、大きさの最大の固有値を探します。sigma がゼロならば、最小の大きさの固有値を探します。2 番目の行列 B は、一般化固有問題に対して含まれます。

$$Av = \lambda Bv$$

ステートメント

```
[U,S,V] = svds(A,k)
```

は、A の k 番目に大きい特異値を求め、

```
[U,S,V] = svds(A,k,0)
```

は、k 番目に小さい特異値を出力します。

例えば、ステートメント

```
L = numgrid('L',65);
A = delsq(L);
```

は、L 字型、2 次元領域で 65 行 65 列のグリッド上に 5 点 Laplace 微分演算子を設定します。ステートメント

```
size(A)
nnz(A)
```

は、A は 14,473 個の非ゼロ要素をもつ 2945 字の行列を示します。

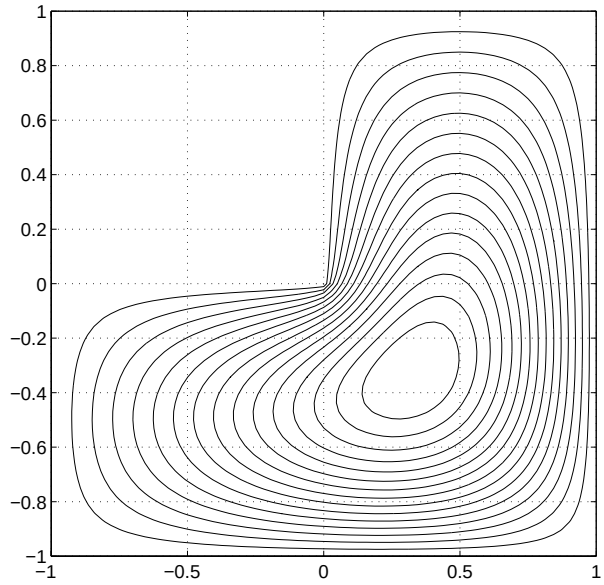
ステートメント

```
[v,d] = eigs(A,1,0);
```

は、最小の固有値と固有ベクトルを計算します。最終的に、

```
L(L>0) = full(v(L(L>0)));
x = -1:1/32:1;
contour(x,x,L,15)
axis square
```

は、適切なグリッド点に渡って固有ベクトルの成分を作成し、結果のコンタプロットを作成します。



Teigs,svds で使う数値的な技法は、[6] に記述されています。

参考文型

[1] Amestoy, P. R., T. A. Davis, and I. S. Duff, "An Approximate Minimum Degree Ordering Algorithm," *SIAM Journal on Matrix Analysis and Applications*, Vol. 17, No. 4, Oct. 1996, pp. 886-905.

[2] Barrett, R., M. Berry, T. F. Chan, et. al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1994.

[3] Davis, T.A., Gilbert, J. R., Larimore, S.I., Ng, E., Peyton, B., "A Column Approximate Minimum Degree Ordering Algorithm," *Proc. SIAM Conference on Applied Linear Algebra*, Oct. 1997, p. 29.

[4] Gilbert, John R., Cleve Moler, and Robert Schreiber, "Sparse Matrices in MATLAB: Design and Implementation," *SIAM J. Matrix Anal. Appl.*, Vol. 13, No. 1, January 1992, pp. 333-356.

[5] Larimore, S. I., *An Approximate Minimum Degree Column Ordering Algorithm*, MS Thesis, Dept. of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, 1998, available at http://www.cise.ufl.edu/tech-reports/

[6] Lehoucq, R. B., D. C. Sorensen, C. Yang, *ARPACK Users' Guide*, SIAM, Philadelphia, 1998.

[7] Saad, Yousef, *Iterative Methods for Sparse Linear Equations*. PWS Publishing Company, 1996.

プログラミングとデータタイプ

MATLAB は、コントロールフローステートメント、関数、データ構造 、入力 / 出力、オブジェクト指向機能を含んだ高水準の言語です。ここでのトピックスは、MATLAB のプログラミングの特徴や技法について、議論しています。

- M- ファイルプログラミング – 言語構成や、MATLAB プログラム (M- ファイル) の作成法を議論します。これには、データタイプ、フローコントロール、配列インデックス、性能の最適化、その他のトピックスを含んでいます。
- キャラクタ配列 – キャラクタ配列の作成法、文字列のセル配列の作成法、文字列の表現法、一般的な文字列演算法、文字列と数値との間での変換を含む、文字列データに対する MATLAB でのサポートについて記述しています。
- 多次元配列 – 2 次元以上の MATLAB 配列の使用法を議論します。ここでは、配列の作成、配列のインデックス化、データの構成や、MATLAB 関数の適用法等を説明します。
- 構造体とセル配列 – 異なるデータの種類に対して、階層的なストレージをもつ 2 つの MATLAB データタイプがあります。構造体は、名前付きのフィールドで構成された種々のデータをもっています。セル配列は、MATLAB 配列を含んでいるセルから構成されています。
- 関数ハンドル - MATLAB 関数ハンドルを使って、関数リファレンスを含む関数に関する情報を捕らえる方法を記述します。それで、MATLAB の環境の中で、それを使って、関数を任意の場所から実行することができます。
- MATLAB クラスとオブジェクト – MATLAB のオブジェクト指向プログラミング機能の説明。クラスとオブジェクトルは、MATLAB に新しいデータタイプや新しい演算を加えることになります。この節は、MATLAB の中でうまく機能するクラスの実現法の例題を示します。

関連情報

つぎの節は、MATLAB のプログラミングで有効な情報を提供します。このドキュメントは、MATLAB のオンラインヘルプで利用できます。

- グラフィックス – 2 次元、3 次元表現でのベクトルや行列のプロット法、凡例の記述法、印刷、プロットの転送法や、グラフィックオブジェクトやそのフィギュアプロパティ、軸プロパティの使い方を示します。
- MATLABからCやFortranプログラムのコール方法 - コールされるMEX-ファイルの中に C や Fortran サブルーチンを作成する方法を記述します。
- CやFortranプログラムからMATLABをコール - CやFortran からMATLABをコールするエンジンライブラリの使い方を議論します。
- MATLAB から Java をコール - Java クラスやオブジェクトへの MATLAB インタフェースの使い方を議論します。
- データの読み込みと書き込み - MATLAB 環境へのデータの読み込みや、MATLAB 環境からの書き出し方法を記述します。
- ActiveX と DDE のサポート - ActiveX と Dynamic Data Exchange(DDE) の使い方を議論します
- シリアルポート I/O - ユーザが自身のシリアルポートに接続できるモデム、プリンタ、科学的な装置等との通信法について記述します。

17

M- ファイルプログラミング

MATLAB プログラミング : クイックスタート 17-3

Scripts 17-7

Functions 17-8

ローカル変数とグローバル変数 17-19

データタイプ 17-22

演算子 17-25

フローコントロール 17-34

サブ関数 17-42

プライベート関数 17-44

サブスクリプトとインデックス 17-45

文字列の計算 17-51

コマンド / 関数の双対性 17-53

空行列 17-54

エラーとワーニング 17-56

時刻と日付 17-59

ユーザ入力の取得 17-66

シェルエスケープ機能 17-67

MATLAB コードの性能の最適化 17-68

MATLAB は、1 つのコマンドで一連の MATLAB ステートメントをファイルに書き込み、それらを実行するためのプログラミング言語を提供します。通常のテキストファイルにプログラムを書き、ファイル名を filename.m とします。filename に対して利用する単語は、プログラムと関連付けられる新規のコマンドとなります。ファイル拡張子 .m により、これは MATLAB M- ファイルとなります。

本章では、M- ファイルでの script および function プログラムの基本的な書き方を説明します。以下のトピックスを含みます。

- "MATLAB プログラミング : クイックスタート "
- "Scripts"
- "Functions"
- " ローカル変数とグローバル変数 "
- " データタイプ "
- " 演算子 "
- " フローコントロール "
- " サブ関数 "
- " プライベート関数 "
- " サブスクリプトとインデックス "
- " 文字列の計算 "
- " コマンド / 関数の双対性 "
- " 空行列 "
- " エラーとワーニング "
- " 時刻と日付 "
- " ユーザ入力の取得 "
- " シェルエスケープ機能 "
- "MATLAB コードの性能の最適化 "

MATLAB プログラミング： クイックスタート

M- ファイルは、一連の MATLAB ステートメントを実行する *script*、あるいは、引数を受け取り出力を生成する *function* です。テキストエディタを使って M- ファイルを作成し、他の MATLAB 関数やコマンドと同様に利用します。

手順はつぎのようになります。

1 テキストエディタを使って M- ファイルを作成します。

```
function c = myfile(a,b)
c = sqrt((a.^2)+(b.^2))
```

2 コマンドラインから M- ファイルを呼び出すか、他の M- ファイル内から呼び出します。

```
a = 7.5
b = 3.342
c = myfile(a,b)

c =

   8.2109
```

M- ファイルの種類

以下の種類の M- ファイルがあります。

Script M- ファイル	Function M- ファイル
• 入力引数、または、出力引数を設定しません。	• 入力引数を受け入れ、出力引数を返します。
• ワークスペース内のデータを使用	• 内部変数はデフォルトにより関数に対してローカルになります。
• 何度も繰り返す必要のある一連のステップを自動化するのに有効	• ユーザのアプリケーション用に MATLAB 言語を拡張するのに有効

M- ファイルとは？

この節は、function M- ファイルの基本的な部分を示し、MATLAB のプログラミングに慣れるように、いくつかの例題を使います。

```
function f = fact(n) % Function 定義ライン
% FACT 因子分解     % H1(help 1) ライン
% FACT(N) は、N の因子分解を出力 H! % Help テキスト
% 通常、N で定義
% 簡単な置き換え、FACT(N) は、PROD(1:N) と等価

f = prod(1:n);        % Function body
```

この関数は、すべての MATLAB 関数で共通ないくつかの要素をもっています。

- *function 定義ライン*．このラインは、関数名と入力引数と出力引数の数と順番を定義します。
- *H1 ライン*．ヘルプの 1 行 "help 1" のラインの略です。MATLAB は、lookfor を使うとき、関数に対して H1 ラインを表示します。または、全ディレクトリ上でヘルプを使うときに H1 ラインを表示します。
- *Help テキスト*．MATLAB は、ユーザが指定した関数のヘルプを求めたとき、H1 ラインを含み、それに続くヘルプテキストを表示します。
- *関数の 本体*．関数のこの部分は、実際に計算したり、出力引数にある値を割り当てるためのコードを含んでいます。

MATLAB 関数の詳細は、17-8 ページの "Functions" を参照してください。

プログラムのヘルプ

M- ファイルの先頭にヘルプテキスト部分を含めることによって、プログラムに対するユーザ情報を与えることができます。これは、関数定義ラインから始まり最初のブランク行で終わります。ヘルプテキストの各ラインは、コメントキャラクタ (%) で始まらなければなりません。MATLAB は、つぎのようにタイプするとこの情報を表示します。

```
help m-file_name
```

特殊なファイル名 Contents.m を使ってファイルを作成することにより、ディレクトリ内にあるディレクトリ全体に対するヘルプエントリを作成することもできます。このファイルは、コメントラインのみの含まなければなりません。つまり、各ラインは、パーセント記号で始まらなければなりません。MATLAB は、つぎのようにタイプすると Contents.m ファイルのラインを表示します。

```
help directory_name
```

ディレクトリが Contents.m ファイルを含まない場合は、help directory_name とタイプすると、ディレクトリ内の各 M- ファイルに対して最初のヘルプライン (H1 ライン) を表示します。

M- ファイルの作成： テキストエディタへのアクセス

M-ファイルは、テキストエディタを使って作成する通常のテキストファイルです。MATLAB は、ユーザが好きな任意のテキストエディタを使うことはできますが、組み込みのエディタも用意しています。

注意 . PC でエディタを開くには、**ファイル (File)** メニュから**新規作成 (New)** を、そして、**M- ファイル (M-File)** を選びます。

M- ファイルをエデットする他の方法は、edit コマンドを使って、MATLAB コマンドラインからの方法です。例えば、

```
edit foo
```

は、ファイル foo.m を開きます。ファイル名を省略すると、Untitled ファイルをオープンします。

ユーザのテキストエディタを開き、示されたラインを入力し、ユーザのディレクトリに fact.m と呼ばれるファイルにテキストをセーブして、関数 fact を作ることができます。

ユーザが一度このファイルを作成すると、ここで、ユーザが実行できることがいろいろあります。

- ユーザのカレントディレクトリの中に存在するファイル名を表示します。

  ```
  what
  ```

- M- ファイル fact.m の内容を表示します。

```
type fact
```

- 関数 fact をコールします。

```
fact(5)

ans =

     120
```

注意　ユーザが作成した M- ファイルや、MATLAB が提供している M- ファイルをあるデレクトリの中でエデットしたものをMATLABデレクトリツリーにセーブしないでください。MATLAB デレクトリツリーにセーブすると、新しいバージョンをインストールする場合に、書き換えてしまう可能性があります。MATLAB/toolbox デレクトリツリー内のファイルは、実行性能を高めるため、MATLAB の各セクションで、ロードして、メモリに常駐させる方法があることにも注意してください。MATLAB/toolbox デレクトリツリーの中の新しいファイル、または、エデットしたファイルをセーブする場合、MATLAB を再起動するか、関数 rehash を使って、デレクトリを再トーロし、ファイルを使う前に、キャシュを更新してください。

Scripts

Scripts は、最も簡単な M- ファイルです。これは、入力引数も出力引数ももっていません。これは、コマンドラインから繰り返し行なうような計算の一連のMATLAB コマンドを自動的に実行するときに有効です。Scripts は、ワークスペースに存在しているデータを使うか、またはそれ自身が作成したデータを使います。Scripts が作成するある変数は、Scripts が終了した後、ワークスペースに残ります。そして、つぎの計算に使うことができます。

簡単な Scripts の例題

つぎのステートメントは、theta に関するいくつかの三角関数を計算し、変数 rho に割り当て、極座標プロットを作成します。

```
% An M–file script to produce      % Comment ライン
% "flower petal" plots
theta = –pi:0.01:pi;               % 計算
rho(1,:) = 2*sin(5*theta).^2;
rho(2,:) = cos(10*theta).^3;
rho(3,:) = sin(theta).^2;
rho(4,:) = 5*cos(3.5*theta).^3;
for i = 1:4
    polar(theta,rho(i,:))           % グラフ出力
    pause
end
```

これらのコマンドを petals.m と名付けられた M- ファイルに入力してください。このファイルは、MATLAB scripts になります。MATLAB コマンドラインで、petals とタイプインすると、scripts の中のステートメントが実行されます。

Script がプロット図を表示した後、Return を押すと、つぎのプロットに移ります。このファイルには、入力引数も出力引数も何もありません。すなわち、petals は、MATLAB のワークスペース中に必要な変数を作成します。完全に実行すると、変数 (i, theta, rho) は、ワークスペース内に残ります。これらの一覧を表示するには、コマンドプロンプトに whos とタイプインしてください。.

Functions

Functions は、入力引数を受け入れ、出力引数を返す M- ファイルです。これらは、M- ファイル自身のワークスペースの中で変数を扱い、MATLAB コマンドプロンプトでアクセスするワークスペースとは異なります。

ここでは、以下の機能に関連するトピックスを説明します。

- " 簡単な Scripts の例題 "
- "Function M- ファイルの基本部分 "
- "Function 名 "
- "Functions ファイルの動き "
- "Function 引数の数のチェック "
- " 転送する引数の数を可変にする "

簡単な Function の例題

関数 average は、ベクトル内の要素の平均を計算する簡単な M- ファイルです。

```
function y = average(x)
% AVERAGE は、ベクトル要素の平均を算出
% AVERAGE(X) は、ベクトル要素の平均です。ここで、X はベクトルです。
% ベクトル入力でない場合、エラーが出力されます。
[m,n] = size(x);
if (~((m == 1) | (n == 1)) | (m == 1 & n == 1))
    error('Input must be a vector')
end
y = sum(x)/length(x);       % Actual computation
```

希望する場合、これらのコマンドを入力して、average.m と呼ばれる M- ファイルを作成してください。関数 average は、一つの入力引数を受け入れ、一つの出力引数を返します。関数 average を使うには、つぎのように行ないます。

```
z = 1:99;

average(z)

ans =
    50
```

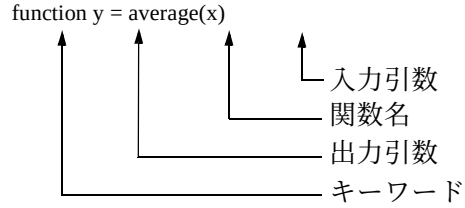
Function M- ファイルの基本部分

Function M- ファイルには、つぎの内容が含まれています。

- "Function 定義ライン "
- "H1 ライン "
- " ヘルプテキスト "
- "Function 本体 "
- " コメント "

Function 定義ライン

Function 定義ラインは、MATLAB に M- ファイルが function で、関数を読み込むときに引数を設定することが必要であることを示します。関数 average に対して、Function 定義ラインは、つぎのようになっています。



すべての MATLAB 関数は、このパターンに従って Function 定義ラインをもっています。

Functionが複数の出力引数をもつ場合は、出力引数全体を鍵括弧で囲んで下さい。入力引数が存在する場合、丸括弧で囲んでください。複数の入力や出力引数を分離するには、カンマを使ってください。ちょっと複雑な例を示します。

```
function [x,y,z] = sphere(theta,phi,rho)
```

出力が存在しない場合、出力をブランクにします。

```
function printresults(x)
```

または、空の鍵括弧を使います。

```
function [] = printresults(x)
```

ユーザが Function に渡す変数は、Function 定義ラインの中の変数名と同じである必要はありません。

H1 ライン

H1 ラインは、最初のヘルプテキストラインのためこのように呼ばれ、Function 定義ラインのすぐ後に続きます。これは、コメントテキストから成り立っているので、H1 ラインは、% マークで始まります。関数 average に対して、H1 ラインは、つぎのようになります。

```
% AVERAGE ベクトル要素の平均
```

これは、MATLAB プロンプトで、help *function_name* とタイプすると表示されるテキストの最初のラインです。さらに、lookfor コマンドは、H1 ラインのみをサーチし、表示します。このラインは、M- ファイルの概要についての重要な情報を与えるため、できるだけ詳しく記述してください。

ヘルプテキスト

ユーザは、H1 ラインで始まり、それに続く 1 行または複数コマンドラインのテキストを入力することで、ユーザ自身の M- ファイルのオンラインヘルプを作成することができます。関数 average に対するヘルプテキストは、つぎのものです。

```
% AVERAGE(X) は、ベクトル要素の平均、ここで、X はベクトルです
% ベクトル以外の入力は、エラーになります。
```

help function_name とタイプすると、MATLAB は Function 定義ラインとコメント文でない最初のラインの間にあるコメントラインを表示します。ヘルプシステムは、このヘルプブロックの後に表れるコメントラインは無視します。

たとえば、help sin とタイプすると、次の結果を得ます。

```
SIN     Sine.
        SIN(X) is the sine of the elements of X.
```

Function 本体

Function 本体は、計算を実行し、値を出力引数に割り当てる MATLAB コードを含んでいます。Function 本体の中のステートメントは、function を呼び、制御文による流れ、会話型の入 / 出力、計算、代入、コメント、ブランクライン等々のプログラミング構造から作られています。

例えば、関数 average の本体は、2、3 の簡単なプログラミングステートメントを含んでいます。

```
[m,n] = size(x);
if (~((m == 1) | (n == 1)) | (m == 1 & n == 1)) % フローのコントロール
    error('Input must be a vector') % エラーメッセージの表示
end
y = sum(x)/length(x);               % 計算と代入
```

コメント

前に記述したように、コメントラインはパーセント (%) で始まります。コメントラインは、M- ファイルの中の任意の位置に設定できます。そして、コードラインの後にも続けることができます。例えば、

```
% すべてのベクトル要素の和
y = sum(x)            % 関数 sum の使用
```

最初のコメントラインは、function 定義ラインのすぐ後に続き、関数に対する H1 ラインと考えられます。H1 ラインとそれに続くコメントラインは、ファイルについてのオンラインヘルプを構成します。

コメントラインに加え、M- ファイルの任意の位置にブランクラインを挿入できます。しかし、ブランクラインは、M- ファイルに対して、ヘルプテキスト部の最後を示すことができます。

Function 名

MATLAB function 名は、変数名の制約と同じ制約をもっています。MATLAB は名前について最初の 31 キャラクタを使います。function 名は、最初の 1 キャラクタは文字で、残りのキャラクタは文字、数字、下線も使うことができます。いくつかのオペレーテングシステムでは、より短い function 名しか使うことができません。

MATLAB function を含むテキストファイルの名前は、拡張子 ".m" をもつ関数名から構成されています。例えば、

```
average.m
```

ファイル名と function 定義ライン名が異なっていると、ファイル名が優先され、内部の名前は無視されます。

そのため、function 定義ラインで設定される function 名は、ファイル名と同じ名前にする必要はありませんが、同じ名前をできるだけ使うことを推薦します。

Functions ファイルの動き

MATLAB コマンドライン、または、他の M- ファイルの内部から Function M- ファイルを読むことができます。すべての必要な引数、すなわち、丸括弧の中に入力引数、鍵括弧の中に出力引数を設定してください。

ここでは、MATLAB 関数の呼び出しに関する情報を提供します。

- "Function 名の認識について "
- "Function を読み込んだとき、何が生じますか "
- "P- コードファイルの作成 "
- "MATLAB は、どのように Function へ引数を渡しますか "
- "Function ワークスペース "

Function 名の認識について

MATLAB は、新しい名前が入ってくると、つぎのステップに従って、設定した Function の認定を行ないます。

1 名前が、変数かどうかをチェックします。

2 名前が、*サブ関数* かどうかをチェックします。サブ関数は、Function を読み込んだときに、同じ M- ファイルの中に設定している MATLAB Function のことです。詳細は、17-42 ページの " サブ関数 " を参照してください。

3 名前が、*プライベート関数 かどうかをチェックします。すなわち、プライベートディレクトリ* すなわち、そのすぐ上のディレクトリの M- ファイルのみをアクセスできるディレクトリに保存されている MATLAB Function かどうかをチェックします。この詳細は、17-44 ページの " プライベート関数 " を参照してください。

4 名前がMATLABサーチパスの上のFunctionかどうかをチェックします。MATLABは設定した名前が最初に表れるファイルを使います。

関数名が重複している場合、MATLAB は上の規則に従って最初にマッチするものを実行します。また、関数名をオーバロードすることもできます。これは、付加的なパッチ則を使い、22-65 ページの "MATLAB がメソッドを読み込む優先順位の決定法 " で説明します。

Function を読み込んだとき、何が生じますか

コマンドライン、または、他の M- ファイルから Function M- ファイルを読み込む

と、MATLAB は、Function を擬似コードに分解したり、メモリに保存します。これにより、MATLAB 実行中に、その関数を使うたびに毎回擬似コードに分解はしません。擬似コードは、コマンド clear を使ってクリアするか、MATLAB を終了するまでメモリ内に残ります。

この clear コマンドの使い方を示します。

シンタックス	詳細
clear function_name	ワークスペースから指定した関数を消去
clear functions	すべてのコンパイルした M- ファイルを消去
clear all	すべての Function と変数を消去

P- コードファイルの作成

pcode コマンドを使って、その後で MATLAB で使うために、Function、または、Scripts の分解したバージョンをセーブすることができます。例えば、

```
pcode average
```

は、average.m を分解し、average.p と名付けられたファイルに擬似的にコード化されたものをセーブします。これは、最初に average.m を分解し、その結果をセーブし、その後、実行するときに average.p を使います。

MATLAB は分解するのにあまり時間を使いません。従って、pcode コマンドにより、処理スピードが変化することはあまりありません。

pcode は、大きな GUI アプリケーションには、スピード的に非常に効率的に働きます。GUI では、アプリケーションが可視化される前に、多くの M- ファイルを分解する必要があります。

pcode に関する他の考えは、ユーザ自身の M- ファイルのアルゴリズムを隠すためです。

MATLAB は、どのように Function へ引数を渡しますか

プログラマの立場から、MATLAB はすべての関数引数を値として取り扱います。しかし、実際に、MATLAB は、関数が変更した引数のみの値を渡します。Function が引数を変更しなく、単に計算の中でこれを簡単に使うと、MATLAB はメモリ使用を最適化するために基準に従って引数を渡します。

Function ワークスペース

各 M- ファイル Function は、MATLAB ベースワークスペースと異なるメモリ領域をもちます。この領域は、function ワークスペースと呼ばれ、各巻数がもつワークスペースの内容をもっています。

MATLAB を使っている間、ユーザがアクセスできる変数は、引数で設定しているもののみで、ベースのワークスペースに存在するものか、または、他の関数の引数になっているもののみです。ユーザが Function に転送する変数は、読み込む Function の中の引数でなければなりません。そして、Function は、出力引数を読み込んだワークスペースに返します。しかし、変数をグローバル変数として設定することができます。グローバル変数は、複数のワークスペースの中でアクセスすることができます。

Function 引数の数のチェック

関数 nargin および nargout は、Function が呼ばれたとき、入力引数の数と出力引数の数を数えるものです。引数の数によって、処理が変化するように条件付きのステートメントを作ることができます。例えば、

```
function c = testarg1(a,b)
if (nargin == 1)
    c = a.^2;
elseif (nargin == 2)
    c = a + b;
end
```

は、入力引数が 1 つならば引数の二乗を計算し、入力引数が 2 つならば、引数の和を計算します。

つぎに、もう少し進んだ例を示します。これは、キャラクタ文字に中に存在するトークンの最初のものを探索するものです。*トークン* は、スペースまたは他のキャラクタで、キャラクタ集合を分離するものです。入力引数が 1 つの場合、関数はデフォルトの分離子スペースを仮定しています。入力引数が 2 つで、ユーザが希望する場合、他の分離子を設定するものです。出力引数も 2 つ設定することができます。

```
function [token,remainder] = strtok(string,delimiters)
% Function は、少なくとも 1 つの入力引数が必要
if nargin < 1
   error('Not enough input arguments.');
end
token = []; remainder = [];
len = length(string);
if len == 0
   return
end

% 入力が 1 つの場合、スペース分離子を使用
if (nargin == 1)
   delimiters = [9:13 32]; % White space characters
end
i = 1;

% 分離子でないキャラクタの始まりを決定
while (any(string(i) == delimiters))
   i = i + 1;
   if (i > len), return, end
end

% トークンの最後の探索
start = i;
while (~any(string(i) == delimiters))
   i = i + 1;
   if (i > len), break, end
end
finish = i – 1;
token = string(start:finish);

% 2 出力引数に対して、最初の分離子の後に続く
% キャラクタをカウント
if (nargout == 2)
   remainder = string(finish + 1:end);
end
```

strtok は、strfun ディレクトリの中の MATLAB M- ファイルです

注意 関数宣言ラインの中の出力引数の順番が重要です。関数が多くの場合に出力する引数は、リストの最初に表れます。付加的で、オプションの引数はリストの最後に付け加えられます。

転送する引数の数を可変にする

関数 varargin と varargout は、任意の数の入力引数、または任意の数の出力引数を関数に自由に設定させることができます。ここでは、これらの関数の利用法と、以下について説明します。

- "varargin の内容を分解 "
- "varargout の内容を格納 "
- " 引数リスト内の varargin と varargout"

MATLAB は、すべての指定された入力引数を配列要素の代わりににセルで構成される MATLAB 配列の一種である *セル配列* に格納します。各セルは、任意のサイズのデータ、または、任意の種類のデータを含むことができます。すなわち、あるものは数値データベクトルで、同じ配列の中の他のものは文字データ配列と言うように設定できます。出力引数に対して、関数コードは、セル配列に格納されなければならないので、MATLAB は引数を呼び出し側に戻します。

つぎの例題の関数は、任意の数の 2 要素ベクトルを受け入れ、ラインを使って、これらを繋げることができます。

```
function testvar(varargin)
for i = 1:length(varargin)
    x(i) = varargin{i}(1); % セル配列のインデックス付け
    y(i) = varargin{i}(2);
end
xmin = min(0,min(x));
ymin = min(0,min(y));
axis([xmin fix(max(x))+3 ymin fix(max(y))+3])
plot(x,y)
```

この方法をコード化して、関数 testvar は、つぎのような例を使って種々の入力を処理しましょう。

```
testvar([2 3],[1 5],[4 8],[6 5],[4 2],[2 3])
testvar([–1 0],[3 –5],[4 2],[1 1])
```

varargin の内容を分解

varargin は、セル配列の中のすべての入力引数を含んでいるので、データを抜き出すためにセル配列にインデックスを付けて使う必要があります。例えば、

```
y(i) = varargin{i}(2);
```

インデックス付きのセル配列は、2 つのサブスクリプト成分をもっています。

- セルインデックス付きの式は、中括弧を使います。
- 内容インデックス付き式は、丸括弧を使います。

上のコードの中で、インデックス付き式 {i} は、varargin の i 番目のセルにアクセスします。式 (2) は、セルの内容の 2 番目の要素を表現します。

varargout の内容を格納

任意の出力引数の数を受け入れるので、すべての出力を varargout セル配列に格納しなければなりません。関数がいくつの出力引数を読み込んでいるのかを決めるために nargout を使います。例えば、つぎのコードは、2 列の入力配列を受け入れます。最初の列は x 座標を表し、2 番目の列は y 座標を表します。これは、配列を [xi yi] ベクトルに分離し、前の例題の関数 testvar に転送します。

```
function [varargout] = testvar2(arrayin)
for i = 1:nargout
    varargout{i} = arrayin(i,:) % セル配列に代入
end
```

for ループの内部の代入ステートメントは、セル配列の代入シンタックスを使います。ステートメントの左辺、セル配列は、データがセルの内部に入ることを示す中括弧を使ってインデックスを示しています。セル配列代入での完全な情報に対しては、" 構造体とセル配列 " を参照してください。

つぎの例は、testvar2 のコールの方法を示しています。

```
a = {1 2;3 4;5 6;7 8;9 0};
[p1,p2,p3,p4,p5] = testvar2(a);
```

引数リスト内の varargin と varargout

varargin、または、varargout は、引数リストの中で、必要な入力変数、または、必要な変数に続けて、最後に設定します。これは、関数が読み込むために必要な引数をまず設定しなければならないからです。例えば、つぎの関数宣言ラインは、varargin と varargout の正しい位置を示しています。

```
function [out1,out2] = example1(a,b,varargin)
function [i,j,varargout] = example2(x1,y1,x2,y2,flag)
```

ローカル変数とグローバル変数

コマンドラインで MATLAB 変数に適用されるガイドラインが、M- ファイルの中の変数にも適用されます。

- 変数のタイプを宣言する必要はありません。一つの変数を他のものに代入する前に、代入文の右辺側の変数は値でなければなりません。
- ある値をある変数に代入する演算は、必要ならば変数を設定し、既に存在しているならばカレントな値に書き換えます。
- MATLAB 変数名は、最初のキャラクタを文字にし、それに続くキャラクタを文字、数字、下線を組み合わせて作成します。MATLAB は、大文字と小文字を区別します。それで、A と a は同じ変数ではありません。
- MATLAB は変数名を最初の 31 文字までを使います。

通常、M- ファイルで定義される各々の MATLAB 関数は、自身のローカル変数をもち、他の関数のものと分離でき、ベースワークスペースとは異なるものです。しかし、ある関数でグローバルとして特別な名前を宣言すると、この変数の一つのコピーを、宣言したすべての関数で使うことができます。任意の関数の中で、この変数への割り当てを行なうことは、それをグローバルとして宣言する他のすべての関数で、それを使うことが可能になります。

Lotka-Volterra predator-prey モデルの相互作用係数との影響を調べます。

$$\dot{y}_1 = y_1 - \alpha y_1 y_2$$

$$\dot{y}_2 = -y_2 + \beta y_1 y_2$$

M- ファイル lotka.m を作成します。

```
function yp = lotka(t,y)
%LOTKA   Lotka-Volterra 予測子モデル
global ALPHA BETA
yp = [y(1) - ALPHA*y(1)*y(2); -y(2) + BETA*y(1)*y(2)];
```

そして、対話的に、つぎのステートメントを代入します。

```
global ALPHA BETA
ALPHA = 0.01
BETA = 0.02
[t,y] = ode23('lotka',0,10,[1; 1]);
plot(t,y)
```

2 つのグローバルステートメントは、コマンドプロンプトで ALPHA と BETA に値を代入し、lotka.m で定義される関数の内部で使うことができます。これらは、会話的に変更でき、ファイルを編集しないで新しい解を得ることもできます。

MATLAB アプリケーションの中で、グローバル変数を使うには、

- その変数にアクセスする必要のあるすべての関数の中で変数を global として宣言します。グローバル変数をアクセスするワークスペースを作成するにも、コマンドラインからそれを global として宣言します。
- 各関数で、変数名が最初に表れる前に global コマンドを設定してください。M- ファイルの最初で設定することをお勧めします。

MATLAB グローバル変数名は、ローカル変数名に比べてより長く、より記述的に付けます。そして、すべてを大文字で表すこともあります。これらのことは、必ずしも必要なことではありませんが、MATLAB コードを読み易くしたり、グローバル変数を再定義するときの間違いを少なくするためにお勧めします。

固定変数

変数は、あるコールから他のコールの中で値を変更しないようにする、固定変数として定義することもできます。固定変数は、一つの関数の中のみで使われます。固定変数は、M- ファイルがクリアされるか、変更されるまで、メモリに残ります。

固定変数は、グローバル変数と同様のものですが、変数名が、グローバルワークスペースに存在していないことが異なります。そして、M- ファイルが変更されたり、クリアされると、値はリセットされます。

3 つの MATLAB 関数が、固定変数の使用をサポートしています。

関数	詳細
mlock	クリア作業の中で、M- ファイルを保護する
munlock	前に、mlock でロックされている M- ファイルのロック状態を解除します。
mislocked	M- ファイルがクリア可能か、否かを示す

特別な値

S いくつかの関数は、ユーザの M- ファイルの中で使うことのできる重要な特別な値を出力します。

関数	戻り値
ans	一番新しく返された値 (変数)。ユーザが出力変数を変数または表現で与えなければ、MATLAB は自動的に ans に結果を格納します。
eps	浮動小数点相対精度。これは、計算の中で MATLAB が使う許容誤差範囲です。
realmax	ユーザのコンピュータで表現できる最大の浮動小数点数
realmin	ユーザのコンピュータで表現できる最小の浮動小数点数
pi	3.1415926535897...
i, j	虚数単位
inf	無限大。n/0 のような計算結果出力、ここで、n はゼロでない値です。結果は、inf になります。
NaN	Not-a-Number、定義できない数値。0/0 や inf/inf のような表現は、NaN になり、代数演算は NaN になります。n/0 は、NaN を出力します。ここで、n は、複素数でも構いません。
computer	コンピュータタイプ
version	MATLAB のバージョンを表す文字

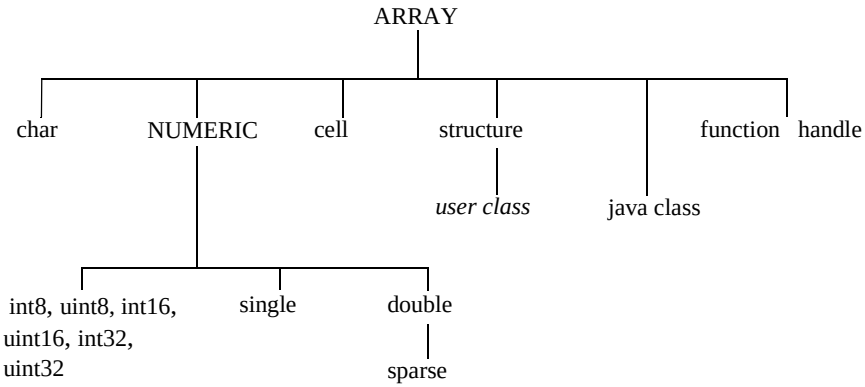
MATLAB 表現でこれらを使った例題を示します。

```
x = 2*pi;
A = [3+2i 7–8i];
tol = 3*eps;
```

データタイプ

MATLAB には、14 の基本的なデータタイプ (またはクラス) があります。これらのデータタイプの各々は、配列の型をしています。この配列は、サイズの最小なものが 0 行 0 列で、任意のサイズの n- 次元配列に成長します。これらの配列の 2 次元バージョンが、*行列と呼ばれています。*

基本的なデータタイプは、下に示すように、ダイアグラムの下部にユニタリで、小文字のテキストで記述されています。また、*user class* に示されているユーザ定義のデータタイプは、structure タイプのサブセットです。



char データタイプは、ユニコード表現で記述されています。キャラクタ文字列は、単に、キャラクタの 1 行 n 列の配列をしています。ユーザは、配列の中の各文字列が同じ長さである場合に、char を使って、文字の配列を作成することができます (これは、MATLAB 配列は、長方形でなければならないからです)。長さの等しくない文字列の配列を作成するには、セル配列を使ってください。

Numeric データタイプは、符号付き、符号なしの整数、単精度、倍精度の浮動小数点、倍精度のスパース配列を含んでいます。つぎのものは、MATLAB の数値データタイプに対して、適切なものです。

- すべての MATLAB での計算は、倍精度で行われます。
- 整数と単精度配列は、倍精度配列よりも、よりメモリの効率的なストレージを使用することになります。
- すべてのデータタイプは、サブスクリプトや整形のような基本的な配列演算をサポートしています。

- 整数精度、または、単精度配列を使って、数学演算を行なうためには、関数 double を使って、それらを倍精度に変換する必要があります。

セル配列は、異なる種類のデータをストレージに保存する手法を提供するものです。同じセル配列のセルの中に、異なるタイプ、サイズの配列をストアすることができます。たとえば、同じセル配列の中のセルに、1 行 50 列の char 配列、7 行 13 列の倍精度配列、1 行 1 列の uint32 配列をストアすることができます。ユーザは、他の MATLAB 行列や配列を使って、同じ行列のインデックス手法を使って、セル配列内のデータにアクセスできます。

MATLAB の structure データタイプは、異なる種類のデータを保存するセル配列に似ています。しかし、構造体の場合は、セルの中ではなく、名前を付けたフィールドでデータを保存する点が異なります。ユーザは、構造体の中に保存したデータグループに、名前を使って、アクセスできます。すなわち、フィールド名を使って、構造体の中のデータにアクセスします。

MATLAB データタイプは、クラスとして実現しています。ユーザ自身の MATLAB クラスを作成することができます。ユーザ定義のクラスは、MATLAB 構造体のクラスから継承し、上のダイアグラムで示すように、構造体のサブセットになります。

MATLAB は、Java プログラム言語とインタフェースをもっており、Java classes からオブジェクトを作成したり、これらのオブジェクトの Java メソッドをコールします。Java クラスは、MATLAB データタイプです。MATLAB インタフェースを通して利用可能な組み込みのクラスやサードパーティのクラスもあります。また、MATLAB にユーザ自身の Java クラスを作成し、取り入れることができます。

関数ハンドルは、MATLAB が計算するために必要な関数に関するすべての情報を含んだ MATLAB データタイプです。関数ハンドルは、一般には、引数リストの中のものを他の関数に渡すものです。そして、ターゲット関数を計算するために、feval コマンドと共に使われるものです。

つぎの表は、データタイプの詳細を記述したものです。

データタイプ	例題	詳細
single	3*10^38	単精度数値配列。単精度は、倍精度より必要なストレージが少なくて済みますが、精度が劣り、レンジも小さくなります。数学的な演算の中では使用することはできません。
double	3*10^300 5+6i	倍精度数値配列 (これは、ほとんどすべての MATLAB 変数タイプであります)。

データタイプ	例題	詳細
sparse	speye(5)	スパース倍精度行列 (2- 次元のみ)。スパース行列は、非ゼロ要素の数が非常に少ない行列で、等価なフル行列に対して必要な領域は、非常に少ないものです。スパース行列は、スパース問題を解くために特別に作成した手法を起動します。
int8, uint8, int16, uint16, int32, uint32	uint8(magic(3))	8 ビット、16 ビット、32 ビットの符号付き、符号なし整数配列。ユーザは、メモリの効率的な使用を考えて、整数量を取り扱うことができます。数学的な操作を行なうことは、できません。
char	'Hello'	キャラクタ配列 (各キャラクタは、16 ビットです)。また、文字列としても参照します。
cell	{17 'hello' eye(2)}	セル配列。セル配列の各々は、別々の配列から構成されています。セル配列は、異なるサイズの関連したデータや情報を選択します。

structure	a.day = 12;
	a.color = 'Red';
	a.mat = magic(3);
		構造体配列。構造体は、フィールド名をもっています。フィールドは、別々の配列を含みます。セル配列同様に、構造体は、共に、関連したデータや情報を選択します。

user class	inline('sin(x)')	MATLAB クラス。このユーザ定義クラスは、MATLAB 関数を使って作成されます。
java class	java.awt.Frame	Java クラス。Java API またはサードパーティで既に定義されたクラスを使ったり、Java 言語のユーザ独自のクラスを作成することができます。
function handle	@humps	MATLAB 関数へのハンドル。引数リストの中から渡すことができ、feval を使って、計算できます。

演算子

MATLAB の演算子は、つぎの 3 つのカテゴリに分けられます。

- 代数演算子、これは数値計算を行ないます。例えば、2 つの数の加算、配列の各要素のベキ乗計算等です。
- 比較演算子、これは"より小さい"や"等しくない"のような演算子を使ってオペランドの量的な比較を行います。
- 論理演算子は AND, OR, NOT を利用します。

ここでは、演算子の優先度についても説明します。

代数演算子

MATLAB は、つぎの代数演算子を提供しています。

演算子	詳細
+	加算
ミ	減算
.*	乗算
./	右除算
.\	左除算
+	単項プラス
ミ	単項マイナス
:	コロン演算
.^	ベキ乗
.'	転置
'	複素共役転置
*	行列の乗算
/	行列の右除算

演算子	詳細
\	行列の左除算
^	行列のべき乗

代数演算子と配列

いくつかの行列演算子を除いて、MATLAB の代数演算子は、同じ次元をもつ配列の対応する要素に機能します。ベクトルや長方形配列に対して、二つのオペランドは、どちらかがスカラでない限り、同じ次元でなければなりません。一つのオペランドがスカラで、他がスカラでない場合、MATLAB は、スカラの値を他のすべての要素に対応させます。このプロパティは、*スカラ展開*として知られています。

つぎの例題は、スカラ展開を使って、スカラオペランドと行列の積を計算するものです。

```
A = magic(3)

A =

     8     1     6
     3     5     7
     4     9     2

3 * A

ans =

    24     3    18
     9    15    21
    12    27     6
```

比較演算子

MATLAB は、つぎの比較演算子を提供します。

演算子	詳細
<	より小さい
<=	より小さいか等しい
>	より大きい
>=	より大きいか等しい
==	等しい
~=	等しくない

比較演算子と配列

MATLAB の比較演算子は、同じ次元をもつ配列の対応する要素を比較します。比較演算子は、常に要素毎の比較を行ないます。つぎの例で、結果の行列は、A の要素が B の対応する要素に等しい位置を示すものです。

```
A = [2 7 6;9 0 5;3 0.5 6];
B = [8 7 0;3 2 5;4 –1 7];
A == B

ans =

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

ベクトルと長方形配列に対して、二つのオペランドは、どちらかがスカラでない場合、同じ次元でなければなりません。一つのオペランドがスカラで、他がスカラでない場合、MATLAB は、スカラ要素をすべての要素と比較します。設定した関係が真である場合、その要素の位置に値 1 を出力します。他の場合は、0 です。

比較演算子と空配列

比較演算子は、二つの配列が同じ大きさか、または、どちらか一方がスカラの場合、次元がサイズゼロの配列にも機能します。しかし、つぎの表現

```
A == []
```

は、A が 0 行 0 列、または、1 行 1 列でない場合、エラーを出力します。

配列がゼロか否かは、つぎの関数でチェックできます。

```
isempty(A)
```

論理演算子

ここでは、MATLAB の論理演算子と以下について説明します。

- " 論理演算子と配列 "
- " 論理関数 "
- " 関数 find を使った論理表現とサブスクリプト機能 "

MATLAB は、以下の論理演算子を提供します。

演算子	詳細
&	AND
|	OR
~	NOT

注意　これらの論理演算子に加え、ops ディレクトリは、ビット単位の論理演算を実行するいくつかの関数を含んでいます。これらの詳細は、オンラインヘルプを参照してください

各論理演算子を使って、論理表現の結果を決める規則の設定集合を表します。

- AND 演算子 & を使った表現は、2 つのオペランドが論理的に真ならば真を出力します。数値の場合、表現は、2 つのオペランドが共にゼロでないならば真になります。つぎの例題は、ベクトル u の中の要素とそれに対応するベクトル v の中の要素の論理 AND を示すものです。

```
u = [1 0 2 3 0 5];
v = [5 6 1 0 0 7];

u & v

ans =

   1   0   1   0   0   1
```

- OR 演算子 | を使った表現は、1 つのオペランドが論理的に真か、または、2 つのオペランドが論理的に真の場合、真になります。OR 表現は 2 つのオペランドが共に偽のときのみ偽になります。数値の場合、表現式は 2 つのオペランドが共にゼロならば偽です。つぎの例題は、ベクトル u の中の要素と、それに対応するベクトル v の要素の論理 OR を示すものです。

```
u | v

ans =

   1   1   1   1   0   1
```

- 演算子 ~ を使った表現は、オペランドを否定するものです。これは、オペランドが真ならば偽の結果を出力し、オペランドが偽ならば真を出力します。数値の場合、ある非ゼロのオペランドはゼロで、ゼロオペランドは 1 になります。つぎの例題は、ベクトル u 要素の否定を示すものです。

```
~u

ans =

    0     1     0     0     1     0
```

論理演算子と配列

MATLAB の論理演算子は、同じ大きさの配列の対応する要素を比較します。ベクトルや長方形配列に対して、2 つのオペランドは、一方がスカラでない限り、同じ

大きさでなければなりません。一つのオペランドがスカラで、他のオペランドがスカラでない場合、MATLAB はスカラでないオペランドのすべての要素に対して、スカラと比較します。設定した関係が真の位置に 1、偽の位置に 0 を出力します。

論理関数

論理演算子に加え、MATLAB はいくつかの論理関数を提供します。

関数	詳細	例題

xor	関数 xor は、対象となるオペランドとの排他的 OR を行ないます。排他的 OR 表現は、一つのオペランドが TRUE ならば TRUE で、他の場合は、FALSE です。数値の場合、関数は一つのオペランドがゼロでなく、他のオペランドがゼロならば 1 を出力します。	a = 1;
		b = 1;
		xor(a,b)
		
		ans =
		0

all	あるベクトルのすべての要素が真であるか、またはすべてがゼロでない場合、関数 all は 1 を出力します。all は行列に関して列単位に機能します。	A = [0 1 2;3 5 0] A = 0 1 2 3 5 0 all(A) ans = 0 1 0
any	引数の任意の要素が、真、または、ゼロでない場合、関数 any は 1 を出力し、他の場合は 0 を出力します。all と同じように、any 関数は、行列に関しては列方向に機能します。	v = [5 0 8]; any(v) ans = 1

いくつかの他の MATLAB 関数も、論理演算を行います。たとえば、関数 isnan は、NaNs に対して 1 を、関数 isinf は、Infs に対して 1 を出力します。論理関数の一覧は、ops デレクトリを参照してください。

関数 find を使った論理表現とサブスクリプト機能

関数 find は、与えられた論理条件を満足する配列要素のインデックスを決定します。これは、行列をマスクしたり、行列にインデックスを作成するのに有効です。最も一般的な型では、find はインデックスを表す単一の行ベクトルを返します。このベクトルは、任意のサイズ、または、型の配列の中のインデックスに対応しています。たとえば、

```
A = magic(4)

A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

i = find(A > 8);
A(i) = 100

A =

   100     2     3   100
     5   100   100     8
   100     7     6   100
     4   100   100     1
```

配列の場合と同様に、長方形行列に対して、条件を満たす行と列両方向のインデックスを得るのに find を使います。より詳細な情報については、ヘルプ機能を使ってください。

演算子の優先順位

代数演算子、比較演算子、論理演算子を任意に組み合わせて、ある表現を作成することができます。優先順のレベルは、MATLAB が表現式を評価するときに順番を決めるものです。個々の優先レベルの中では、演算子は、等しい優先順位をもっていて、その場合は、左から右に評価します。MATLAB 演算子の優先順位則を、高い順位から低い順位に、つぎの表にまとめています。

1 括弧 ()

2 転置 (.'), ベキ乗 (.^), 複素共役転置 ('), 行列のベキ乗 (^)

3 単項プラス (+), 単項マイナス (-), 論理否定 (~)

4 乗算(.*),右除算(./), 左除算(.\), 行列の乗算(*),行列の右除算(/), 行列の左除算 (\)

5 加算 (+), 減算 (-)

6 コロン演算子 (:)

7 より小さい (<), より小さいか等しい (<=), より大きい (>), より大きいか等しい (>=), 等しい (==), 等しくない (~=)

8 論理 AND(&)

9 論理 OR(|)

デフォルトの優先順の書き換え

デフォルトの優先順は、つぎの例題で示すように、括弧を使って、書き換えます。

```
A = [3 9 5];
B = [2 1 5];
C = A./B.^2

C =
    0.7500    9.0000    0.2000

C = (A./B).^2

C =
    2.2500   81.0000    1.0000
```

表現は、ユーザがサブスクリプトを通してアクセスする値を含むこともできます。

```
b = sqrt(A(2)) + 2*B(1)

b =
     7
```

フローコントロール

MATLABには、8つのフローをコントロールするステートメントが用意されています。

- elseやelseifと共に使うifは、ある論理条件をもとに一つのグループ化したステートメントを実行します。
- case や otherwise と共に使う switch は、ある論理条件の値に従って、種々のグループに分かれたステートメントを実行します。
- while は、ある論理条件をもとに、一つのグループ化したステートメントを、条件結果に基づく回数だけ実行します。
- for は、設定した回数だけ一つのグループ化したステートメントを実行します。
- continue は、for、または、while のつぎの繰り返しまで、続けて、残りのステートメントをスキップします。
- break は、for、または、while ループの実行を停止します。
- try...catch は、実行中にエラーが検出される場合、フローコントロールを変更します。
- return は、実行を起動関数に戻します。

すべてのフロー構造は、フローをコントロールするブロックの終わりを示すのに end を使います。

注意 or や while ループをベクトル化されたコードで置き換えることにより、MATLAB コードの実行速度は速くなります。詳細は、詳細は、1-91 ページの "MATLAB コードの性能の最適化 " を参照してください。

---

if, else, elseif

if は、論理式を計算し、式の値をベースに一つのグループ化したステートメントを実行します。最も簡単な型として、つぎの書式で表現することができます。

```
if logical_expression
    statements
end
```

論理式が 1(真) ならば、MATLAB は if と end の間のすべてのステートメントを実行します。そして、end ステートメントのつぎのラインを実行します。条件が 0(

偽) ならば、MATLAB は if と end の間のすべてのステートメントをスキップし、end ステートメントのつぎのラインを実行します。

たとえば、

```
if rem(a,2) == 0
    disp('a is even')
    b = a/2;
end
```

if ステートメントは、複数個、入れ子にすることができます。

論理式が、スカラでない値に対して計算しようとすると、引数のすべての要素は非ゼロでなければなりません。例えば、X を行列とします。そして、ステートメント

```
if X
    statements
end
```

は、つぎのものと等価です。

```
if all(X(:))
    statements
end
```

else と elseif ステートメントは、if について再度条件分岐を行ないます。

- else ステートメントは、論理条件をもっていません。それに関連したステートメントは、if(または、可能なら elseif 条件) が偽 (0) の場合に実行します。
- elseif ステートメントは前に設定した if が 0(偽) ならば計算する論理条件をもっています。これに関連したステートメントは論理条件が 1 ならば実行します。また、if ブロックの中で複数の elseif を使うこともできます。

```
if n < 0              % In が負の場合、エラーメッセージが表示されます。
    disp('Input must be positive');
elseif rem(n,2) == 0 % n が正で偶数の場合、2 で割ります。
    A = n/2;
else
    A = (n+1)/2;      % n が正で奇数の場合、1 を足して 2 で割ります。
end
```

if ステートメントと空配列

空配列で省略した if 条件は、偽条件を表します。つぎのもの

```
if A
   S1
else
   S0
end
```

は、A が空配列のときステートメント S0 を実行します。

switch

switch は、一変数値または式の値を基に、あるステートメントを実行します。この基本型は、

```
switch expression（スカラ、または、文字列）
    case value1
         statements % Executes if expression is value1
    case value2
         statements % Executes if expression is value2
    .
    .
    .
    otherwise
         statements % Executes if expression does not
                        % does not match any case
end
```

このブロックは、つぎのものから構成されています。

- switch と言う言葉に、計算される式を続けます。
- いく組かの case グループがあります。これらのグループは、case のつぎに式に対して取り得る値が続きます。そして、これらはすべて単一ラインです。それに続く複数ラインは、式が与えられた値になったときに実行するステートメント群です。switch は、MATLAB が、つぎの case ステートメントまたは otherwise ステートメントに到達すると実行を終了します。最初に一致した case のみを実行します。
- オプションの otherwise グループ。これは、式の値が case のグループのどれとも一致しないときに実行するステートメントを後に続けます。otherwise グループの実行は、end ステートメントで終了します。
- 一つの end ステートメント

switch は、入力した式と各々の case の値とを比較します。数値式に対して、case ステートメントは、(value==expression)　　なら真です。文字式に対して、strcmp(value,expression) なら真です。.

つぎに示すコードは、switch ステートメントの簡単な例です。これは、ある値に対して、変数 input_num をチェックします。input_num が -1, 0, 1 のいずれかのとき、case ステートメントは、テキストとしてスクリーンに値を表示します。input_num が、これらの値のどれとも一致しないなら、otherwise ステートメントに行き、テキスト 'other value' を表示します。

```
switch input_num
    case –1
        disp('negative one');
    case 0
        disp('zero');
    case 1
        disp('positive one');
    otherwise
         disp('other value');
end
```

注意　C 言語の switch 構成と違って、MATLAB の switch は、各 case1 つしか処理を行ないません。これは、最初の case ステートメントが TRUE ならば、他の case ステートメントは実行されません。それゆえ、break ステートメントを使う必要はありません。

---

switch は、case の式をセル配列の中に入れることにより、単一の case ステートメントに複数の条件を取り扱うことができます。

```
switch var
    case 1
        disp('1')
    case {2,3,4}
        disp('2 or 3 or 4')
    case 5
        disp('5')
    otherwise
        disp('something else')
end
```

while

while ループは、コントロールされた式が 1(TRUE) の間、一つ、または、一つのグループ化されたステートメントを繰り返し実行します。つぎの書式で表します。

```
while expression
    statements
end
```

式に行列が含まれていると、実行を繰り返し続けるには、すべての要素が 1 でなければなりません。行列要素をスカラ値要素に減らすには、関数 all や any を使ってください。

例えば、つぎの while ループは、n! が 100 桁になる最初の整数 n を探すものです。

```
n = 1;
while prod(1:n) < 1e100
    n = n + 1;
end
```

while ループから外へ出るには、break ステートメントを使います。

while ステートメントと空配列

while に続く条件を省略すると、偽条件を表します。

```
while A, S1, end
```

は、A が空配列のとき、ステートメント S1 を実行しません。

for

for ループは、前もって設定した回数だけ一つのステートメント、または、一つのグループ化されたステートメントを実行します。つぎの書式を使います。

```
for index = start:increment:end
    statements
end
```

デフォルトの increment は 1 です。ユーザは任意の数字を設定することができ、負の値を設定することもできます。正のインデックスの場合、インデックスの値が end 値を超えると停止し、負の場合、end 値よりも小さくなると停止します。

例えば、つぎの例題は、5 回繰り返します。

```
for i = 2:6
    x(i) = 2*x(i-1);
end
```

for ループに対して入れ子構造を作成することができます。

```
for i = 1:m
    for j = 1:n
        A(i,j) = 1/(i + j -1);
    end
end
```

注意 for と while ループをベクトル化コードで置き換えることにより MATLAB の実行速度を高めることができます。詳細は、1-91 ページの "Vectorization of Loops" を参照してください。

インデックスとして配列を使用

for ループのインデックスとして配列を使うことができます。たとえば、m 行 n 列配列 A を考えましょう。ステートメント

```
for i = A
    statements
end
```

は、i をベクトル A(:,k) と設定します。最初のループの繰り返しに対して、k は、1 に等しく、2 番目の k は 2 に、と言うように k が n に等しくなるまで繰り返しま

す。ループは、A の中の列の数に等しい回数だけ繰り返します。各々の繰り返しに対して、i は、A の列の一つを含むベクトルです。

continue

ステートメント continue は、for、または、while ループのつぎの繰り返しまで、コントロールを続け、ループの本体の中の残りのステートメントをスキップします。入れ子のループでは、continue は、それを囲む for、または、while ループのつぎの繰り返しまで、コントロールを続けます。

つぎの例題は、continue ループを示すものでファイル magic.m の中のすべてのブランクラインとコメントをスキップして、ラインをカウントするものです。ステートメント continue は、ブランクラインやコメントラインが表れた場合に、カウントを行わないで、他の場合は、magic.m の中のライン数をカウントするために使います。

```
fid = fopen('magic.m','r');
count = 0;
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line) | strncmp(line,'%',1)
        continue
    end
    count = count + 1;
end
disp(sprintf('%d lines',count));
```

break

break ステートメントは、for ループや while ループの実行を停止します。break ステートメントに出会うと、実行はループの外のつぎのステートメントを実行します。入れ子になったループの中で、break は、最も内部のループのみを停止します。

つぎの例題は、ファイル fft.m の内容を MATLAB キャラクタ配列に読み込む while ループを示しています。break ステートメントは、最初の空ラインに出会うと、while ループを終了するために使用します。戻ってくるキャラクタ配列は、fft プログラムに対して、M- ファイルヘルプを含んでいます。

```
fid = fopen('fft.m','r');
s = '';
while ~feof(fid)
   line = fgetl(fid);
   if isempty(line)
      break
   end
   s = strvcat(s,line);
end
disp(s)
```

try ... catch

try ... catch ステートメントの一般的な型は、つぎのものです。

```
try,
   statement,
   ...,
   statement,
catch,
   statement,
   ...,
   statement,
end
```

ここで、示している try と catch の間のステートメント群は、エラーが生じると実行されます。catch と end の間のステートメントが、その後で、実行されます。lasterr を使って、エラーの原因を調べることができます。catch と end の間で、エラーが生じた場合、MATLAB は、他の try ... catch が構成されない限り、実行を停止します。

return

return は、コマンドのカレントの流れを停止し、コントロールを起動した関数、または、キーボードに戻します。return は、keyboard モードを停止するために使用します。コールされた関数は、通常、関数の最後に達した場合に、コントロールを、それを起動した関数に戻します。return は、コールされる関数の中に、途中で終了させるように挿入され、コントロールを起動した関数に転送します。

サブ関数

Function M- ファイルは、複数の関数に対するコードを含むことができます。ファイルの中の最初の関数を*メイン関数* と言い、関数は M- ファイル名で読み込みます。ファイルの中の付加的な関数は、*サブ関数* で、同じファイルの中のメイン関数またはサブ関数にのみ表れているものです。

各々のサブ関数は、自分自身が関数であることを示す関数定義ラインから始めます。関数は、お互いに続けて記述します。種々のサブ関数は任意の順で並べることができますが、必ずメイン関数が先頭です。

```
function [avg,med] = newstats(u) % メイン関数
% NEWSTATS Find mean and median with internal functions.
n = length(u);
avg = mean(u,n);
med = median(u,n);

function a = mean(v,n)            % サブ関数
% Calculate average.
a = sum(v)/n;

function m = median(v,n)          % サブ関数
% Calculate median.
w = sort(v);
if rem(n,2) == 1
    m = w((n+1)/2);
else
    m = (w(n/2)+w(n/2+1))/2;
end
```

サブ関数 mean,median は、入力リストの平均と中央値を計算します。メイン関数 newstats は、入力の長さ n を決め、サブ関数を読み込み、n をこれらに渡します。同じ M- ファイルの中の関数は、グローバル宣言をしない限り、また引数として渡さない限り、同じ変数をアクセスすることはできません。加えて、ヘルプ機能は、1 つの M- ファイルの中でメイン関数のみにアクセスできます。

関数を M- ファイルの中から読み込むと、MATLAB は、関数がサブ関数であるかどうかを知るためにファイルをチェックします。そして、その名前をもったプライベート関数かどうか、そして、ユーザのパス上にある標準のM-ファイルかをチェックします。まず、サブ関数に対して、チェックしているので、同じ名前をもつサブ関数を使っている既に存在する M- ファイルと取り替えます。たとえば、mean

は、このコードを使います。関数名は、その M- ファイルの中ではユニークなものでなければなりません。

プライベート関数

プライベート関数は、特別に設定した名前 private をもつサブディレクトリに保存された関数です。これらは、親ディレクトリの中で関数だけ見ることができます。たとえば、ディレクトリ newmath が MATLAB サーチパス上にあると仮定しましょう。private と呼ばれる newmath のサブディレクトリは、newpath の中の関数のみを呼ぶことができる関数を含んでいます。プライベート関数は、親ディレクトリ以外では見ることができないので、他のディレクトリの中の関数と同じ名前を使うことができます。これは、他のディレクトリにオリジナルのものを残したまま、特別な関数のユーザ自身のバージョンを作成するときに有効です。MATLAB が標準の M- ファイル関数の前にプライベート関数を探しますので、プライベートでない M- ファイル test.m の前にプライベート関数と名付けられた test.m を見つけます。

ユーザは、コンピュータ上にディレクトリ、または、フォルダを作成する標準的な手法を使って、private と呼ばれるサブディレクトリを作ることにより簡単にユーザ自身の private ディレクトリを作ることができます。ユーザのパスに、これらの priveta ディレクトリを設定してください。

サブスクリプトとインデックス

サブスクリプト

ここでは MATLAB 行列の要素にアクセスし、割り当てるためのサブスクリプトの使用法を説明します。以下について説明します。

- " 行列の連結 "
- " 行と列の削除 "

A の i 行、j 列の要素は、A(i,j) で定義されています。たとえば、A = magic(4) を考えましょう。ここで、A (4,2) は 4 行目の 2 列目の数値です。この場合、A (4,2) は 15 です。A の 4 列目の和は次式で計算されます。

```
A(1,4) + A(2,4) + A(3,4) + A(4,4)
```

これは、単一サブスクリプトをもつ行列 A(k) の要素と考えることもできます。これは、行ベクトル、列ベクトルを作成する一般的な方法です。しかし、フルの 2 次元行列に適用でき、この場合、配列はオリジナルの行列の列から作られる一つの長い列ベクトルと見做すことができます。それで、この例題では、A(8) は A(4,2) にストアされている 14 を表す、もう一つの方法です。

行列の要素以外を値を使おうとすると、エラーになります。

```
t = A(4,5)
Index exceeds matrix dimensions
```

一方、行列の要素以外の要素をストアすると、新しい結果はサイズを大きくしたものになります。

```
x = A;
x(4,5) = 17

x =

    16     2     3    13     0
     5    11    10     8     0
     9     7     6    12     0
     4    14    15     1    17
```

コロンを含んだサブスクリプト表現は、行列のある部分を取り出します。

```
A(1:k,j)
```

は、A の j 番目の最初の k 要素です。そして、

```
sum(A(1:4,4))
```

は、4 番目の列の和を計算しています。しかし、良い方法があります。コロン自身は、行列のある行または列の *すべての* 要素を表し、キーワード end は*最後の* 行または列を意味しています。それで、

```
sum(A(:,end))
```

は、A の最後の列の要素の和です。

```
ans =

    34
```

行列の連結

連結は、小さな行列を合わせてより大きな行列を作る過程です。実際に、最初の行列も各々の要素を連結したものです。鍵括弧が、連結演算を定義するものです。例題として、4 行 4 列の魔方陣 A から始め、つぎのものを作りましょう。

```
B = [A A+32;A+48 A+16]
```

結果は、4 つのサブ行列を連結することにより、8 行 8 列の行列が得られます。

```
B =

    16     2     3    13    48    34    35    45
     5    11    10     8    37    43    42    40
     9     7     6    12    41    39    38    44
     4    14    15     1    36    46    47    33
    64    50    51    61    32    18    19    29
    53    59    58    56    21    27    26    24
    57    55    54    60    25    23    22    28
    52    62    63    49    20    30    31    17
```

この行列は、他の魔方陣を作成する途中の結果です。この要素は整数で 1 から 64 までの数を再配置したものです。列の和は8行8列の魔方陣の和に等しいものです。

```
sum(B)
```

```
ans =

    260    260    260    260    260    260    260    260
```

しかし、行の和 sum(B')' は等しくなりません。正しい 8 行 8 列の魔方陣にするためには、さらに操作を加える必要があります。

行と列の削除

鍵括弧を使った行列から行や列を削除することができます。まず、つぎの行列を考えます。

```
X = A;
```

つぎに、X の 2 番目の列を削除しましょう。

```
X(:,2) = []
```

X は、つぎのように変化します。

```
X =
    16     3    13
     5    10     8
     9     6    12
     4    15     1
```

行列から単一要素を削除するとすると、結果は行列にはなりません。たとえば、e

```
X(1,2) = []
```

は、エラーになります。しかし、単一サブスクリプトは単一要素を削除したり、一連の要素を削除でき、残りの要素を行ベクトルに変形します。

```
X(2:2:10) = []
```

は、つぎの結果を得ます。

```
X =

    16     9     3     6    13    12     1
```

アドバンスドなインデックス

MATLAB は、実際の大きさに関係なく、値の列として各々の配列をストアしています。この列は、配列の列の後ろにつぎの列をくっ付けて、またくっ付けての繰り返しで作成しています。

たとえば、MATLAB がつぎの行列をストアしているとします。

```
A = [2 6 9; 4 2 8; 3 0 1]
```

は、つぎのようになります。

```
2
4
3
6
2
0
9
8
1
```

単一サブスクリプトインデックスにストレージ列の中に直接アクセスします。A(3) は、列の中の 3 番目にアクセスし、数値 3 になります。A(7) は、7 番目の値で、9 になります。

多くのサブスクリプトを与えると、MATLAB はユーザが設定した配列をベースにストレージ列の中のインデックスを計算します。たとえば、A を 2 次元配列 [d1 d2] と考えましょう。ここで、d1 は配列の行数で、d2 は列数です。行と列を表す 2 つのサブスクリプト (i,j) を与えると、オフセットはつぎのようになります。

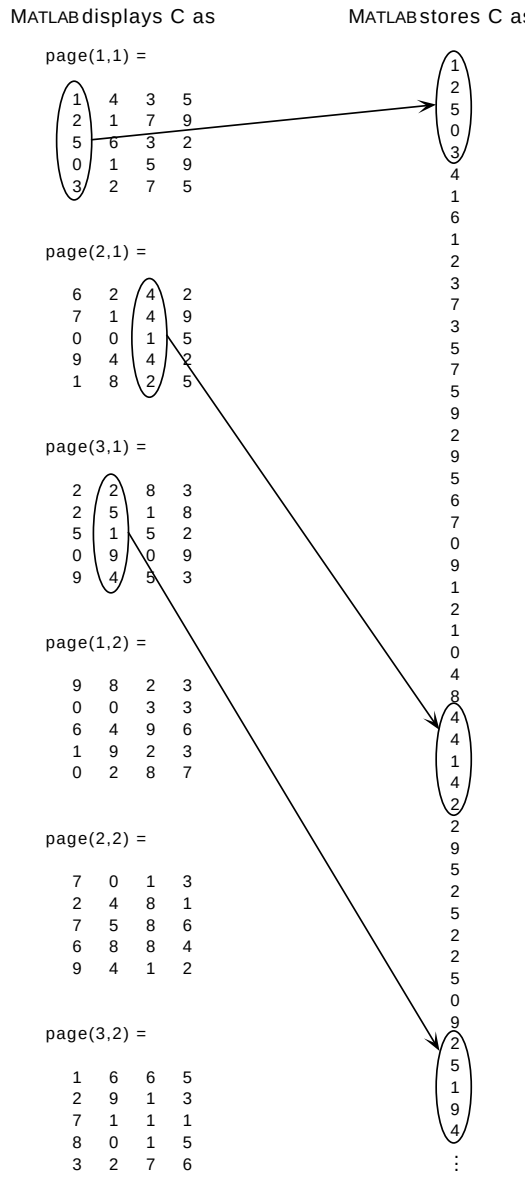
```
(j–1)*d1+i
```

A(3,2) を与えると、MATLAB は、A のストレージ列を (2-1)*3+3、または、6 になります。列の要素番号 6 を見ると、0 になります。

多次元配列へのインデックス

このストレージやインデックス法を多次元配列に拡張します。この場合、MATLAB はストレージ列を作成するページをベースに機能し、列方向に要素を付け加えて行きます。

例えば、5-4-3-2 の配列 C を考えましょう。



再び、単一サブスクリプトは、この列の中に直接インデックスを付けます。例えば、C(4) は、つぎの結果になります。

```
ans =

     0
```

2つのサブスクリプト (i,j) を使って行と列のインデックスを設定すると、MATLAB は上で記述したようにオフセットを計算します。2 つのサブスクリプトは、オリジナルの配列次元の範囲で設定されているなら、多次元配列の最初のページを常にアクセスします。

複数のサブスクリプトが存在すると、すべてのサブスクリプトがオリジナル配列の次元内になければなりません。たとえば、C は 5 つの行しかないので、C(6,2) はまちがいです。

2 つ以上のサブスクリプトがあれば、MATLAB はそのインデックスの設定法に従って、拡大していきます。たとえば、4 次元配列の中でサイズ [d1 d2 d3 d4] をもつ 4 つのサブスクリプト (i,j,k,l) を考えます。MATLAB は、つぎのようにストレージ列の中でオフセットを計算します。

```
(l–1)(d3)(d2)(d1)+(k–1)(d2)(d1)+(j–1)(d1)+i
```

たとえば、サブスクリプト (3,4,2,1) を使って配列 C のインデックスは、値 5 を出力します (ストレージ列の中の 38 番目)。

一般的に、サブスクリプト (s_1 s_2 s_3 ... s_n) を使った次元 [d_1 d_2 d_3 ... d_n] をもつ配列に対するオフセットの式は、

$$(s_n-1)(d_{n-1})(d_{n-2})...(d_1)+(s_{n-1}-1)(d_{n-2})...(d_1)+...+(s_2-1)(d_1)+s_1$$

この方法により、サブスクリプトの番号を使って配列にインデックスを付けることができます。1 と言う数字をもったものをサブスクリプトリストに付けるには、0 になるので、

```
C(3,2,1,1,1,1,1,1)
```

は、つぎのものと等価です。

```
C(3,2)
```

文字列の計算

文字列計算は、MATLAB 言語に付加された強力な柔軟性のある機能です。これは、ユーザが設定した文字列を実行したり、実行可能な文字列を作成したりするような演算が可能になります。

eval

関数 eval は、MATLAB の式、ステートメント、または、関数コールを含む文字列を計算します。最も簡単な型は、

```
eval('string')
```

です。たとえば、このコードは、n 次の Hilbert 行列を作成する式を eval を使って実行するものです。

```
t = '1/(i+j-1)';
for i = 1:n
    for j = 1:n
        a(i,j) = eval(t);
    end
end
```

つぎの例は、ステートメントを eval を使って実行するものです。

```
eval('t = clock');
```

計算用の文字列の作成

eval への入力を文字列を連結することにより作成します。このコードは、変数名 P1, P2, ...P10 で設定された 10 個の変数を作成し、それらの各々に異なる値を設定する方法を示しています。

```
for i=1:10
    eval(['P',int2str(i),'= i.^2'])
end
```

feval

feval は、MATLAB の式と言うより、むしろ関数名を文字列に含ませ、それを実行することができる点が eval と異なります。実行される関数は、関数ハンドルまたは関数名を含む文字列のいずれかによって最初の引数で指定されます。

ユーザは、feval と input を使って、M- ファイルにより定義されるいくつかの仕事の中の一つを選択することができます。つぎの例では、関数は、sin,cos 等々のような名前をもっています

```
fun = [@sin; @cos; @log];
k = input('Choose function number: ');
x = input('Enter value: ');
feval(fun(k),x)
```

注意 可能な場合は、eval よりも feval を使ってください。feval を使う M- ファイルは、高速に実行され、MATLAB コンパイラを使ってコンパイルすることができます。

コマンド / 関数の双対性

MATLAB コマンドは、

```
load
help
```

のようなものです。ほとんどのコマンドは、オペランドを設定した修飾子と共に使います。

```
load August17.dat
help magic
type rank
```

コマンド修飾子を設定する別の方法は、関数の文字引数にすることです。

```
load('August17.dat')
help('magic')
type('rank')
```

これが、MATLAB の *"コマンド / 関数の双対性"* です。つぎの型のコマンド

```
command argument
```

は、つぎの関数型で記述することができます。

```
command('argument')
```

関数アプローチの利点は、文字引数が他のものから作られるときにわかります。つぎの例題は、複数データファイル August1.dat, August2.dat 等々を処理するものです。この例題は、整数を文字キャラクタに変換する関数 int2str を使ってファイル名を作成します。

```
for d = 1:31
   s = ['August' int2str(d) '.dat']
   load(s)
% Process the contents of the d-th file
end
```

空行列

少なくとも 1 つの次元がゼロである行列は、*空行列* と呼ばれます。最もシンプルな空行列は、0 行 0 列です。より複雑な例は、0 行 5 列または 10-by-0-by-20 です。

0 行 0 列行列を作成するには、値を指定せずにブラケット演算子を使います。

```
A = [];

whos A
  Name      Size          Bytes  Class

  A         0x0               0  double array
```

空配列は、関数 zeros，ones，rand，eye を使って作成できます。たとえば、0 行 5 列の行列を作成するには、つぎのようにします。

```
E = zeros(0,5)
```

空行列の作業

空行列に関する基本的なモデルは、m 行 n 列の行列に対し設定される任意の演算で、m と n が結果の次元で、m または n のどちらかがゼロでも可能です。結果の大きさは、同じ関数で計算したものと同じで、結果の内容がゼロで見積もられたものです。

たとえば、水平方向への連結

```
C = [A B]
```

は、A と B の行数が等しいことが必要です。それで、A が m 行 n 列で、B が m 行 p 列ならば、C は m 行 (n+p) 列になります。そして、m,n,p のいずれかがゼロでも成り立ちます。

MATLAB の中のほとんどの演算は、行ベクトルまたは列ベクトルを生みます。これは、結果が空行ベクトルになる場合も可能です。

```
r = zeros(1,0)
```

または、空列ベクトル

```
C = zeros(0,1)
```

MATLAB の中のすべての行列を使って、次元に関する整合性に関するルールが成り立ちます。つぎの例題の中で、1 行 3 列の行列を 0 行 3 列の空行列を付加しようとする試みは、エラーになります。

```
[1 2 3] + ones(0,3)

??? Error using ==> +
Matrix dimensions must agree.
```

sum,max のようないくつかの MATLAB 関数は、次元を減らす機能をもっています。行列を引数とすると、これらの関数は、結果をベクトルとして出力します。また、ベクトルを入力すると、スカラを出力します。空を入力すると、つぎのような結果になります。

- sum([ ]) は 0
- prod([ ]) は 1
- max([ ]) は [ ]
- min([ ]) は [ ]

If または While の空行列の利用

if または while ステートメントの式の部分が空行列である場合は、MATLAB は式を false として評価します。以下の例は、A が空配列なので、ステートメント S0 を実行します。

```
A = ones(25,0,4);

if A
   S1
else
   S0
end
```

エラーとワーニング

多くの場合、種々のエラーが生じたときに、特別の反応を示すことが望まれます。たとえば、入力がもっと必要ならばユーザに入力するためのプロンプトが表示されたり、より詳しいエラーやワーニングメッセージが表示されたり、デフォルト値を使って計算を再度行ったりすることです。MATLABのエラーを取り扱う機能は、特別なエラー状況をアプリケーションにチェックさせ、その状況に従って適切なコードを実行します。

eval や lasterr を使ったエラーの取り扱い

MATLAB の中でのエラーの取り扱いに関する基本的なツールは、:

- 関数 eval は、関数を実行し、エラーが最初に生じると実行するために 2 番目の関数を使います。
- lasterr は、MATLAB により生じる最も新しいエラーを含んだ文字列を出力します。

関数 eval は、2 つの引数を使った型でエラー取り扱い機能を与えます。

```
eval ('trystring','catchstring')
```

trystring により設定される演算が適切に機能するなら、eval は単に結果を出力します。trystring がエラーを発生させると、関数は catchstring を実行します。catchstring を使って、trystring により発生するエラーを決め、適切に動作するような関数を設定します。

trystring/catchstring 型の eval は、関数 lasterr と共に使うと特に有効です。lasterr は、MATLAB により発生する最新のエラーメッセージを含んだ文字を出力します。trystring により発生するエラーを " 捉える " ための関数 catchstring の内部の lasterr を使います。

たとえば、この関数は、行列乗算の間に生じる固有のエラーメッセージに対してチェックするために lasterr を使います。エラーメッセージは、両オペランドの内側の大きさが異なるため行列計算が不可能であることを示しています。メッセージが生じたなら、コードは乗算を適切に行なうため、行列の一方の大きさを整合性をもつように小さくします。

```
function C = catchfcn(A,B)
l = lasterr;
j = findstr(l,'Inner matrix dimensions')
if (~isempty(j))
```

```
    [m,n] = size(A)
    [p,q] = size(B)
    if (n>p)
        A(:,p+1:n) = []
    elseif (n<p)
        B(n+1:p,:) = []
    end
    C = A*B;
else
    C = 0;
end
```

eval の 2 つの引数をもった型を関数 catchfcn と共に使います。

```
clear
A = [1  2  3; 6  7  2; 0  1  5];
B = [9  5  6; 0  4  9];
eval('A*B','catchfcn(A,B)')

A = 1:7;
B = randn(9,9);
eval('A*B','catchfcn(A,B)')
```

エラーメッセージとワーニングメッセージの表示

関数 error と fprintf を使って、スクリーン上にエラーメッセージを表示します。関数 error は、つぎの書式を使います。

```
error('error string')
```

M- ファイルの中から関数 error を呼ぶと、error は引用符で囲まれた文字列を表示し、M- ファイルの実行を停止します。たとえば、M- ファイル myfile.m の中で、エラーが生じたときに、つぎのステートメントが現れるとします。

```
if n < 1
  error('n must be 1 or greater.')
end
```

n が 0 のとき、つぎの文字がスクリーン上に現れ、M- ファイルが停止します。

```
??? Error using ==> myfile
n must be 1 or greater.
```

MATLAB の中で、ワーニングメッセージは、エラーメッセージと同じようなものです。しかし、ワーニングメッセージは、プログラムを停止することはありません。関数 warning を使って、ワーニングメッセージを表示することができます。

```
warning('warning string')
```

関数 lastwarn は、MATLAB により出力する最新のワーニングメッセージを表示します。

時刻と日付

MATLAB は、時刻と日付を取り扱う関数を用意しています。これらの関数は、MATLAB ツールボックスの timefun と名付けたディレクトリに保存されています。

カテゴリ	関数名	詳細
現在の時刻、日	clock	日付を単位に現在の日付と時刻
	date	日付文字として現在の日付
	now	日付ベクトルとして現在の日付と時刻
変換	datenum	連続型数字に変換
	datestr	日付の文字表現を使った変換
	datevec	日付要素
ユーティリティ	calendar	カレンダ
	datetick	書式付きでラベルを作成
	eomday	月の最終日
	weekday	曜日
タイミング	cputime	秒単位で CPU 時刻
	etime	経過時間
	tic, toc	ストップウォッチタイマ

日付書式

ここでは、以下のトピックスを説明します。

- " 日付書式のタイプ "
- " 日付書式間の変換 "
- " 日付文字書式 "
- " 出力書式 "

日付書式のタイプ

MATLAB は、3 つの異なる日付書式、すなわち、日付文字、ある基準から数えた連続日付番号、日付ベクトルを用意しています。

日付を扱うとき、一般には日付文字 (16-Sep-1996) を使います。MATLAB は内部では*連続日付番号* (729284) を使います。連続日付番号は、ある固定した日付より日を単位とした日数で表します。MATLAB では、連続日付番号の 1 は、西暦 0 年の 1 月 1 日を基準にしています。MATLAB は、日を単位として分数で時刻を表します。例えば、午後 6 時は 0.75 です。そして、'16-Sep-1996, 6:00 pm' は MATLAB では 729284.75 です。

日付を必要とするすべての関数は、日付文字、または、連続日付番号のどちらでも使えます。MATLAB コマンドラインレベルで日付を取り扱うとき、数が少なければ、日付文字の方が便利です。日付の大きな数字を取り扱い関数を使う場合や日付をより多くの計算に使おうとすると、日付番号を使ったほうが便利です。

日付ベクトルは、いくつかの MATLAB 関数では内部書式になっており、計算の中では使うことができません。日付ベクトルは、[year month day hour minute second] の要素から構成されています。

MATLAB は、日付文字を連続日付番号に変換し、また逆変換もできます。日付は日付ベクトルにも変換できます。

つぎに、MATLAB で使われる 3 つの日付書式の例を示します。

日付書式	例
日付文字	02-Oct-1996
連続番号	729300
日付ベクトル	1996 10 2 0 0 0

日付書式間の変換

日付書式変換を行なう関数を示します。

関数	詳細
datenum	日付文字から連続日付番号に変換

関数	詳細
datestr	連続日付番号を日付文字に変換
datevec	日付番号または日付文字列を個別の日付要素に分割

つぎに、ある日付書式を他の日付書式に変換する例を示しています。

```
d1 = datenum('02-0ct-1996')

d1 =

    729300

d2 = datestr(d1+10)

d2 =

12-0ct-1996

dv1 = datevec(d1)

dv1 =

    1996     10     2     0     0     0

dv2 = datevec(d2)

dv2 =

    1996     12     2     0     0     0
```

日付文字書式

関数 datenum は、日付計算を行なうために重要なものです。datenum は、種々の書式で入力文字を使うことができ、'dd-mmm-yyyy', 'mm/dd/yyyy', 'dd-mmm-yyyy, hh:mm:ss.ss' を入力文字として受け入れます。他のキャラクタにより分離された文字や数字から6個のフィールドまでの型を作ることができます。

- 日に関するフィールドは、1 から 31 までです。

- 月に関するフィールドは、1 から 12 までの整数か、または少なくとも 3 文字を使ったアルファベットです。
- 年に関するフィールドは、非負の整数で、2 桁のみの整数が設定されていると、年は 19yy を仮定しています。年が省略されていると、デフォルトとして現在の年が使われています。
- 時、分、秒フィールドはオプションです。これらは、コロンで分離されている整数か、または、'AM'、または、'PM' に続く整数です。

たとえば、現在の年が 1996 ならば、つぎのものはすべて等価です。

```
'17-May-1996'
'17-May-96'
'17-May'
'May 17, 1996'
'5/17/96'
'5/17'
```

また、つぎの表現もお互い同じ時刻を表しています。

```
'17-May-1996, 18:30'
'5/17/96/6:30 pm'
```

数字のみの入力に対するデフォルトの書式は米国標準を使い、3/6 は、3 月 6 日で、6 月 3 日ではありません。

入力日付文字のベクトルを作成するならば、列ベクトルを使って、すべての文字は同じ長さであることをチェックしてください。長さがバラバラの場合、スペースかゼロを設定してください。

出力書式

関数 datestr(D,dateform) は、連続日付 D を 19 種類の日付文字出力書式のいずれかを使って日、時を表します。デフォルトの出力は日 - 月 - 年文字：01-Mar-1996 です。オプション整数引数 dateform を使って出力書式を選択してください。

つぎの表は、各 dateform の値に対応する日付文字列のフォーマットを示します。

dateform	フォーマット	詳細
0	01-Mar-1996 15:45:17	日 - 月 - 年 時 : 分 : 秒
1	01-Mar-1996	日 - 月 - 年

dateform	フォーマット	詳細
2	03/01/96	月 / 日 / 年
3	Mar	月、3 文字
4	M	月、1 文字
5	3	月
6	03/01	月 / 日
7	1	月の中の日
8	Wed	曜日 (3 文字)
9	W	曜日 (1 文字)
10	1996	年 (4 文字)
11	96	年 (2 文字)
12	Mar96	月と年
13	15:45:17	時 : 分 : 秒 S
14	03:45:17 PM	時 : 分 : 秒 AM または PM
15	15:45	時 : 分
16	03:45 PM	時 : 分 AM または PM
17	Q1-96	4 半期カレンダ - 年
18	Q1	4 半期カレンダ

つぎに 1996 年 4 月 1 日を、関数 datestr を使って、種々に変換する例を示します。

```
d = '01-Mar-1999'

d =

01-Mar-1999

datestr(d)

ans =

     01-Mar-1999

datestr(d, 2)

ans =

     03/01/99

datestr(d, 17)

ans =

     Q1-99
```

現在の日付と時刻

関数 date は、本日の日付を文字で出力します。

```
date

ans =

     02-Oct-1996
```

関数 now は、現在の日付と時刻を連続日付番号を出力します。

```
now

ans =

      729300.71

datestr(now)

ans =

      02-Oct-1996 16:56:16

datestr(floor(now))

ans =

      02-Oct-1996
```

ユーザ入力の取得

M- ファイルを実行中にユーザから入力を行なうには、つぎの方法があります。

- プロンプトを表示し、キーボード入力
- ユーザがキーを押すまでポーズ
- グラフィカルユーザインタフェースを作成

この節は、最初の 2 つのトピックスを取り扱います。3 番目のトピックは、"Creating Graphical User Interfaces" のオンラインドキュメントで説明されています。

キーボード入力に対するプロンプト

関数 input は、プロンプトを表示し、ユーザからの入力を待ちます。つぎの書式で表します。

```
n = input('prompt_string')
```

関数は、*prompt_string* を表示し、キーボードからの値の入力を待っています。ユーザが式を入力すると、関数はそれを計算し、その値を入力します。この関数は、メニューを使ったアプリケーションを実行するときに有効です。

input は、数値よりも文字列としてユーザの入力を返します。文字入力を得るには、's' を関数引数の後に付けます。

```
name = input('Enter address: ','s');
```

実行中に一時停止

実行ステップ中に一時停止を実行し、その間にいくつかの種類の M- ファイルを利用することができます。たとえば、簡単なスクリプトの例題に示しているpetals.m スクリプトは、個々のプロットの間は一時停止し、つぎのプロットに移動するには、任意のキーを押す必要があります。

引数をもたない pause コマンドは、ユーザがキーを押すまで実行を停止しています。n 秒間停止するには、

```
pause(n)
```

とします。

シェルエスケープ機能

*シェルエスケープ機能*を使って、ユーザ自身の C、または、Fortran プログラムにアクセスすることが有効な場合があります。シェルエスケープ機能は、シェルエスケープコマンド！を使って、外部のスタンドアロンプログラムを新しい MATLAB 関数として機能させることができます。シェルエスケープ M- 関数は

1. ディスクに適切な変数をセーブします。
2. 外部プログラムを実行します (データファイルを読み、データを処理し、結果をディスクに書き込みます)
3. 処理されたファイルからワークスペースにロードし直します。

を行なう M- ファイルです。たとえば、つぎの garfield.m のコードを見てみましょう。この関数は、外部関数gareqnを使って、Garfield方程式の解を求めるものです。

```
function y = garfield(a,b,q,r)
save gardata a b q r
!gareqn
load gardata
```

この M- ファイルは、つぎの手順で処理します。

1. ワークスペース内の入力引数a, b, q, r を save コマンドを使って、MAT- ファイルにセーブします。
2. シェルエスケープ演算子を使って、gareqn と名付けた C、または、Fortran プログラムを読み込みます。そして、この計算を行なうためにワークスペース変数を使います。gareqn は、MAT- ファイル gareqn に結果を書き出します。
3. gardata MAT- ファイルをロードして、結果を得ます。

MATLAB コードの性能の最適化

この節は、MATLABコードの実行速度やメモリの管理を改良する手法を記述します。

- ループのベクトル化
- ベクトルの占有容量を前もって確保

MATLAB は、行列言語で、ベクトルや行列演算用に設計されています。最も良い性能を得るため、可能な部分で、これらのことを利用してください。

メモリの節約とメモリ使用量の改良の方法については、17-71 ページの " メモリ使用に関する注意 " を参照してください。

ループのベクトル化

アルゴリズムをベクトル化することにより、ユーザの M- ファイルをスピードアップすることができます。ベクトル化は、for や while を等価なベクトル、または、行列に変換するものです。

簡単な例題

つぎのコードは、0 から 10 の範囲で 1001 値の正弦値を計算する一つの方法です。

```
i = 0;
for t = 0:.01:10
    i = i+1;
    y(i) = sin(t);
end
```

同じコードのベクトル化バージョンは、

```
t = 0:.01:10;
y = sin(t);
```

2 番目の例は、最初のものより実行スピードがアップし、MATLAB は、この方法を使います。示されるコードを M- ファイルスクリプトとして作成し、ユーザのシステム上でテストしてください。このとき、コマンド tic と toc を使って計算時間を測ってください。

アドバンストな例題

repmat は、ベクトル化を利用した関数の例です。これは、3 つの入力引数、すなわち、配列 A、行次元 M、列次元 N を受け入れます。

repmat は、配列 A の要素を含んだ出力配列を作成します。これは、M 行 N 列の大きさの中にタイルを並べるように配列したものです。

```
A = [1 2 3; 4 5 6];
B = repmat(A,2,3);

B =

    1    2    3    1    2    3    1    2    3
    4    5    6    4    5    6    4    5    6
    1    2    3    1    2    3    1    2    3
    4    5    6    4    5    6    4    5    6
```

repmat は、出力配列の中の要素の位置を作成するためにベクトル化を使っています。

```
function B = repmat(A,M,N)
if nargin < 2
   error('Requires at least 2 inputs.')
elseif nargin == 2
   N = M;
end

% Step 1 行列の大きさを得る
[m,n] = size(A);

% Step 2 1から行 / 列のサイズまで、インデックスのベクトル
mind = (1:m)';
nind = (1:n)';

% Step 3 上のベクトルからインデックス行列の作成
mind = mind(:,ones(1,M));
nind = nind(:,ones(1,N));

% Step 4 出力配列を作成
B = A(mind,nind);
```

Step 1 は、入力配列の行と列のサイズを得ます。

Step 2 は、2 列のベクトルを作成します。mind は、1 から A の行のサイズまでの整数を含んでいます。nind は、1 から A の列サイズまでの整数を含んでいます。

Step 3 は、MATLAB のベクトル化のトリックを使って、データの 1 つの列を繰り返し使うことにより希望する列の行列を作成します。コードは、つぎのようになります。

```
B = A(:,ones(1,n_cols))
```

ここで、n_cols は、行列の希望する結果生じる列の数です。

Step 4 は、配列のインデックスを使って出力配列を作成します。行のインデックス配列 mind の各要素は、以下の手順を使って列のインデックス配列 nind の各列と組になります。

1 行のインデックス mind の最初の要素は、nind の各々の要素と組み合わせています。MATLAB は、列方向に nind 行列を移動し、そして、mind(1,1) は nind(1,1) と共に移動し、つぎに nind(2,1) と、等々と共に移動します。結果は、出力配列の最初の行に入ります。

2 mind により列方向に移動し、各々の要素は、上のように nind の要素と共に移動します。nind 行列を通して完全に移動したものは、出力配列の一行を満たします。

配列を前もって割り当てる

出力結果をストアする配列を前もって割り当てることによりコードの実行時間を改良することができます。前もって割り当てることは、結果が大きくなる度に、MATLAB が配列をリサイズすることを必要としません。配列の種類に対して、つぎのような適切な配列割り当て関数を使います。

配列タイプ Type	関数	例題
数値配列	zeros	`y = zeros(1,100);`

e セル配列	cell	`B = cell(2,3);`
		`B{1,3} = 1:3;`
		`B{2,2} = 'string';`

構造体配列	struct,repmat	`data = repmat(struct('x',[1 3],...` `'y',[5 6]), 1, 3);`

前もって配列を割り当てることは、大きな行列を扱うとき、メモリの分割を少なくします。MATLAB の実行中、メモリは、ダイナミックなメモリの割り当てと削除により小さく分割されます。このため、フリーメモリが十分必要になり、しかし、大きな変数を保存するために十分な連続したスペースを取ることは困難です。配列を前もって割り当てることは、計算の始めに大きなデータに対して十分なスペースを確保することにより、このことを防ぐことになります。

メモリ使用に関する注意

この節は、メモリを保護したり、メモリの使い方を改良する方法を議論します。

- " メモリ管理関数 "
- " メモリから関数の削除 "
- " 入れ子になった関数の読み込み "
- " 変数とメモリ "
- "PC 固有の事柄 "
- "UNIX 固有の事柄 "
- " メモリアウトが意味するものは ?"

メモリ管理関数

MATLAB は、メモリの取り扱いを行なう 5 つの関数を用意しています。

- clear は、メモリから変数を取り除きます。
- pack は、存在している変数をディスクにセーブし、これらを一つのまとまったものとして再ロードします。時間を必要とするため、ループや、または、M-ファイル関数の中では、pack を使わないでください。
- quit は、MATLAB を終了し、すべての割り当てられているメモリをシステムに戻します。
- save は、選択した変数をディスクにセーブします。
- load は、save コマンドを使ってセーブしたデータファイルを再ロードします。

注意 save と load は、MATLAB の低水準ファイル I/O ルーチンより速く処理します。save と load は、速く実行し、メモリの分割を少なくします。

いくつかのシステムにおいて、whos コマンドは、残っているフリーメモリの総量を表示します。しかし、つぎのことは、知っておいてください。

- ワークスペースから変数を削除するなら、whos により示されるフリーメモリの総量は、削除される変数が最も大きいメモリアドレスを占めていない限り変化しません。実際に、数字は、一塊の使っていないメモリの量を示しています。最も大きい部分を消去することは、フリーメモリを大きくしますが、最も大きい部分を占める部分の周りの変数を消去することは、なにも影響を与えません。このことは、ユーザは、実際に whos により示されるものよりも多くのフリーメモリをもっていることを意味しています。
- 仮想メモリをもつコンピュータは、MATLAB でも他のハードウエアでも制限を課していないので残りのフリーメモリの量を表示しません。

メモリから関数の削除

MATLAB は、起動時に、matlab/toolbox ディレクトリの下に位置するすべてのファイルに対する M- ファイルと MEX- ファイルの一覧表を作成します。この表は、メモリにストアされ、関数 path で読み込まれ新しい一覧表が作成されるときのみに書き換えられます。ファンクション M- ファイルコードと MEX- ファイルで再設定されたコードは、対応する関数が読み込まれたとき、メモリにロードされます。M- ファイルコード、または、再割り当てされるコードは、つぎの場合にメモリから削除されます。

- 関数が再度読み込まれるか、または、新しいバージョンが既に存在している。
- 関数が、clear コマンドで完全にクリアされる。
- すべての関数が、clear functions コマンドを使って削除される。
- MATLAB がメモリアウトになる。

入れ子になった関数の読み込み

入れ子になった関数により使われるメモリの総量は、それらを連続的に読み込んで使うメモリの総量と同じです。つぎの二つの例題は、同じメモリを必要としています。

```
result = function2(function1(input99));

result = function1(input99);
result = function2(result);
```

変数とメモリ

メモリは、代入式の中の左辺変数がメモリ内に存在しないときは、いつでも変数に対して割り当てられます。ステートメント

```
x = 10
```

は、メモリを割り当てますが、ステートメント

```
x(10) = 1
```

は、x の 10 番目の要素が存在するなら、メモリの割り当ては行ないません。

メモリを保護するため、

- 大きなテンポラリ変数を作成することを避けます。そして、必要なくなれば、テンポラリ変数をクリアしてください。
- 入力と出力と同じ変数を関数に使わないようにします。これらは、リファレンスとして重ね書きされます。例えば、

  ```
  y = fun(x,y)
  ```

  は、y が入力と出力であるので、好ましいものではありません。
- 変数を空行列に設定してフリーメモリにするか、

  ```
  clear variable_name
  ```

を使って、削除します。

- できるだけ変数を再使用します。

グローバル変数． 変数を global として宣言することは、単にシンボルテーブルの中でフラグが付くだけです。これは、グローバル変数を定義しないものより多くのメモリを使うものではありません。つぎの例題を考えましょう。

```
global a
a = 5;
```

さて、MATLAB ワークスペースの中に a のコピーを一つストアしたものです。つぎのようにタイプインします。

```
clear a
```

は、MATLAB ワークスペースから a を取り除きます。しかし、global ワークスペースの中には、まだ存在しています。

```
clear global a
```

により、グローバルワークスペースより削除されます。

PC 固有の事柄

- MATLAB を使って、Microsoft Windows システムを操作する関数はありません。Windows はフォント、ウインドウ、スクリーンオブジェクトを追跡するためにシステムリソースを使います。リソースは、マルチフィギュアウインド、マルチフォント、またはいくつかの Uicontrol を使います。システムリソースをフリーアップする最も良い方法は、すべての使っていないウインドウを閉じることです。アイコン化されたウインドウは、まだリソースとして使用しています。
- パーマネントスワップファイルの性能は、テンポラリスワップファイルよりも良いものです。
- スワップファイルは、インストールされている RAM のサイズの 2 倍で十分です。

UNIX 固有の事柄

- オペレーテングシステムから必要とされる MATLAB のメモリは、MATLAB のプロセスが終了するまでオペレーテングシステムには戻りません。
- MATLAB は、カレントの変数をストアするために十分なメモリ使用ができない場合、オペレーテングシステムからメモリを必要とします。必要とされるメモリセグメントのサイズが MATLAB ヒープの中で利用可能な限り、ヒープの中のメモリを再利用します。

  たとえば、あるマシン上で、つぎのステートメントは、約 15.4MB の RAM を使います。

  ```
  a = rand(1e6,1);
  b = rand(1e6,1);
  ```

  つぎのものは、約 16.4MB の RAM を必要とします。

  ```
  c = rand(2.1e6,1);
  ```

そして、つぎのステートメントは、約 32.4MB の RAM を必要とします。

```
a = rand(1e6,1);
b = rand(1e6,1);
clear
c = rand(2.1e6,1);
```

これは、MATLAB が 2 つの 1MB の配列により前もって占められるスペースに 2.1MB の配列をフィットすることができないためです。メモリのオーバアロケーションを妨げる最も簡単な方法は、最大ベクトルを前もって割り当てることです。つぎの一連のステートメントは、約 32.4MB の RAM を使います。

```
a = rand(1e6,1);
b = rand(1e6,1);
clear
c = rand(2.1e6,1);
```

一方、つぎのステートメントは、約 16.4MB の RAM を使います。

```
c = rand(2.1e6,1);
clear
a = rand(1e6,1);
b = rand(1e6,1);
```

まず、最長ベクトルを割り当てることは、利用可能なメモリの最適な使用を行なうことになります。

メモリアウトが意味するものは？

Out of Memory メッセージは、MATLAB がカレントで利用可能なものよりも多くのメモリをオペレーテングシステムに要求したために生じるものです。この節で議論されているいくつかの手法を使って、利用可能なメモリを最適化するようにしてください。Out of Memory メッセージが表示されたなら、

- スワップファイルのサイズを増やす
- MATLAB にアクセスできるメモリ上の外部からの制約がないことを確認します (UNIX システムでは、チェックのために limit コマンドを使います)
- システムにより多くのメモリを付加
- ユーザが使っているデータのサイズを小さくします

のいずれかを行ってください。

18

キャラクタ配列 (文字列)

キャラクタ配列 18-5
キャラクタ配列の作成 18-5
2 次元キャラクタ配列の作成 18-6
キャラクタを数値に変換 18-7

文字列のセル配列 18-8
文字列のキャラクタ配列とセル配列との間の変換 . . . 18-8

文字列の比較 18-10
文字列の等価性の比較 18-10
演算子を使ってキャラクタのチェック 18-11
文字列内のキャラクタ分類 18-12

検索と置換 18-13

文字列 / 数字の変換 18-15
配列 / 文字列の変換 18-16

この章では、MATLAB の文字列データのサポートについて説明します。MATLAB が文字列を表現する 2 つの方法について説明します。

- " キャラクタ配列 "
- " 文字列のセル配列 "

文字列に関して行う操作についても以下のトピックスを説明します。

- " 文字列の比較 "
- " 検索と置換 "
- " 文字列 / 数字の変換 "

つぎの表は、MATLAB Toolboxのstrfunディレクトリにある文字列関数の一覧です。

カテゴリ	関数	詳細
一般的なもの	blanks	ブランク文字列
	cellstr	キャラクタ配列から文字列のセル配列を作成
	char	キャラクタ配列の作成 (文字列)
	deblank	後ろに続くブランクを取り除く
	eval	MATLAB 表現を使った文字列の実行
文字チェック	iscellstr	文字列のセル配列に対して真
	ischar	キャラクタ配列に対して真
	isletter	アルファベット文字に対して真
	isspace	スペースキャラクタに対して真

カテゴリ	関数	詳細
文字演算	findstr	お互いの中の任意の文字列の検索
	lower	小文字に変換
	strcat	文字列の連結
	strcmp	文字列の比較
	strcmpi	大文字と小文字を無視して文字列を比較
	strjust	入力が文字列ならばそのまま出力
	strmatch	文字列に対して一致しているものの検索
	strncmp	文字列の最初の N キャラクタを比較
	strncmpi	文字列を無視して最初の N キャラクタを比較
	strrep	文字列の置き換え
	strtok	文字列の中のトークンの検索
	strvcat	文字列を垂直的に連結
	upper	大文字に変換
文字列と数字の変換	double	文字列を数値コードに変換
	int2str	整数を文字列に変換
	mat2str	行列を実行可能な文字列に変換
	num2str	数字を文字列に変換
	sprintf	書式付きデータを文字列に変換
	str2double	文字列を倍精度値に変換
	str2mat	文字列からキャラクタ行列を作成
	str2num	文字列を数値に変換
	sscanf	書式を制御して文字列の読み込み

カテゴリ	関数	詳細
基底数の変換	base2dec	基底 B 文字列を 10 進整数に変換
	bin2dec	バイナリ文字列を 10 進整数に変換
	dec2base	10 進整数を基底 B 文字列に変換
	dec2bin	10 進整数をバイナリ文字列に変換
	dec2hex	10 進の整数を 16 進に変換
	hex2dec	16 進文字列を 10 進の整数に変換
	hex2num	IEEE16 進を倍精度数に変換

キャラクタ配列

MATLAB では、用語*文字列* は、キャラクタの配列とします。MATLAB は、各キャラクタを内部で、対応する数値として表現します。しかし、これらの値を直接アクセスしない限り、スクリーン上で表示するように単にキャラクタとして取り扱うことができます。

ここでは以下について説明します。

- " キャラクタ配列の作成 "
- "2 次元キャラクタ配列の作成 "
- " キャラクタを数値に変換 "

キャラクタ配列の作成

キャラクタデータは、シングルコートで囲んで設定してください。例えば、つぎのラインは、name と名付けられた 1 行 13 列のキャラクタ配列を作成します。

```
name = 'Thomas R. Lee';
```

whos により、ワークスペースの中の情報を表示します。

```
Name        Size          Bytes  Class

name        1x13             26  char array
```

1 キャラクタは、ストレージの中で 2 バイトを使うことがわかります。

関数 class と ischar は、1 キャラクタ配列で name の識別を示します。

```
class(name)

ans =

char

ischar(name)

ans =

    1
```

2 つ以上のキャラクタ配列をつなげて新規のキャラクタ配列を作成することができます。文字列の連結関数 strcat あるいは、MATLAB の連結演算子 [] を使って行います。

```
name = 'Thomas R. Lee';
title = ' Sr. Developer';
strcat(name,',',title)

ans =

Thomas R. Lee, Sr. Developer
```

strvcat を使って文字列を垂直方向に連結することもできます。

2 次元キャラクタ配列の作成

2 次元キャラクタ配列を作成するとき、各々の行は同じ長さになることを確認してください。例えば、つぎの 2 つの入力行は、正確に 13 キャラクタなので、このラインは正しいものです。

```
name = ['Thomas R. Lee' ; 'Sr. Developer']

name =

Thomas R. Lee
Sr. Developer
```

異なる長さの文字列でキャラクタ配列を作成するときは、短い文字列のほうにブランクを付加して同じ長さにしてください。

```
name = ['Thomas R. Lee   '; 'Senior Developer'];
```

文字列配列を作る簡単な方法は、関数 char を使うことです。char は、短い文字列に長い文字列と同じになるように自動的にブランクを加えます。この例題で、char は 13 キャラクタの 'Thomas R. Lee' に 3 つのブランクを付けて、2 番目のものと同じにしています。

```
name = char('Thomas R. Lee','Senior Developer')

name =

Thomas R. Lee
Senior Developer
```

ある配列から文字列を抜き取るとき、関数 deblank を使って後ろに続くブランクを取り除きます。

```
trimname = deblank(name(1,:))

trimname =

Thomas R. Lee

size(trimname)

ans =

    1    13
```

キャラクタを数値に変換

キャラクタ配列を 16 ビットの数値として各々のキャラクタを格納します。関数 double を使って、文字を数値に変換します。そして、char は、逆の変換を行ないます。

```
name = double(name)

name =

    84   104   111   109   97   115   32   82   46   32   76   101   101

name = char(name)
name =

Thomas R. Lee
```

str2numを使ってキャラクタ配列を文字列によって表わされる数値に変換します。

```
str = '37.294e-1';
val = str2num(str)

val =

    3.7294
```

文字列のセル配列

1 グループの文字列を標準のキャラクタ配列に格納する代わりにセル配列に格納することが便利なことがしばしばあります。セル配列は同じ長さの行をもつキャラクタ配列にするためブランクを文字列に付け加える必要がありません。文字列のセル配列と共に機能する関数があります。

- 文字列の標準キャラクタ配列とセル配列とをお互いに変換することができます。
- 文字列比較演算を文字列セル配列に適用することができます。

セル配列の詳細は、" 構造体とセル配列 " の章を参照してください。

文字列のキャラクタ配列とセル配列との間の変換

関数 cellstr は、文字列のキャラクタ配列を文字列のセル配列に変換します。キャラクタ配列を考えましょう。

```
data = ['Allison Jones';'Development  ';'Phoenix      '];
```

行列の各行は、すべてが同じ長さ (この場合は、13 キャラクタ) になるようにブランクを加えます。

cellstr を使って、data 配列から文字列の中の一つを各々のセルに含ませたセルの列ベクトルを作成します。

```
celldata = cellstr(data)

celldata =

    'Allison Jones'
    'Development'
    'Phoenix'
```

関数 cellstr は、入力文字行列の行に加えられたブランクを取り除いていることに注意してください。

```
length(celldata{3})

ans =

     7
```

関数 iscellstr は、入力引数が文字列のセル配列であるかどうかを決定します。celldata の場合は、論理真 (1) を出力します。

```
iscellstr(celldata)

ans =

     1
```

char を使って、ブランクを加えた標準のキャラクタ配列に戻すことができます。

```
strings = char(celldata)

strings =

Allison Jones
Development
Phoenix
```

文字列 / 数値の変換

関数 str2double は、文字列のセル配列を文字列によって表わされる倍精度値に変換します。

```
c = {'37.294e-1'; '-58.375'; '13.796'};
str2double(c)

ans =

    3.7294
  -58.3750
   13.7960
```

文字列の比較

文字列と文字列に準ずるものを比較する 2 つの方法があります。

- 2 つの文字列のある部分、または、全体が等価かどうか比較します。
- 2 つの文字列の中の個々のキャラクタが等価かどうか比較します。
- ある文字列の中のすべての要素を分類し、個々の要素がキャラクタかスペースかを決めます。

これらの関数は、キャラクタ配列、セル配列共に機能します。

文字列の等価性の比較

2 つの入力文字列が等価かどうかを決定するには、2 つの関数があります。

- strcmp は、2 つの文字列が等価かどうかを決めます。
- strncmp は、2 つの文字列の最初の n キャラクタが等価かどうかを決めます。
- strcmpi と strncmpi は大文字と小文字を無視すること以外は、strcmp と strncmp と同じです。

2 つの文字列を考えます。

```
str1 = 'hello';
str2 = 'help';
```

文字列 str1 と str2 は等価ではありません。それで、strcmp を使うと 0 を出力します。例えば、

```
C = strcmp(str1,str2)

C =

     0
```

注意 C プログラマにとって、これは、MATLAB の strcmp と C の strcmp() との間の重要な違いです。C の場合、2 つの文字列が等しいと 0 を出力します。

---

str1 と str2 の最初の 3 文字は等価で、strncmp を使って、3 文字までを比較すると 1 を出力します。.

```
C = strncmp(str1,str2,2)

C =
```

これらの関数は、文字列のあるセル配列上にセル単位で機能します。文字列の 2 つのセル配列を考えましょう。

```
A = {'pizza';'chips';'candy'};
B = {'pizza';'chocolate';'pretzels'};
```

つぎに、文字列比較関数を適用しましょう。

```
strcmp(A,B)


ans =

    1
    0
    0

strncmp(A,B,1)

ans =

    1
    1
    0
```

演算子を使ってキャラクタのチェック

ユーザが比較しようとする配列が同じ大きさであるか、または、一方がスカラの場合、MATLAB の比較演算子をキャラクタ配列に適用することができます。例えば、2 つの文字列の中の各キャラクタが一致していることを決めるため等価演算子 (==) を使います。

```
A = 'fate';
B = 'cake';
A == B

ans =

     0   1   0   1
```

他のすべての比較演算子 (>，>=，<，<=，==，!=) は、対応するキャラクタの数値を比較します。

文字列内のキャラクタ分類

文字列内のキャラクタを分類する 2 つの関数を用意しています。

- sletter は、キャラクタが文字列かどうかを決めます。
- isspace は、キャラクタがスペース (ブランク、タブ、新しいライン) かどうかを決めます。

例えば、mystring と名付けた文字列を作成しましょう。

```
mystring = 'Room 401';
```

isletter は、文字列の中の各々のキャラクタを試し、mystring と同じ長さの出力ベクトルを作成します。

```
A = isletter(mystring)

A =

     1   1   1   1   0   0   0   0
```

mystring の最初 4 つのキャラクタが文字なので、A の最初の 4 要素は 1(真) です。

検索と置換

MATLABは文字列の中でキャラクタの検索や置換をするいくつかの関数を用意しています。label と名付けた文字列を考えましょう。

```
label = 'Sample 1, 10/28/95';
```

関数 strrep は、標準の検索、置換演算を行ないます。例えば、日付を '10/28' から '10/30' に変更するのに strrep を使います。

```
newlabel = strrep(label,'28','30')

newlabel =

Sample 1, 10/30/95
```

findstr は、長い文字列の中の文字列に準ずるものの始まる位置を出力します。label の中に 'amp' 文字列の中のものが生じる位置を求めるには、

```
position = findstr('amp',label)

position =

    2
```

'amp' が label の中で表れる位置は、2 番目のキャラクタです。

関数 strtok は、入力文字列の中で、最初に分離キャラクタが表れる位置の前のキャラクタを出力します。デフォルトの分離キャラクタは、ホワイトスペースキャラクタの集合です。ユーザは、strtok を使って、文章を言葉に分割できます。例えば、

```
function all_words = words(input_string)
remainder = input_string;
all_words = '';

while (any(remainder))
  [chopped,remainder] = strtok(remainder);
  all_words = strvcat(all_words,chopped);
end
```

関数 strmatch は、キャラクタ配列または文字列のセル配列の行を調べ、与えられた一連のキャラクタで始まる文字列を見つけます。それらのキャラクタで始まる行のインデックスを出力します。

```
maxstrings = strvcat('max','minimax','maximum')

maxstrings =

max
minimax
maximum

strmatch('max',maxstrings)

ans =

     1
     3
```

文字列 / 数字の変換

MATLAB の文字列 / 数字変換機能は、数値をキャラクタ文字列に変換します。数値を個々の数字で表した文字列表現、または、値を 16 進、または、バイナリ文字列に変換して保存します。スカラ x を考えましょう。

```
x = 5317;
```

デフォルトとして、MATLAB は値 5317 を含んだ 1 行 1 列の倍精度配列として数字 x を保存しています。関数 int2str は、このスカラを文字列 '5317' を含む 1 行 4 列のベクトルに分割します。

```
y = int2str(x);
size(y)

ans =

     1     4
```

関連した関数 num2str は、出力文字列の書式について、より細かく制御することができます。オプションの 2 番目の入力引数は、出力文字列の中の桁数を設定するか、または、実際の書式を設定します。

```
p = num2str(pi,9)

p =

3.14159265
```

int2str と num2str は共に、ラベル付きのプロットに対して便利です。例えば、つぎのラインは、num2str を使って、プロットの x 軸に自動的にラベルを用意するものです。

```
function plotlabel(x,y)
plot(x,y)
str1 = num2str(min(x));
str2 = num2str(max(x));
out = ['Value of f from ' str1 ' to ' str2];
xlabel(out);
```

数値 / 文字列変換関数の他のクラスは、数値を他の基底、例えば、バイナリまたは 16 進表現の数値に表現するものです。例えば、関数 dec2hex は、10 進値を対応する 16 進表現に変換します。

```
dec_num = 4035;
hex_num = dec2hex(dec_num)

hex_num =

FC3
```

文字列変換関数の一覧は、strfun デレクトリを参照してください。

配列 / 文字列の変換

MATLAB の関数 mat2str は、配列を MATLAB で計算できる文字列に変換します。この文字列は、例えば、eval のような関数で使用可能な入力になり、MATLAB コマンドラインでタイプしたとすると、すぐに入力文字列を計算します。

3 行 2 列の配列 A を作成しましょう。

```
A = [1 2 3; 4 5 6]

A =

     1     2     3
     4     5     6
```

mat2str は、コマンドラインで A を作成するために入力したテキストを含む文字列を出力します。

```
B = mat2str(A)

B =

[1 2 3; 4 5 6]
```

19

多次元配列

多次元配列 19-3
多次元配列の作成 19-4
多次元配列に関する情報の取得 19-8
インデックス付け 19-8
サイズの変更 19-10
配列の次元の並べ替え 19-11

多次元配列の計算 19-14
ベクトルを操作する関数 19-14
要素単位に操作する関数 19-14
面と行列に操作する関数 19-14

データの多次元配列化 19-16

多次元セル配列 19-18

多次元構造体配列 19-19
多次元構造体配列への関数の適用 19-19

この章では、2 次以上の次元をもつ MATLAB 配列、*多次元配列* について説明します。多次元配列は、数値、文字列、セル、構造体のどの配列でも構いません。これらの配列は、たとえば、多変量データの表現や 2 次元データの複数ページの表現など、広範囲に利用できます。

この章では、以下のトピックスについて説明します。

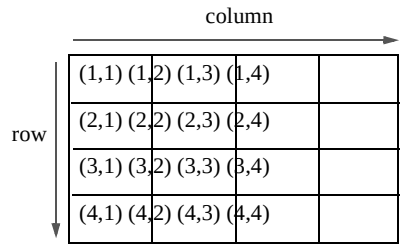
- " 多次元配列 "
- " 多次元配列の計算 "
- " データの多次元配列化 "
- " 多次元セル配列 "
- " 多次元構造体配列 "

MATLAB は、多次元配列を直接サポートする関数を提供します。データアーキテクチャを使って動作する M- ファイルを作成することで、この機能を拡張することができます。

関数	詳細
cat	配列の連結
ipermute	配列の次元の並べ替えの逆操作
ndgrid	多次元関数の作成と補間
ndims	配列の次元数
permute	配列の次元の並べ替え
shiftdim	次元のシフト
squeeze	1 の次元の削除

多次元配列

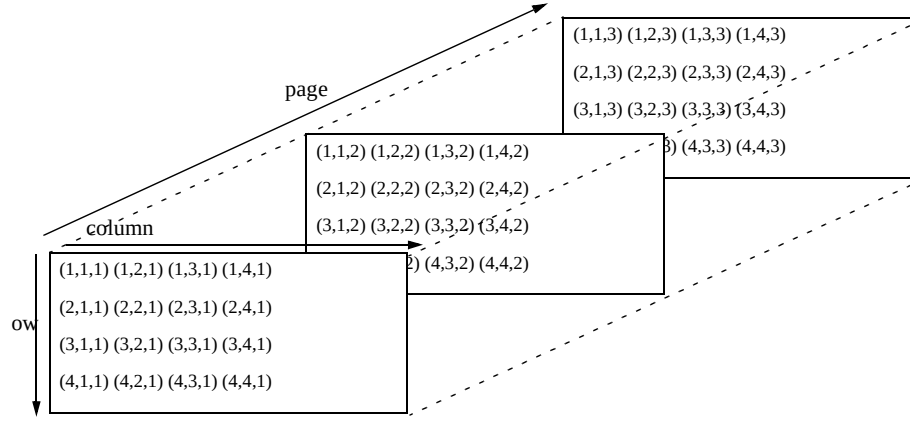
MATLAB の多次元配列は、通常の 2 次元行列の拡張です。行列は、2 つの次元、すなわち、行の次元と列の次元をもっています。



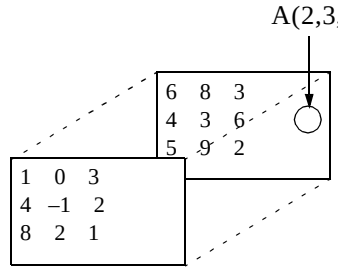
2 つのサブスクリプトを使って、2 次元行列の要素にアクセスすることができます。1 つは行のインデックスで、もう 1 つは列のインデックスです。

多次元配列も、付加的な サブスクリプトで、インデックスを指定します。たとえば、3 次元配列は 3 つのサブスクリプトを使います。

- 次元 1 の 1 番目の参照、行
- 次元 2 の 2 番目の参照、列
- 次元3の3番目の参照。このガイドは3次元以上を表現するために *ページ*の概念を使います。



たとえば、ページ 2 の 2 行 3 列の要素にアクセスするためには、サブスクリプト (2,3,2) を使います。




```
A(:,:,1) =

     1     0     3
     4    –1     2
     8     2     1

A(:,:,2) =

     6     8     3
     4     3     6
     5     9     2
```

配列の次元を増すには、サブスクリプトを加えます。たとえば、4 次元配列は 4 つのサブスクリプトをもちます。最初の 2 つは行と列の組合わせで、後の 2 つはデータの 3 次元目と 4 次元目にアクセスします。

注意　一般の多次元配列関数は datatypes ディレクトリにあります。

多次元配列の作成

多次元配列の作成に、2 次元行列に使った方法を使うことができます。さらに、MATLAB は多次元配列の作成に有効な特殊な結合関数を提供します。

この節では、つぎのことを説明します。

- インデックスを使って、配列の作成。
- MATLAB 関数を使って、配列の作成。
- 関数 cat を使って、多次元配列の作成。

インデックスを使って配列の作成

多次元配列を作成する 1 つの方法は、2 次元配列を作成し、それを拡張することです。たとえば、つぎのような簡単な 2 次元配列 A から始めます。

```
A = [5 7 8; 0 1 9; 4 3 6];
```

A は 3 行 3 列の配列で、行の次元は 3 で、列の次元は 3 です。A に 3 次元目を加えるために、つぎのようにします。

```
A(:,:,2) = [1 0 4; 3 5 6; 9 8 7]
```

MATLAB は、つぎのように出力します。

```
A(:,:,1) =

     5     7     8
     0     1     9
     4     3     6

A(:,:,2) =

     1     0     4
     3     5     6
     9     8     7
```

同様の割り当てステートメントを使って、配列に行、列、ページを追加することができます。

多次元配列の拡張． A を任意の次元に拡張するためには、つぎのようにします。

- 適切なサブスクリプトを加えて、希望する値を割り当てます。
- 対応する配列次元と同じ要素数を割り当てます。数値配列に対しては、すべての行は同じ要素数でなければならず、すべてのページは同じ行数と列数でなければなりません。

次元全体に同じ値を与えるために、MATLAB のスカラ展開機能であるコロン演算子を利用することができます。

```
A(:,:,3) = 5;
A(:,:,3)

ans =

     5     5     5
     5     5     5
     5     5     5
```

A を、3-3-3-2 の 4 次元配列に展開するためには、つぎのように入力してください。

```
A(:,:,1,2) = [1 2 3; 4 5 6; 7 8 9];
A(:,:,2,2) = [9 8 7; 6 5 4; 3 2 1];
A(:,:,3,2) = [1 0 1; 1 1 0; 0 1 1];
```

最初の 2 つの代入後、MATLAB は対応する次元のサイズを維持するために、A にゼロを設定することに注意してください。

関数を使って配列の作成

2 次元配列の作成方法と同じように、多次元配列を作成するために、randn, ones,zeros のような MATLAB 関数を使うことができます。ユーザが与える引数は、結果の配列に対応する次元のサイズを表わします。たとえば、一様分布の乱数の 4-3-2 の配列を作成するためには、つぎのようにします。

```
B = randn(4,3,2)
```

1 つの定数値をもつ配列を作成するためには、関数 repmat を使います。repmat は、配列の次元のベクトルとして (この場合、1 行 1 列の配列として) 配列を作ります。

```
B = repmat(5,[3 4 2])

B(:,:,1) =

     5     5     5     5
     5     5     5     5
     5     5     5     5
B(:,:,2) =

     5     5     5     5
     5     5     5     5
     5     5     5     5
```

注意 配列の任意の次元が、サイズがゼロで空配列の形式をもつことがあります。たとえば、10-0-20 は、多次元配列としては、正しいサイズです。

関数 cat を使って多次元配列の作成

関数 cat は、多次元配列の簡単な作成法です。指定した次元について配列のリストを連結していきます。

```
B = cat(dim,A1,A2...)
```

A1, A2 等は結合する配列で、dim は配列を結合する次元です。たとえば、cat を使って新しい配列を作るために、つぎのようにします。

```
B = cat(3,[2 8; 0 5],[1 3; 7 9])

B(:,:,1) =

     2     8
     0     5


B(:,:,2) =

     1     3
     7     9
```

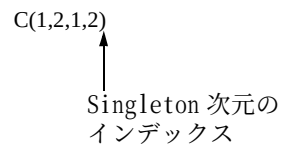
関数 cat は、既存のデータと新しいデータを任意に組合わせることが可能です。さらに、cat を入れ子の形式にすることができます。たとえば、下記の行は 4 次元配列を作成します。

```
A = cat(3,[9 2; 6 5],[7 1; 8 4])
B = cat(3,[3 5; 0 1],[5 6; 2 1])
D = cat(4,A,B,cat(3,[1 2; 3 4],[4 3;2 1]))
```

cat は、必要ならば次元の間に "1" のサブスクリプトを自動的に加えます。たとえば、2-2-1-2 の配列を作成するためには、つぎのように入力します。

```
C = cat(4,[1 2; 4 5],[7 8; 3 2])
```

上記の場合では、cat は最後の次元が 1 でない 4 次元配列を作成するために必要な次元を挿入します。dim 引数が 5 ならば、上記のステートメントは 2-2-1-1-2 の配列を作成します。結果、配列を表すインデックスに 1 を加えます。4 次元の場合の値 8 にアクセスするためには、つぎの表現を使ってください。



多次元配列に関する情報の取得

作成した多次元配列に関する情報を得るために、MATLAB 関数とコマンドを使うことができます。

情報	関数	例題
配列のサイズ	size	size(C) ans = 2 2 1 2 rows columns dim3 dim4
配列の次元	ndims	ndims(C) ans = 4
配列のストレージと形式	whos	whos Name Size Bytes Class A 2x2x2 64 double array B 2x2x2 64 double array C 4-D 64 double array D 4-D 192 double array Grand total is 48 elements using 384 bytes

インデックス付け

2 次元行列に適用する多くの概念は多次元配列にも拡張されます。

多次元配列の 1 つの要素にアクセスするには、整数のサブスクリプトを使います。各サブスクリプトは 1 番目のインデックスを行の次元に、2 番目のインデックスを列の次元に、3 番目のインデックスを最初のページの次元に等々、インデックス付けを行います。

ランダムな整数からなる 10-by-5-by-3 の配列 nddata を考えます。

```
nddata = fix(8*randn(10,5,3));
```

たとえば、nddata のページ 2 の要素 (3,2) にアクセスするには、nddata(3,2,2) を使います。

配列のサブスクリプトとしてベクトルを使うことができます。この場合、各ベクトルの要素は配列の次元により定義される領域内で有効なサブスクリプトでなければなりません。nddata のページ 3 の要素 (2,1), (2,3), (2,4) にアクセスするためには、つぎのようにしてください。

```
nddata(2,[1 3 4],3);
```

コロンと多次元配列のインデックス

MATLAB のコロンを使ったインデックス付けは、多次元配列にも拡張されます。たとえば、nddata のページ 2 の 3 番目の列にアクセスするためには、nddata(:,3,2) を使ってください。

コロン演算子は、データのサブセットにアクセスするのにも有効です。たとえば、nddata(2:3,2:3,1) は、ページ 1 のデータのサブセットである 2 行 2 列の配列になります。この行列は、配列の最初のページの 2 行目、3 行目と 2 列目、3 列目のデータから成り立ちます。

コロン演算子は、代入ステートメントの両辺で配列のサブスクリプトとして表わすことができます。たとえば、ゼロからなる 4 行 4 列の配列を作成するためには、つぎのようにします。

```
C = zeros(4,4)
```

配列 nddata の 2 行 2 列のサブセットを C の中央の 4 要素に代入します。

```
C(2:3,2:3) = nddata(2:3,1:2,2)
```

多次元配列のインデックス付けのあいまいさを防ぐ

つぎのようなステートメント

```
A(:,:,2) = 1:10
```

は、データを受け取る側の次元の形についての情報を与えないため、あいまいな表現です。上記の例で、ステートメントは 1 次元ベクトルを 2 次元のものに代入しようとします。MATLAB は、そのような場合、エラーを出力します。あいまいさを解決するためには、代入するデータに対する対象物 (左側の型) すなわち、データの型と対象物に関する十分な情報が必要です。たとえば、つぎのようにです。

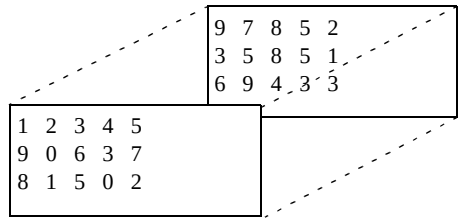
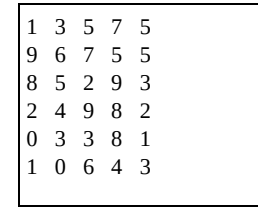
```
A(1,:,2) = 1:10;
```

サイズの変更

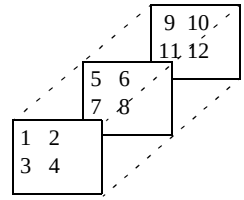
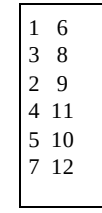
型、または、サイズを変更しない限り、MATLAB 配列は、作成時に指定した次元を保持しています。要素を加えたり、削除することで配列のサイズを変更します。同じ要素を保持しながら、配列の行、列、ページの次元の再指定をして配列の形を変更することもできます。関数 reshape は、後者の操作を行います。多次元配列について、その形式は、つぎのようになります。

```
B = reshape(A,[s1 s2 s3 ...])
```

s1, s2 等は、変更される行列の各次元の希望するサイズを表わします。変更後の配列は、オリジナルの配列と同じ要素数をもたなければなりません (次元のサイズの積は一定になります)。

M	**reshape(M, [6 5])**
	

関数 reshape は、列単位で操作します。オリジナルのデータ構造の各列から連続する要素を操作することで行列を変更します。

C	**reshape(C, [6 2])**
	

つぎに示すのは、nddata を新しい配列に変更します。

```
B = reshape(nddata,[6 25])
C = reshape(nddata,[5 3 10])
D = reshape(nddata,[5 3 2 5])
```

1 の次元 (シングルトン) の削除

MATLAB は、配列の作成や変更時に明示的に指定する場合や 1 次元の配列になる計算を行うときに、1 の次元を作成します。

```
B = repmat(5,[2 3 1 4]);
size(B)

ans =

     2     3     1     4
```

関数 squeeze は、配列から 1 の次元を取り除きます。

```
C = squeeze(B);
size(C)

ans =

     2     3     4
```

関数squeezeは、2次元配列に影響を与えません。行ベクトルは、行のまま残ります。

配列の次元の並べ替え

関数 permute は、配列の次元の並べ替えを行います。

```
B = permute(A,dims);
```

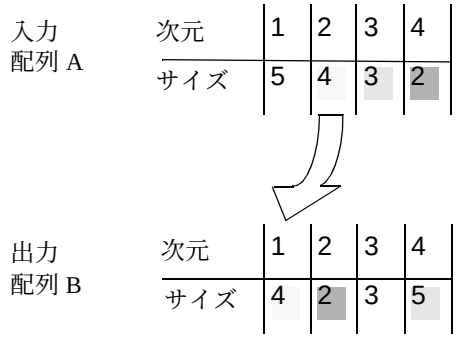
dims は、A の次元に対して新しい順番を指定するベクトルです。ここで、1 は 1 番目の次元 (行)、2 は 2 番目の次元 (列)、3 はページに対応します。

A	B= permute(A,[2 1 3])	C = permute(A,[3 2 1])
A(:,:,1) = 1 2 3 4 5 6 7 8 9 A(:,:,2) = 0 5 4 2 7 6 9 3 1	B(:,:,1) = 1 4 7 2 5 8 3 6 9 B(:,:,2) = 0 2 9 5 7 3 4 6 1 行と列のサブスクリプトは保存されます（ページ単位の転置）	C(:,:,1) = 1 2 3 0 5 4 C(:,:,2) = 4 5 6 2 7 6 C(:,:,3) = 7 8 9 9 3 1 Row and page sub-scripts are

関数 permute のより詳細な観察として、サイズ 5-4-3-2 の配列 A を考えます。列の次元を 1 番目に、つぎに 2 ページ目の次元、1 ページ目の次元、最後に行の次元になるように次元を再構成します。結果は、4-2-3-5 の配列になります。

B = permute(A,[2 4 3 1])

A の次元 2 を B の最初のサブスクリプト位置に移動し、次元 4 を 2 番目のサブスクリプト位置に移動、等を行います。



permute の引数リストの次元の順番は、出力配列のサイズと型を決定します。この例では、2 番目の次元は最初の位置に移動します。元の配列の 2 番目の次元のサイズが 4 であるため、出力配列の最初の次元もサイズが 4 になります。

行列の行と列の次元を入れ替える関数 transpose の拡張として、permute の操作を考えることができます。permute に対して、入力次元のリストの順序はサブスクリプトの並べ替えを決定します。上記の例では、A の要素 (4,2,1,2) は、B の要素 (2,2,1,4) になり、A の要素 (5,4,3,2) は、B の要素 (4,2,3,5) になります。

逆置換

関数 ipermute は、permute の逆操作です。入力配列 A と次元のベクトル v を与えると、ipermute は permute(B,v) が A を出力するような配列 B を出力します。

たとえば、つぎのステートメントは、入力配列 C と等価な配列 E を作成します。

```
D = ipermute(C,[1 4 2 3]);
E = permute(D,[1 4 2 3])
```

同じ次元ベクトルを使って、ipermute をコールして変換した後で、オリジナルの配列を得ることができます。

多次元配列の計算

多くの MATLAB の計算関数や数学関数は、多次元配列を引数としています。これらの関数は、多次元配列の特定の次元を操作します。すなわち、個々の要素、ベクトル、行列を操作します。

ベクトルを操作する関数

sum や mean 等のようにベクトルを操作する関数は、デフォルトでは、多次元配列の最初に 1 でない次元に対して操作します。これらの関数のほとんどは、操作する次元をオプションで指定することができます。しかし、これには例外があります。たとえば、2 つのベクトルの外積を求める関数 cross は、長さ 3 をもつ最初に 1 でない次元の操作を行います

注意　ほとんどの場合、これらの関数は入力引数に対する制限があります。たとえば、複数の配列を引数とする関数は、配列が同じサイズである必要があります。関数の引数の詳細はオンラインヘルプを参照してください。

要素単位に操作する関数

elfun ディレクトリにある三角関数や指数関数のように、2 次元配列の要素単位に操作を行う MATLAB 関数は、多次元の場合に対しても全く同様に操作します。たとえば、関数 sin は、関数の入力引数と同じサイズの配列を出力します。出力配列の各要素は、入力配列の対応する要素の正弦値です。

同様に、代数演算、論理演算、比較演算は各次元で同じサイズである多次元配列の対応する要素を操作します。オペランドがスカラと配列であれば、演算子は配列の各要素にスカラを適用します。

面と行列に操作する関数

matfun ディレクトリにある線形代数や行列関数のように、面や行列に対する演算を行う関数は、引数として多次元配列を受け付けません。つまり、matfun ディレクトリにある関数や配列演算子 *, ^, \, / に対して多次元引数を使うことはできません。多次元引数やオペランドを与えるとエラーになります。

多次元配列内で行列関数や行列に対する演算子を適用するためにインデックスを使うことができます。たとえば、3 次元配列 A を作成します。

```
A = cat(3,[1 2 3;9 8 7;4 6 5],[0 3 2;8 8 4;5 3 5],[6 4 7;6 8 5;...
5 4 3]);
```

多次元配列全体に関数 eig を適用するとエラーになります。

```
eig(A)
??? Error using ==> eig
Input arguments must be 2-D.
```

しかし、配列内の面に eig を適用することはできます。たとえば、配列の 1 ページだけ (この場合、2 ページ目) をインデックス付けするためにコロン表記を使います。

```
eig(A(:,:,2))

ans =

   -2.6260
   12.9129
    2.7131
```

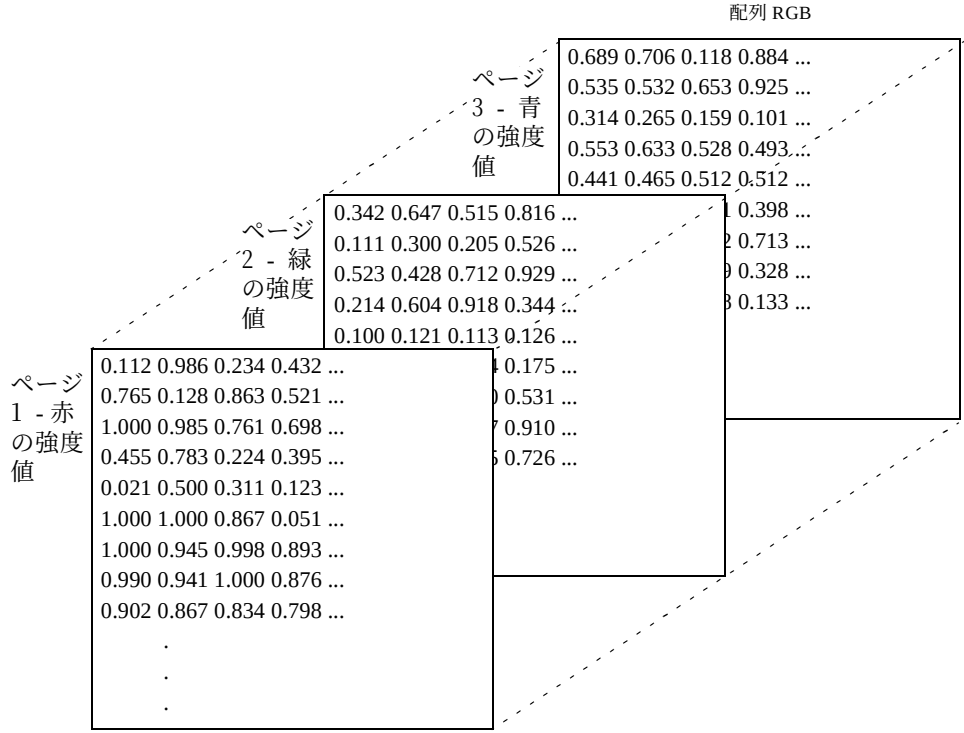
注意　最初のサブスクリプトがコロンでなければ、エラーを避けるために squeeze を使わなければなりません。たとえば、eig(A(2,:,:)) は、入力のサイズが [1 3 3] なのでエラーになります。しかし、式 eig(squeeze(A(2,:,:))) は、eig に有効な 2 次元行列を渡します。

データの多次元配列化

多次元配列を使って、データを表現する 2 通りの方法があります。

- 2 次元データの面またはページとして表現します。すると、これらのページを行列として扱うことができます。
- 多変数データ、または、多次元データを扱います。たとえば、各要素が室内の等間隔点で観測された温度、または、気圧に対応する 4 次元配列のようなものです。

たとえば、RGB イメージを考えます。単一のイメージに対して、多次元配列は、データを保存しアクセスするためのおそらく最も簡単な方法です。



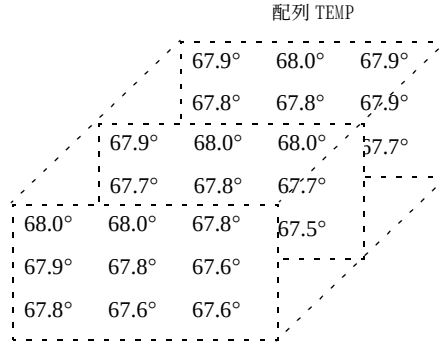
イメージの面全体にアクセスするためには、つぎの式を使います。

```
red_plane = RGB(:,:,1);
```

サブイメージにアクセスするためには、つぎの式を使います。

```
subimage = RGB(20:40,50:85,:);
```

RGB イメージは、表示やフィルタリングのような操作に対して面内でアクセスされる必要があるデータの良い例です。しかし、他の場合ではデータ自身が多次元であるかもしれません。たとえば、室内の等間隔点で観測された温度について考えます。ここで、各値の位置はデータ集合の整数部分です。各要素の 3 次元での物理的な位置は方向を表す情報です。このようなデータは、多次元配列として表現できます。



すべての観測値の平均を求めるために、つぎの式を使います。

```
mean(mean(mean(TEMP)));
```

各ページの room 要素 (2,2) の中間値のベクトルを得るために、つぎの式を使います。

```
B = TEMP(2,2,:);
```

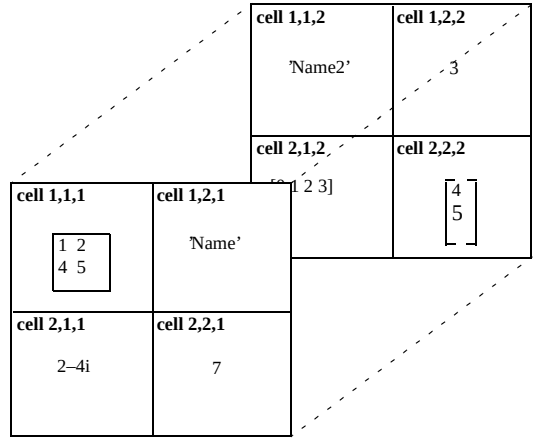
多次元セル配列

数値配列のように、MATLAB の多次元セル配列の操作は、2 次元セル配列操作の拡張です。数値配列に対して使うのと同様に多次元セル配列を作成するために関数 cat を使うことができます。

たとえば、簡単な 3 次元セル配列 C を作成します。

```
A{1,1} = [1 2;4 5];
A{1,2} = 'Name';
A{2,1} = 2-4i;
A{2,2} = 7;
B{1,1} = 'Name2';
B{1,2} = 3;
B{2,1} = 0:1:3;
B{2,2} = [4 5]';
C = cat(3,A,B);
```

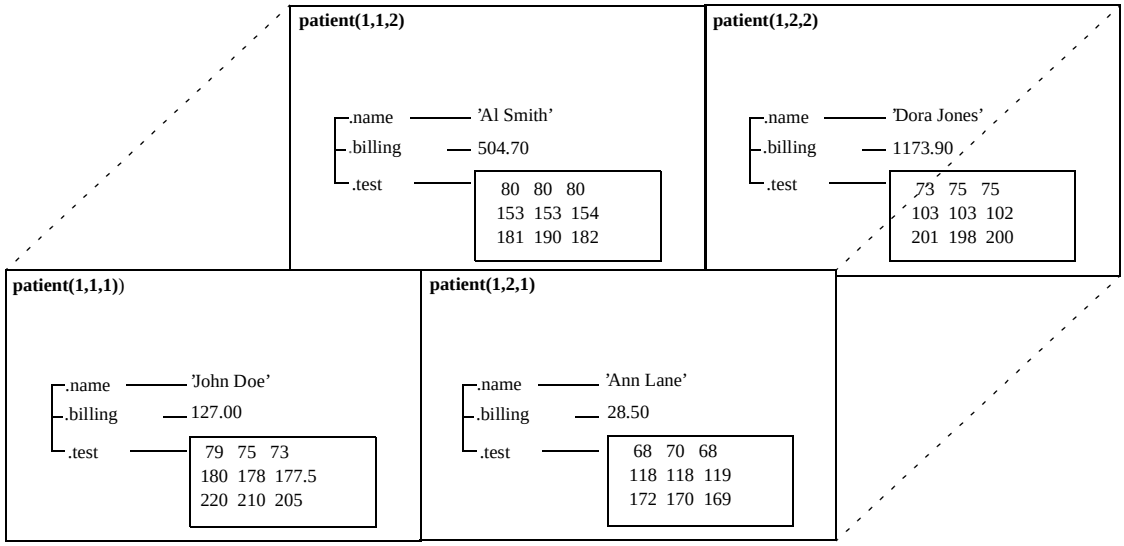
C のセルに対するサブスクリプトは、つぎのようになります。



多次元構造体配列

多次元構造体配列は、2 次元構造体配列の拡張です。他のタイプの多次元配列のように、直接代入、または、関数 cat を使って作成できます。

```
patient(1,1,1).name = 'John Doe';patient(1,1,1).billing = 127.00;
patient(1,1,1).test = [79 75 73; 180 178 177.5; 220 210 205];
patient(1,2,1).name = 'Ann Lane';patient(1,2,1).billing = 28.50;
patient(1,2,1).test = [68 70 68; 118 118 119; 172 170 169];
patient(1,1,2).name = 'Al Smith';patient(1,1,2).billing = 504.70;
patient(1,1,2).test = [80 80 80; 153 153 154; 181 190 182];
patient(1,2,2).name = 'Dora Jones';patient(1,2,2).billing = 1173.90;
patient(1,2,2).test = [73 75 75; 103 103 102; 201 198 200];
```



多次元構造体配列への関数の適用

多次元構造体配列に関数を適用するために、インデックスを使ってメンバとメンバの要素に対して操作を行います。たとえば、patient(1,1,2) の test 配列の列の和を求めます。

```
sum((patient(1,1,2).test));
```

同様に、patient 配列に、すべての billing メンバを追加します。

```
total = sum([patient.billing]);
```

20

構造体とセル配列

構造体 20-3
構造体配列の作成 20-4
構造体配列内のデータのアクセス 20-6
構造体配列を使った関数 size の使用 20-9
構造体にフィールドを追加する 20-9
構造体からフィールドを削除する 20-9
関数と演算子の適用 20-9
構造体の操作のために関数を作成 20-10
構造体配列内のデータの構造化 20-11
入れ子型の構造体 20-16

セル配列 20-18
セル配列の作成 20-19
セル配列からデータを得る 20-22
セルの削除 20-23
セル配列の変更 20-24
セル配列を使って、変数のリストを置き換える . . . 20-24
関数と演算子の適用 20-26
セル配列内のデータの構造化 20-26
セル配列を入れ子にする 20-28
セル配列と数値配列の変換 20-29
構造体のセル配列 20-30

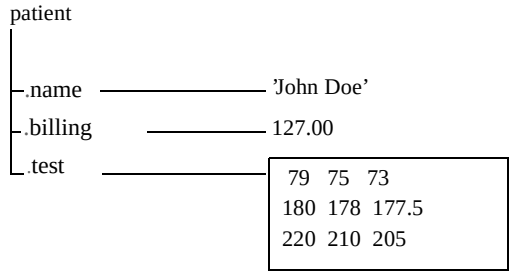
構造体は、フィールドと名付けられたものから構成される種々のデータを集めたものです。セル配列は、MATLAB 配列の特別なクラスで、その要素は、MATLAB 配列を含むセルから構成されています。構造体とセル配列は、異なる種類のデータに対して、階層的な格納機能をもっています。2 つのものは、データを組織化する方法が基本的には異なります。構造体の中では、フィールド名を使って、データにアクセスできますが、セル配列では、行列のインデックス操作を使って行います。

つぎの表は、構造体やセル配列と共に機能する MATLAB 関数を記述しています。

カテゴリ	関数	詳細
構造体関数	fieldnames	構造体のフィールド名の取得
	getfield	構造体のフィールドの内容の取得
	isfield	フィールドが構造体配列かをチェック
	isstruct	構造体かをチェック
	rmfield	構造体のフィールドを削除
	setfield	構造体フィールドの内容を設定
	struct	構造体配列の作成または変換
	struct2cell	構造体配列をセル配列に変換
セル配列関数	cell	セル配列を作成
	cell2struct	セル配列を構造体配列に変換
	celldisp	セル配列の内容の表示
	cellfun	セル関数をセル配列に適用
	cellplot	セル配列の内容をグラフィカルに表示
	deal	入力を出力にコピー
	iscell	セル配列かをチェック
	num2cell	数値配列をセル配列に変換

構造体

構造体 は、*フィールド* と呼ばれる名前が付いた " データコンテナ " をもつ MATLAB 配列です。構造体のフィールドは任意の種類のデータを含むことができます。たとえば、名前を表わすテキスト文字列、費用を表わすスカラ、医学検査の結果の行列等があります。



通常の配列のように、構造体は本質的に配列指向です。値 5 が 1 行 1 列の数値配列であるように、1 つの構造体は、1 行 1 列の構造体配列です。多次元構造体配列を含む任意のサイズまたは型の構造体配列を作ることができます。

注意　この節の例題は、2 次の構造体配列を扱います。高次の構造体配列の例については、" 多次元配列 " を参照してください。

以下の一覧はこの節の内容をまとめたものです。

- " 構造体配列の作成 "
- " 構造体配列内のデータのアクセス "
- " 構造体配列を使った関数 size の使用 "
- " 構造体にフィールドを追加する "
- " 構造体からフィールドを削除する "
- " 関数と演算子の適用 "
- " 構造体の操作のために関数を作成 "
- " 構造体配列内のデータの構造化 "
- " 入れ子型の構造体 "

構造体配列の作成

2 つの方法で構造体を作ることができます。

- 代入ステートメントを使う
- 関数 struct を使う

代入ステートメントを使って構造体配列を作成

データを個々のフィールドに代入することにより、簡単な 1 行 1 列の構造体配列を作ることができます。MATLAB は、ユーザがこの作業を続けるに従い、自動的に構造体を作ります。たとえば、この節の初めに示した 1 行 1 列の patient 構造体配列を作ります。

```
patient.name = 'John Doe';
patient.billing = 127.00;
patient.test = [79 75 73; 180 178 177.5; 220 210 205];
```

つぎのように入力します。

```
patient
```

コマンドラインでの結果は、つぎのようになります。

```
name: 'John Doe'
billing: 127
test: [3x3 double]
```

patient は、3 つのフィールドをもつ構造体を含む配列です。構造体配列を拡張するためには、構造体名の後にサブスクリプトを追加します。

```
patient(2).name = 'Ann Lane';
patient(2).billing = 28.50;
patient(2).test = [68 70 68; 118 118 119; 172 170 169];
```

patient 構造体配列は、サイズ [1 2] です。構造体配列が 1 つ以上の要素を含むならば、MATLAB は配列名をタイプインしたときに個々のフィールドの内容を表示しないことに注意してください。その代わりに、構造体が含む情報の種類のまとめを表示します。

```
patient

patient =

1x2 struct array with fields:
    name
    billing
    test
```

この情報を得るために関数 fieldnames を使うこともできます。fieldnames は、フィールド名を含む文字列のセル配列を出力します。

構造体を拡張すると、MATLAB は、つぎのことを満足するように、指定していないフィールドを空行列で満たします。

- 配列内のすべての構造体は、同じフィールド数をもちます。
- すべてのフィールドは、同じフィールド名をもちます。

たとえば、patient(3).name = 'Alan Johnson' と入力すると、patient 配列をサイズ [1 3] に拡張します。このとき、patient(3).billing と patient(3).test は、空行列になります。

注意 フィールドのサイズは、配列内のすべての要素に対して一致する必要はありません。patient の例では、name フィールドは異なる長さをもつことができ、test フィールドは異なるサイズの配列になっています。

関数 struct を使って、構造体配列を作成

関数 struct を使って、構造体からなる配列を前もって設定できます。その基本的な型は、つぎのものです。

```
str_array = struct('field1',val1,'field2',val2, ...)
```

ここで、引数は、フィールド名とその値です。フィールド値は、MATLAB データコンストラク、または、値のセル配列のいずれかで表現された単一値で設定できます。引数リストの中のすべてのフィールド値は、同じスケール (単一値、または、セル配列) でなければなりません。

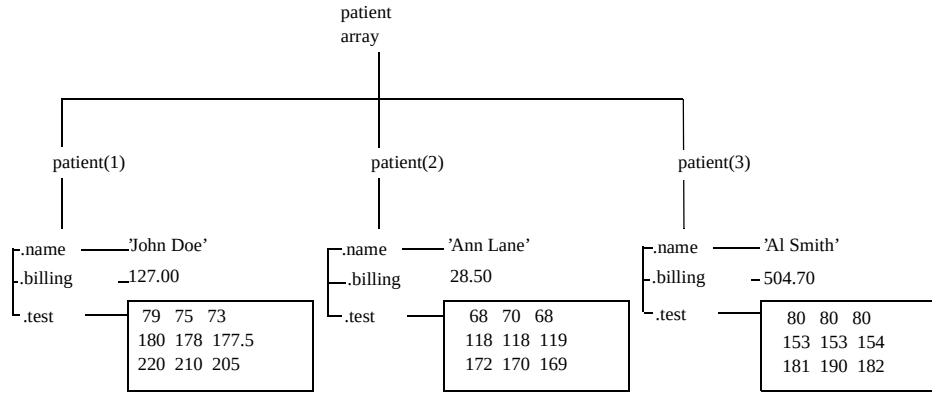
前もって構造体配列を設定する 2 つの方法を使用することができます。これらの方法は、構造体フィールドが初期化される方法が異なるものです。たとえば、構

造体フィールド temp と rainfall をもつ 1 行 3 列の構造体配列 weather を設定することを考えましょう。この配列を設定するには、つぎに示す 3 つの方法があります。

方法	シンタックス	初期化
struct	`weather(3) = struct('temp',72,'rainfall',0.0);`	weather(3) は、示されるフィールド値を使って初期化されます。配列内の他の構造体に対するフィールド weather(1) と weather(2) は、空行列で初期化されます。
struct with repmat	`weather = repmat(struct('temp',72, 'rainfall',0.0),1,3);`	weather 配列内のすべての構造体は、1 セットのフィールド値を使って初期化されます。
struct with cell array syntax	`weather = struct('temp',{68,80,72}, 'rainfall',{0.2,0.4,0.0});`	weather 配列内の構造体は、セル配列で設定したフィールド値を使って初期化されます。

構造体配列内のデータのアクセス

構造体配列のインデックスを使って、構造体配列内の任意のフィールドやフィールド要素を取得することができます。同様に、任意のフィールドやフィールド要素に値を設定することができます。この節の例として、つぎの構造体を考えます。



標準のサブスクリプトを構造体配列名に加えることで、サブ配列にアクセスできます。たとえば、つぎのステートメントは、1 行 2 列の構造体配列になります。

```
mypatients = patient(1:2)

1x2 struct array with fields:
    name
    billing
    test
```

mypatients 配列の中の最初の構造体は、patients 配列の中の最初の構造体と同じです。

```
mypatients(1)

ans =

       name: 'John Doe'
    billing: 127
       test: [3x3 double]
```

特別な構造体のフィールドにアクセスするため、構造体配列名の後にピリオド (.)、その後にフィールド名を設定します。

```
str = patient(2).name

str =

Ann Lane
```

フィールド内の要素にアクセスするためには、フィールド名に適切なインデックス付けのメカニズムを追加してください。すなわち、フィールドが配列を含めば、配列のサブスクリプトを使ってください。フィールドがセル配列を含めば、セル配列のサブスクリプトを使ってください。

```
test2b = patient(3).test(2,2)

test2b =

   153
```

たとえば、構造体のフィールドに値を代入するために同じ表記法を使ってください。

```
patient(3).test(2,2) = 7;
```

複数の構造体に対して、一度にフィールド値を抽出することができます。たとえば、つぎのラインは、billing フィールドのすべてを含む 1 行 3 列のベクトルを作成します。

```
bills = [patient.billing]

bills =

  127.0000   28.5000  504.7000
```

同様に、最初の 2 つの構造体に対して、test データを含むセル配列を作成することができます。

```
tests = {patient(1:2).test}

tests =

    [3x3 double]    [3x3 double]
```

setfield と getfield を使って、フィールド値にアクセスする

直接インデックス法は、通常フィールド値を代入したり、取得するための最も効率の良い方法です。しかし、文字列としてのみフィールド名がわかっている場合、たとえば、M- ファイル内でフィールド名を得るために関数 fieldnames を使った場合、同じことを行うために関数 setfield と getfield を使うことができます。

getfield は、フィールド、または，フィールド要素から値を得ます。

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

ここで、field_index はオプションで、array_index は、1 行 1 列の構造体配列に対するオプションです。関数シンタックスは、つぎの式に対応します。

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

たとえば、patient 配列の 2 番目の構造体の name フィールドにアクセスするためには、つぎの式を使います。:

```
str = getfield(patient,{2},'name');
```

同様に、setfield は、つぎの構文を使って、フィールドに値を代入します。

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

構造体配列を使った関数 size の使用

構造体配列、または、任意の構造体フィールドのサイズを得るために、関数 size を使ってください。引数として構造体配列名を与えると、size は配列の次元のベクトルを出力します。array(n).field の形式で引数を与えると、size 関数は、フィールドの内容のサイズを含むベクトルを出力します。

たとえば、1 行 3 列の構造体配列 patient に対して、size(patient) はベクトル [1 3] を出力します。ステートメント size(patient(1,2).name) は、patient の要素 (1,2) に対する name 文字列の長さを出力します。

構造体にフィールドを追加する

単一の構造体にフィールドを加えることにより、ある配列内のすべての構造体にフィールドを追加することができます。たとえば、patient 配列に社会保険番号フィールドを追加するためには、つぎのような代入を使います。

```
patient(2).ssn = '000–00–0000';
```

patient(2).ssn は、代入された値をもちます。配列内の他のすべての構造体は、ssn フィールドをもちますが、値を明示的に代入しなければ、これらのフィールドは空行列を含みます。

構造体からフィールドを削除する

関数 rmfield を使って構造体配列内の各構造体から引数で与えたフィールドを削除できます。もっとも基本的な型は

```
struc2 = rmfield(array,'field')
```

で、array は構造体配列で、'field' は、そこから削除するフィールド名です。たとえば、patient 配列から name フィールドを削除するためには、つぎのように入力します。

```
patient = rmfield(patient,'name');
```

関数と演算子の適用

他の MATLAB 配列に対して行うのと同様に、フィールドやフィールド要素に対して演算を行います。演算するデータにアクセスするためにはインデックスを使ってください。たとえば、patient(2) 内の test 配列の行の平均を求めるためには、つぎのようにします。

```
mean((patient(2).test)');
```

構造体配列内のフィールドに関数や演算子を適用するためには複数の方法があります。patient 配列にすべての billing フィールドの和を計算する方法は、つぎのようになります。

```
total = 0;
for j = 1:length(patient)
      total = total + patient(j).billing;
end
```

このような演算を簡単にするために、MATLAB は構造体配列内の同じ名前のフィールドすべてに対して演算を行うことができます。関数コール内で、array.field を中括弧で囲んでください。たとえば、つぎの式を使って、patient 配列内のすべての billing フィールドを合計することができます。

```
total = sum ([patient.billing]);
```

これは、カンマで分離する方法と等価です。

```
total = sum ([patient(1).billing, patient(2).billing...]);
```

この構文は、オペランドのフィールドがスカラのフィールドである場合に最も有効です。

構造体の操作のために関数を作成

特定のフィールド構造をもつ構造体に対して動作する関数を作成することができます。このような関数は、構造体のフィールドや要素にアクセスできます。

注意　構造体に動作する M- ファイル関数を作成するとき、ユーザ自身でエラーチェックを行わなければなりません。すなわち、コードは、予想されるフィールドに対してチェックすることを保証しなければなりません。

例として、水源での様々な毒素のレベルを時間毎に測定した記録データの集合を考えます。データは、15 の観測点からなり、各観測点は、3 つの測定値をもちます。

このデータを 15 個の構造体からなる配列に構造化することができます。各構造体は、3 つのフィールドをもち、各フィールドは得られた 3 つの測定値に対するものです。

関数 concen は、特定の特徴をもつ構造体の配列に下記のように操作します。その引数は、フィールド lead, mercury, chromium を含まなければなりません。

```
function [r1,r2] = concen(toxtest);
% 2 要素ベクトルで、r1 は、各観測点での mercury と lead の比、
% r2 は、lead と chronium の比です。
r1 = [toxtest.mercury]./[toxtest.lead];
r2 = [toxtest.lead]./[toxtest.chromium];
% 同一プロット上に、種々の色を使って、
% lead, mercury ,chromium の濃度のプロット
lead = [toxtest.lead];
mercury = [toxtest.mercury];
chromium = [toxtest.chromium];
plot(lead,'r'); hold on
plot(mercury,'b')
plot(chromium,'y'); hold off
```

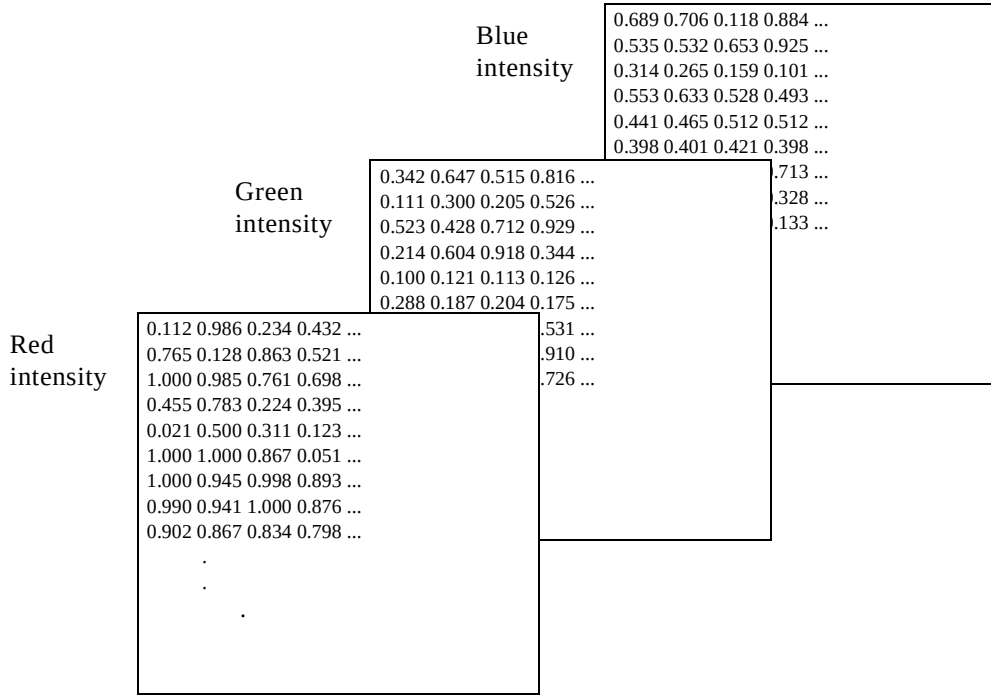
test のようなサンプル構造体配列を使って、この関数を試してください。

```
test(1).lead = .007; test(2).lead = .031; test(3).lead = .019;
test(1).mercury = .0021; test(2).mercury = .0009;
test(3).mercury = .0013;
test(1).chromium = .025; test(2).chromium = .017;
test(3).chromium = .10;
```

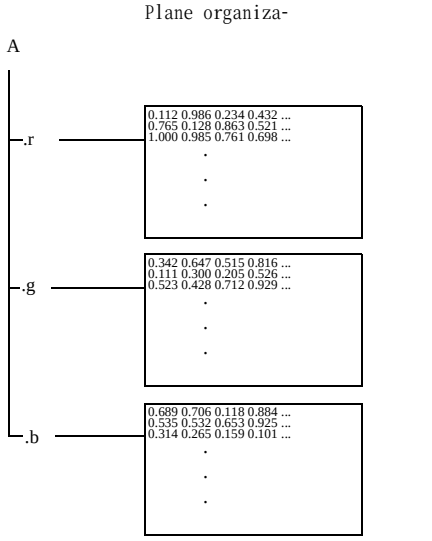
構造体配列内のデータの構造化

構造体配列の構造化の鍵は、情報のサブセットへどのようにアクセスしたいかを決定することです。これは構造体をもつ配列をどのように作るか、また構造体のフィールドをどのように分解するかを決定します。

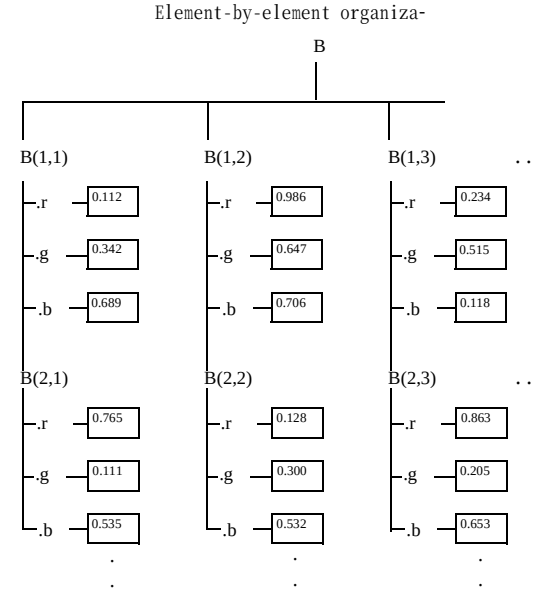
たとえば、3 つの異なる配列 RED，GREEN，BLUE に格納された 128 行 128 列の RGB イメージを考えます。



このようなデータを構造体配列に構造化する方法が少なくとも 2 つあります。



1-by-1 structure array where each field is a



128-by-128 structure array where each field is a single data

面の構造化

上記の case 1 の場合、構造体の各フィールドがイメージ面全体になります。つぎのようにして、この構造体を作成できます。

```
A.r = RED;
A.g = GREEN;
A.b = BLUE;
```

この方法を使って、一度にイメージ全体に渡り、機能する表示やフィルタリング、他の処理を行ったイメージの面全体を抽出ができます。たとえば、赤の面にアクセスするためには、つぎのようにします。

```
red_plane = A.r;
```

面の構造化は、この場合に複数のイメージに拡張可能である長所があります。イメージが多数あれば、それらをイメージ全体を含む A(2), A(3) 等として格納できます。

面の構造化の短所は、面のサブセットへのアクセスが必要なときに明らかになります。たとえば、サブイメージにアクセスするためには、各フィールドに別々にアクセスする必要があります。

```
red_sub = A.r(2:12,13:30);
grn_sub = A.g(2:12,13:30);
blue_sub = A.b(2:12,13:30);
```

要素単位の構造化

右のケースは、データのサブセットのアクセスが容易である長所を示します。この構造化により、データをセットアップするためには、つぎのようにします。

```
for i = 1:size(RED,1)
   for j = 1:size(RED,2)
      B(i,j).r = RED(i,j);
      B(i,j).g = GREEN(i,j);
      B(i,j).b = BLUE(i,j);
   end
end
```

要素単位の構造化を使って、単一のステートメントでデータのサブセットにアクセスできます。

```
Bsub = B(1:10,1:10);
```

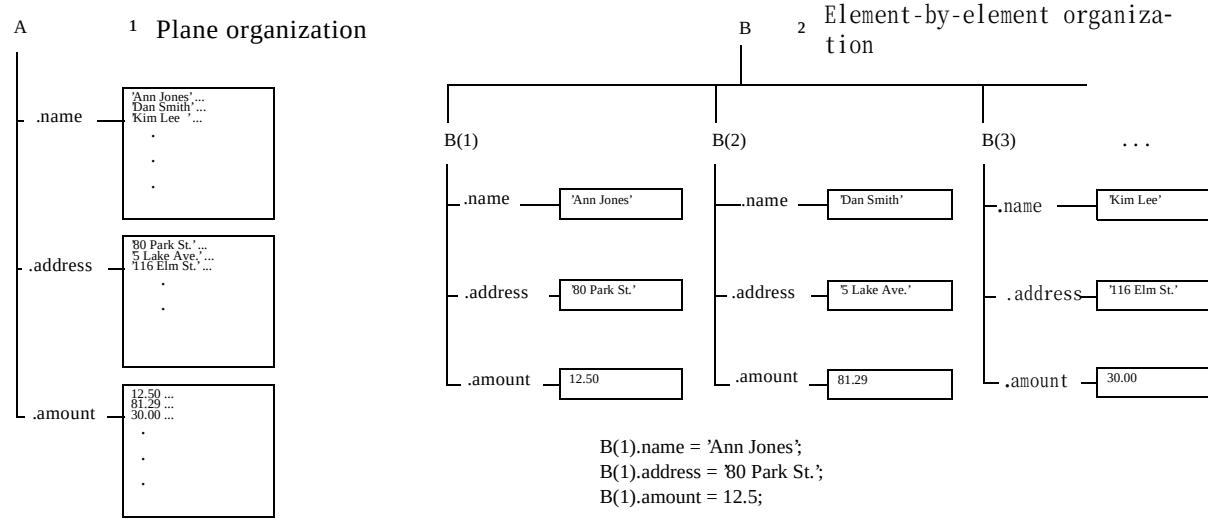
しかし、要素単位の方法を使ってイメージの面全体にアクセスするためには、ループが必要です。

```
red_plane = zeros(128,128);
for i = 1:(128*128)
    red_plane(i) = B(i).r;
end
```

要素単位の構造化は、ほとんどのイメージ処理のアプリケーションに対しては、最良の構造体配列の選択にはなりません。しかし、構造体フィールドの対応するサブセットにルーチン的にアクセスする必要がある他のアプリケーションに対しては最も良い方法かもしれません。つぎの節の例は、このタイプのアプリケーションを説明するものです。

例題 - 簡単なデータベース

簡単なデータベースの構造化を考えます。




```
A.name = strvcat('Ann Jones','Dan Smith',...);
A.address = strvcat('80 Park St.','5 Lake Ave.',...);
A.amount = [12.5;81.29;30; ...];
```

```
B(1).name = 'Ann Jones';
B(1).address = '80 Park St.';
B(1).amount = 12.5;

B(2).name = 'Dan Smith';
B(2).address = '5 Lake Ave.';
B(2).amount = 81.29;
    .
    .
    .
```

各々の可能な構造化は、データにどのようにアクセスするかにより、異なる長所があります。

- 面の構造化は、すべてのフィールド値を一度に簡単に操作できます。たとえば、amount フィールドのすべての値の平均を求めます。
  - 面の構造化を使うと、

    ```
    avg = mean(A.amount);
    ```

  - 要素単位の構造化を使うと、

    ```
    avg = mean([B.amount]);
    ```

- 要素単位の構造化は、1 つのクライアントに関連するすべての情報に簡単にアクセスできます。与えられたクライアント名とアドレスをスクリーン上に表示する M- ファイル client.m を考えます。

  面の構造化を使って、個々のフィールドを渡します。

  ```
  function client(name,address)
  disp(name)
  disp(address)
  ```

  関数 client を呼び出すにはつぎのようにします。

  ```
  client(A.name(2,:),A.address(2,:))
  ```

  要素単位の構造化を使って、構造体全体を渡します。

  ```
  function client(B)
  disp(B)
  ```

  関数 client を呼び出すにはつぎのようにします。

  ```
  client(B(2))
  ```

- 要素単位の構造化は、文字列配列のフィールドを簡単に拡張することができます。面の構造化に対して前もって文字列の長さの最大値がわからない場合、長い文字列を適応させるために、name フィールドや address フィールドを頻繁に作成し直す必要があるかもしれません。

データは、選択する構造化の方法とは一般に無関係です。ユーザはデータへのアクセスやデータの操作をどのようにしたいかを考えなければなりません。

入れ子型の構造体

構造体フィールドは、他の構造体や構造体からなる配列さえも含むことができます。構造体を作成したら、既存の構造体フィールド内に構造体を入れ子にするために関数 struct や直接代入ステートメントを使うことができます。

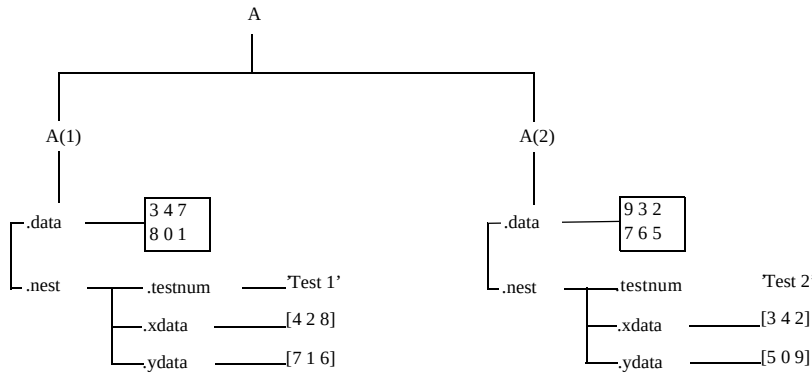
関数 struct を使って、入れ子構造体を作る

入れ子型の構造体を作るためには、関数 struct へのコールを入れ子にすることができます。たとえば、1 行 2 列の構造体配列を作成します。

```
A = struct('data',[3 4 7; 8 0 1],'nest',...
    struct('testnum','Test 1', 'xdata',[4 2 8],...
    'ydata',[7 1 6]));
```

直接代入ステートメントを使って入れ子構造体配列を作成することができます。以下のステートメントは、配列に 2 番目の要素を追加します。

```
A(2).data = [9 3 2; 7 6 5];
A(2).nest.testnum = 'Test 2';
A(2).nest.xdata = [3 4 2];
A(2).nest.ydata = [5 0 9];
```



入れ子型の構造体のインデックス

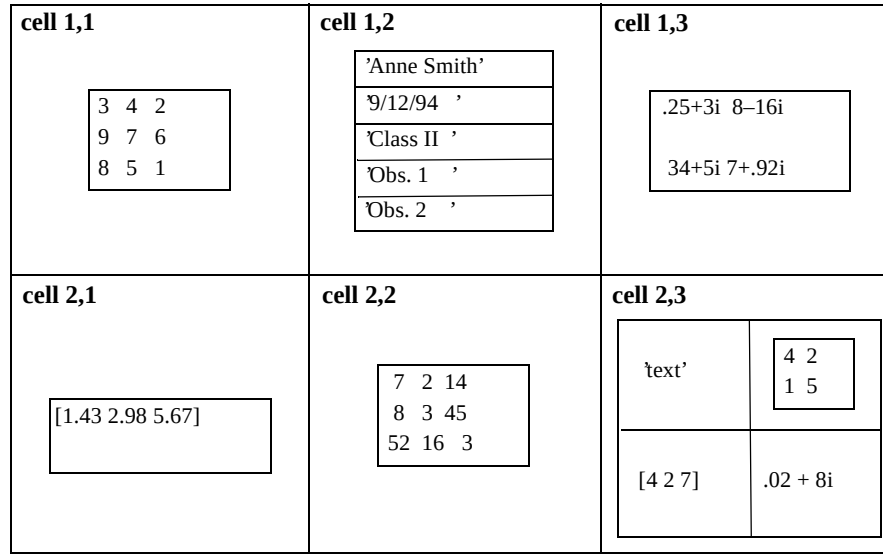
入れ子型の構造体にインデックスを付けるために、ドット表記を使って入れ子になったフィールド名を追加します。インデックス付きの式の最初のテキスト文字列は構造体配列を示し、続く式は他の構造体を含むフィールド名にアクセスします。

たとえば、以前に作成した配列 A は、2 つのレベルの入れ子をもちます。

- A(1) 内の入れ子型の構造体にアクセスするためには、A(1).nest を使ってください。
- A(2) 内の入れ子型の構造体の xdata フィールドにアクセスするためには、A(2).nest.xdata を使ってください。
- A(1) 内の ydata フィールドの要素 2 にアクセスするためには、A(1).nest.ydata(2) を使ってください。

セル配列

セル配列 は、他の MATLAB 配列をもつことができるコンテナであるセルを要素とする MATLAB 配列です。たとえば、セル配列の中の 1 つのセルが実数行列を、別のセルがテキスト文字列の配列を、さらに他のセルが複素数の値のベクトルを含むことも可能です。



多次元構造体配列を含む任意の有効なサイズまたは型のセル配列を作ることができます。

注意 この節の例は、2 次元セル配列を扱います。高次のセル配列の例については、" 多次元配列 " を参照してください。

以下の一覧は、本節の内容をまとめたものです。

- " セル配列の作成 "
- " セル配列からデータを得る "
- " セルの削除 "
- " セル配列の変更 "

- " セル配列を使って、変数のリストを置き換える "
- " 関数と演算子の適用 "
- " セル配列内のデータの構造化 "
- " セル配列を入れ子にする "
- " セル配列と数値配列の変換 "
- " 構造体のセル配列 "

セル配列の作成

つぎの方法でセル配列を作成することができます。

- 代入ステートメントの使用
- 関数 cells を使って、配列を前もって設定し、セルにデータを代入します。

代入ステートメントの使用

一度に、1 つのセルにデータを代入することで、セル配列を作ることができます。MATLAB は、ユーザがその作業を続けるに従い自動的にセル配列を作ります。セルにデータを代入する方法は 2 つあります。

- セルにインデックスを付ける

  Enc 通常の配列の表記を使って、セルのサブスクリプトを括弧で囲んでください。代入ステートメントの右辺でセルの内容を中括弧、"{}" で囲んでください。たとえば、2 行 2 列のセル配列 A を作成するには、つぎのようにします。

```
A(1,1) = {[1 4 3; 0 5 8; 7 2 9]};
A(1,2) = {'Anne Smith'};
A(2,1) = {3+7i};
A(2,2) = {-pi:pi/10:pi};
```

注意 "[]" が数値配列に対する空行列を意味するように、表記 "{}" は空のセル配列を意味します。任意のセル配列の代入に空のセル配列を使うことができます。

- 内容にインデックスを付ける

  通常の配列の表記法を使って、セルのサブスクリプトを括弧で囲んでください。代入ステートメントの右辺でセルの内容を指定してください。

```
A{1,1} = [1 4 3; 0 5 8; 7 2 9];
```

```
A{1,2} = 'Anne Smith';
A{2,1} = 3+7i;
A{2,2} = –pi:pi/10:pi;
```

このガイドの種々の例は、1 つの構文だけを使わずに、セルと内容の設定の代表的な使用法を示しています。2 つの形式を交互に使うことができます。

注意　前を付けた数値配列が既に存在する場合、数値配列を消去せずに代入により同じ名前のセル配列を作成しないでください。数値配列を消去しなければ、MATLAB は、ユーザがセルと数値の構文を混ぜて使おうとしたと仮定し、エラーを出力します。同様に、MATLAB はユーザがあるセル配列に代入を行うときに、そのセル配列か、既に存在しているとそのセル配列を消去しません。この節の例が予期しない結果になった場合は、ワークスペースからセル配列を消去して再度行ってください。

MATLAB は、短縮型でセル配列 A を表示します。

```
A =

    [3x3 double]    'Anne Smith'
[3.0000+ 7.0000i]    [1x21 double]
```

セルのすべての内容を表示するためには、関数 celldisp を使ってください。セルの構造の高水準のグラフィカル表示には、cellplot を使ってください。

カレントの配列の次元以外のセルにデータを代入する場合、MATLAB は自動的にユーザが指定したサブスクリプトを含むように配列を拡張します。MATLAB は、挿入したセルを空行列で満たします。たとえば、下記の割り当ては、2 行 2 列のセル配列 A を 3 行 3 列のセル配列にします。

```
A(3,3) = {5};
```

cell 1,1 1 4 3 0 5 8 7 2 9	**cell 1,2** 'Anne Smith'	**cell 1,3** [ ]
cell 2,1 3+7i	**cell 2,2** [–3.14...3.14]	**cell 2,3** [ ]
cell 3,1 [ ]	**cell 3,2** [ ]	**cell 3,3** 5

セル配列の構文： 括弧を使って

鍵括弧が数値配列を作るのと同じように、中括弧 "{}" はセル配列を作ります。セルの入れ子を示すために中括弧を入れ子にできること以外は、中括弧は鍵括弧と同様に機能します (詳細は、" セル配列を入れ子にする " を参照してください)。

中括弧は、列の終わりを表わすために、カンマ、または、スペースを、セル間の行の終わりを表わすためにセミコロンを使います。たとえば、

```
C = {[1 2], [3 4]; [5 6], [7 8]};
```

は、つぎのようになります。

cell 1,1 [1 2]	**cell 1,2** [3 4]
cell 2,1 [5 6]	**cell 2,2** [7 8]

数値配列に対するのと同様に、セル配列を連結するために鍵括弧を使ってください。

関数 cell を使って、セル配列を前もって設定

関数 cell を使って指定したサイズの空のセル配列を前もって設定します。たとえば、空の 2 行 3 列のセル配列を作成するためには、つぎのようにします。

```
B = cell(2,3);
```

B のセルを満たすために代入ステートメントを使ってください。たとえば、つぎのようにします。

```
B(1,3) = {1:3};
```

セル配列からデータを得る

セル配列からデータを得たり、標準配列、または、新しいセル配列として結果を格納できます。この節では、つぎのことを説明します。

- 内容のインデックスを使って、セルの内容にアクセスします。
- セルのインデックスを使って、セルのサブセットにアクセスします。

内容のインデックスを使って、セルの内容にアクセスする

1 つのセルのデータの一部、または、すべてにアクセスするために、代入ステートメントの右辺で内容のインデックスを使うことができます。代入ステートメントの左辺でセルの内容を受け取るために変数を指定してください。代入ステートメントの右辺のセルのインデックス表現を中括弧で囲んでください。これはセル自身ではなく、セルの内容を代入していることを意味します。

2 行 2 列のセル配列 N を考えます。

```
N{1,1} = [1 2; 4 5];
N{1,2} = 'Name';
N{2,1} = 2-4i;
N{2,2} = 7;
```

つぎのようにして、N{1,2} 内の文字列を得ることができます。

```
c = N{1,2}

c =

Name
```

注意　代入において、内容のインデックスを使って、単一のセルにはアクセスできますが、セルのサブセットにはアクセスできません。たとえば、ステートメント A{1,:} = *value* と B = A{1,:} は共に正しくありません。しかし、カンマで分離した変数を使う部分（たとえば関数入力としてまたは配列の作成時に）では、セルのサブセットを使うことができます。詳細は、" セル配列を使って、変数のリストを置き換える " を参照してください。

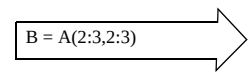
セルの内容のサブセットを得るためには、インデックスの式を連結します。たとえば、セル N{1,1} 内の配列の要素 (2,2) を得るためには、つぎのようにします。

```
d = N{1,1}(2,2)

d =

     5
```

セルのインデックスを使って、セルのサブセットにアクセスする

新しいセル配列を作成しながら、他の変数に任意のセルの集合を代入するためには、セルのインデックスを使ってください。セル配列内のセルのサブセットにアクセスするためには、コロン演算子を使ってください。

cell 1,1 3	**cell 1,2** 5	**cell 1,3** 9
cell 2,1 5	**cell 2,2** 6	**cell 2,3** 0
cell 3,1 4	**cell 3,2** 7	**cell 3,3** 2



cell 1,1 6	**cell 1,2** 0
cell 2,1 7	**cell 2,2** 2

セルの削除

単一のステートメントを使って、セルの次元全体を削除できます。通常の配列の削除のように、セルの行、または、列を削除するときにベクトルのサブスクリプトを使い、空行列を次元に代入してください。

```
A(cell_subscripts) = []
```

セルを削除するとき、中括弧は代入ステートメントには全く現れません。

セル配列の変更

他の配列のように、関数 reshape を使ってセル配列を変更できます。セルの数は変更後も同じでなければなりません。セルの追加や削除には、reshape は使用できません。

```
A = cell(3,4);
size(A)

ans =
    3     4

B = reshape(A,6,2);
size(B)

ans =
    6     2
```

セル配列を使って、変数のリストを置き換える

セル配列は、カンマで分離したMATLAB変数のリストで置き換えることができます。

- 関数入力リスト
- 関数出力リスト
- 表示操作
- 配列の構造 (鍵括弧と中括弧)

複数のセルのインデックス付けのためにコロンと中括弧表記を一緒に使った場合、MATLAB は、各セルの内容を別々の変数として扱います。たとえば、各セルが別々のベクトルを含むセル配列 T があると仮定します。式 T{1:5} は、T の最初の 5 つのセルのベクトルのカンマで分離したものと等価です。

セル配列 C を考えます。

```
C(1) = {[1 2 3]};
C(2) = {[1 0 1]};
C(3) = {1:10};
C(4) = {[9 8 7]};
C(5) = {3};
```

conv を使って、C(1) と C(2) のベクトルを畳込みます。

```
d = conv(C{1:2})

d =

     1     2     4     2     3
```

ベクトル 2,3,4 を表示します。

```
C{2:4}

ans =

    1    0    1

ans =

    1    2    3    4    5    6    7    8    9   10

ans =

    9    8    7
```

同様に、つぎのステートメントで新しい数値配列を作成することができます。

```
B = [C{1}; C{2}; C{4}]

B =

    1    2    3
    1    0    1
    9    8    7
```

別々の出力引数を表わすセルをもつ新しいセル配列を作成するため、代入ステートメントの左辺に内容を指示するインデックスを使うこともできます。

```
[D{1:2}] = eig(B)
```

```
D =

    [3x3 double]    [3x3 double]
```

D{1} と D{2} を使って、実際の固有値と固有ベクトルを表示できます。

注意 引数 varargin と varargout を使って、ユーザが作成する MATLAB 関数に対する可変の入出力引数の数を指定できます。varargin と varargout はセル配列で、様々なサイズや種類の MATLAB データをもつことができます。詳細は 17-16 ページの " 転送する引数の数を可変にする " を参照してください。

関数と演算子の適用

セルの内容に関数や演算子を適用するためにインデックスを使います。たとえば、単一のセルの内容を引数として使って、関数をコールするために内容のインデックスを使います。

```
A{1,1} = [1 2; 3 4];
A{1,2} = randn(3,3);
A{1,3} = 1:5;
B = sum(A{1,1})

B =

     4     6
```

入れ子型でないセル配列のいくつかのセルに関数を適用するために、ループを使います。

```
for i = 1:length(A)
    M{i} = sum(A{1,i});
end
```

セル配列内のデータの構造化

セル配列は、異なるサイズや種々のデータからなるデータの構造化に有効です。セル配列は、つぎのようなアプリケーションに対しては構造体よりも有効です。

- 1 つのステートメントでデータの複数のフィールドにアクセスする必要がある場合。

- カンマで分離した変数群のようなデータのサブセットにアクセスしたい場合。
- 決まったフィールド名の集合がない場合。
- 構造体からルーチン的にフィールドを削除する場合。

1 つのステートメントで複数のフィールドにアクセスする例として、データがつぎのものから成り立つと仮定します。

- 実験で得られた測定値からなる 3 行 4 列の配列
- 実験者名を含む 15 文字の文字
- 5 回の実験から得られた測定記録を含む 3-4-5 の配列

多くのアプリケーションに対して、このデータに対するデータ構造で最も良いものは構造体です。しかし、情報の最初の 2 フィールドのみをルーチン的にアクセスする場合は、インデックス付けの目的のためにはセル配列がより便利でしょう。

つぎの例は、セル配列 TEST の最初と 2 番目の要素へのアクセス方法を示しています。

```
[newdata,name] = deal(TEST{1:2})
```

つぎの構造体 TEST の最初と 2 番目の要素へのアクセス方法を示しています。

```
newdata = TEST.measure
name = TEST.name
```

引数 varargin と varargout は、カンマで分離されたものの代わりのセル配列のユーティリティの例です。3 行 3 列の数値配列 A を作成します。

```
A = [0 1 2;4 0 7;3 1 2];
```

A に関数 normest(2- ノルム推定) を適用し、B の個々のセルに関数出力を代入します。

```
[B{1:2}] = normest(A)

B =

    [8.8826]    [4]
```

関数からのすべての出力値は B の別々のセルに格納されます。B(1) は、ノルム推定を含み、B(2) は繰り返し回数を含みます。

セル配列を入れ子にする

セルは、他のセル配列を含んだり、セル配列の配列さえも含むことができます (セルでないデータを含んでいるセルは、*leaf cells* と言われます)。ユーザは、入れ子の中括弧、セル関数、直接代入ステートメントを使って、入れ子のセル配列を作ります。そして、個々のセル、セルのサブ配列、セル要素にアクセスしたり、それらを取り扱うことができます。

入れ子型の中括弧を使って、入れ子型の配列を作る

入れ子型のセル配列を作成するために中括弧の組を入れ子にすることができます。たとえば、

```
clear A
A(1,1) = {magic(5)};
A(1,2) = {{[5 2 8; 7 3 0; 6 7 3] 'Test 1'; [2–4i 5+7i] {17 []}}}

A =

    [5x5 double]    {2x2 cell}
```

代入ステートメントの右辺は、2 組の中括弧で囲まれることに注意してください。最初の組は、セル配列 A のセル (1,2) を表わします。2 つ目の組は、2 行 2 列のセル配列を外側のセルに "package" します。

関数 cell を使って、入れ子の配列を作る

関数 cell を使って、セル配列を入れ子にするためには、cell の出力を既存のセルに割り当てます。

1 1 行 2 列の空のセル配列を作成します。

```
A = cell(1,2);
```

2 A(1,2) 内に 2 行 2 列のセル配列を作成します。

```
A(1,2) = {cell(2,2)};
```

3 代入ステートメントを使って、入れ子型の配列を含む A を満たします。

```
A(1,1) = {magic(5)};
A{1,2}(1,1) = {[5 2 8; 7 3 0; 6 7 3]};
A{1,2}(1,2) = {'Test 1'};
A{1,2}(2,1) = {[2–4i 5+7i]};
A{1,2}(2,2) = {cell(1,2)}
```

```
A{1,2}{2,2}(1) = {17};
```

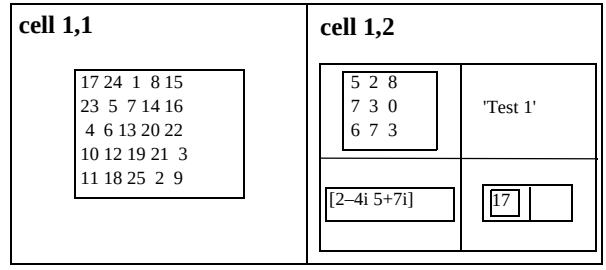
入れ子型のサブスクリプトの最終レベルまでの中括弧の使用に注意してください。これは、セル内でセルにアクセスするためには、セルの内容にアクセスする必要があるからです。

上記のステップ 3 に示したステートメントを使って、直接代入により入れ子型のセル配列を作ることもできます。

入れ子型のセル配列のインデックス

入れ子型のセルにインデックスを付けるため、インデックス付けの式を結合します。サブスクリプトの最初の組は、セルの一番上の層にアクセスし、つぎの括弧の組は、つぎの層にアクセスします。

たとえば、配列 A は 3 レベルの入れ子をもちます。



- セル (1,1) の 5 行 5 列の配列にアクセスするためには、A{1,1} を使います。
- セル (1,2) の位置 (1,1) にある 3 行 3 列の配列にアクセスするためには、A{1,2}{1,1} を使ってください。
- セル (1,2) の 2 行 2 列のセル配列にアクセスするためには、A{1,2} を使ってください。
- セル (1,2) の位置 ((2,2) にある空のセルにアクセスするためには、A{1,2}{2,2}{1,2} を使ってください。

セル配列と数値配列の変換

セルと数値の書式の変換には、for ループを使ってください。たとえば、セル配列 F を作成します。

```
F{1,1} = [1 2; 3 4];
F{1,2} = [–1 0; 0 1];
```

```
F{2,1} = [7 8; 4 1];
F{2,2} = [4i 3+2i; 1–8i 5];
```

F の内容を数値配列 NUM にコピーするために、3 つの for ループを使ってください。

```
for k = 1:4
   for i = 1:2
      for j = 1:2
         NUM(i,j,k) = F{k}(i,j);
      end
   end
end
```

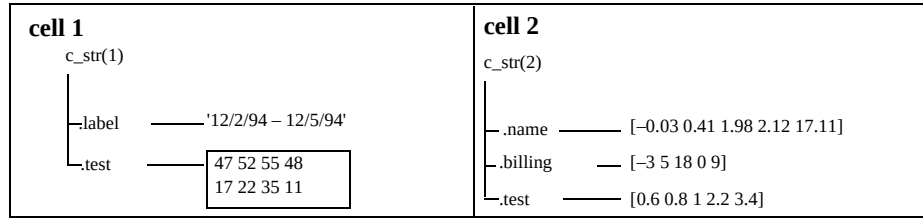
同様に、数値配列の各値をセル配列の単一セルに代入するためには、for ループを使わなければなりません。

```
G = cell(1,16);
for m = 1:16
   G{m} = NUM(m);
end
```

構造体のセル配列

異なるフィールドの構造をもつ構造体の集合をセル配列を使って格納します。

```
c_str = cell(1,2);
c_str{1}.label = '12/2/94 – 12/5/94';
c_str{1}.obs = [47 52 55 48; 17 22 35 11];
c_str{2}.xdata = [–0.03 0.41 1.98 2.12 17.11];
c_str{2}.ydata = [–3 5 18 0 9];
c_str{2}.zdata = [0.6 0.8 1 2.2 3.4];
```



配列 c_str のセル 1 は、文字列とベクトルの 2 つのフィールドをもつ構造体を含みます。セル 2 は、3 つのベクトルのフィールドをもつ構造体を含みます。

構造体からなるセル配列を作成するときは、内容のインデックスを使わなければなりません。同様に、セル内の構造体の内容を得るためには、内容のインデックスを使わなければなりません。内容のインデックスの構文は、つぎのようになります。

```
cell_array{index}.field
```

たとえば、セル 1 の構造体の label フィールドにアクセスするためには、c_str{1}.label を使ってください。

関数ハンドル

関数ハンドルを利用する利点 21-3
簡単な関数ハンドル 21-5

関数ハンドルの構築 21-7
関数名の最大長 21-7

ハンドルを使って関数を計算 21-8
関数の計算と多重定義 21-8
関数ハンドルの計算の例題 21-9

関数ハンドル情報の表示 21-12

関数ハンドルの演算 21-13
関数ハンドルを関数名に変換 21-13
関数名を関数ハンドルに変換 21-14
データタイプのテスト 21-15
等式に対するテスト 21-15

関数ハンドルのセーブとロード 21-17

エラー状態の取り扱い 21-18
存在しない関数へのハンドル 21-18
関数ハンドルコンストラクタの中に含まれるパス . . . 21-18
非スカラ関数ハンドルの計算 21-19

履歴について - 関数名の計算 21-20

関数 ハンドル は、MATLAB が関数を実行したり、計算したりするために必要な関数のすべての情報をもつ MATLAB のデータタイプです。ユーザは、任意の MATLAB 関数へのハンドルを作成することができ、関数を参照する方法として、ハンドルを使います。関数ハンドルは、一般的には、引数リストの中から、他の関数に渡され、その関数は、ハンドルを使って、計算することができます。

MATLAB 関数ハンドルは、関数への参照以外にも使われます。関数ハンドルは、他の異なる引数タイプを取り扱うために、関数メソッドの選択や多重定義の選択を表すこともしばしばあります。ユーザが、関数へ渡すハンドルを作成すると、MATLAB は、パス上に存在するその名前のすべての組み込み、M- ファイルメソッドのスナップショットを取ります。そして、ハンドルの中のこれらのメソッドのすべてに対して、アクセス情報をストアします。

関数ハンドルを計算する場合、MATLAB は、ハンドルが作成されたときに、ハンドルの中に組み込まれた関数のみを考えます。パス上、または、スコープの中に存在する他の関数は、考えません。これは、ハンドル内に存在するどの関数を組み合わせ、ハンドルが計算するために必要な引数はどれかを決め、MATLAB が送り出す実際の関数を決定します。

ここでは、以下のトピックスを説明します。

- " 関数ハンドルを利用する利点 "
- " 関数ハンドルの構築 "
- " ハンドルを使って関数を計算 "
- " 関数ハンドル情報の表示 "
- " 関数ハンドルの演算 "
- " 関数ハンドルのセーブとロード "
- " エラー状態の取り扱い "
- " 履歴について - 関数名の計算 "

関数ハンドルを利用する利点

関数ハンドルを使って、以下のことが行えます。

- 関数アクセス情報を他の関数に渡す
- 多重定義関数ののすべてのメソッドを取り込む
- サブ関数やプライベート関数に広くアクセスできる
- 関数を計算した場合、信頼性を高める
- ユーザ関数を定義するファイル数を減らせる
- 繰り返し演算の中で、性能を高める
- 配列、構造体、セル配列の中のハンドルを取り扱う

この概要の節は、簡単な関数ハンドルを使った例題も含んでいます。21-5 ページの " 簡単な関数ハンドル " を参照してください。

関数アクセス情報を他の関数に渡す

関数ハンドルをコール時に引数として、他の関数に渡すことができます。ハンドルは、アクセス情報を含んでいて、それを使って、渡された関数が、ハンドルにくっ付いている関数をコールすることができます。

関数のハンドルが、関数を計算する部分に存在していない場合さえも、他の関数の中から関数ハンドルを計算することができます。これは、計算を行う関数が、関数ハンドルの中に必要なすべての情報をもっているからです。

同じ理由で、ハンドルの関数が MATLAB のサーチパス上になくても、関数ハンドルを計算することができます。

ユーザは、MATLAB の feval コマンドを使って、関数ハンドルの中の関数を計算する必要があります。関数ハンドルを引数として他の関数に渡す場合、ハンドルを受け取った関数は、関数ハンドルを計算するために、feval を使います。

多重定義関数のハンドルの取得

多くの MATLAB 関数は多重定義されているので、関数ハンドルは、関数と実現するいくつかのコードソース (たとえば、組み込みコード、M- ファイル) にマッピングされています。関数ハンドルは、ハンドルが作成された時点で MATLAB パス上に存在する、多重定義されたソースまたは*メソッド* のすべてへのアクセスをストアします。

多重定義関数ハンドルを計算するとき、MATLAB は、計算するために、どのメソッドを選択するかと、関数コールの中で渡される引数タイプの選択に関して、一般

的なルールに従っています。MATLAB がどのように多重定義関数を選択するかに関する情報については、22-65 ページの "MATLAB がメソッドを読み込む優先順位の決定法 " を参照してください。

たとえば、標準の MATLAB パス上に関数 abs を定義する 3 つの組み込み関数と 1 つの M- ファイル関数があります。関数 abs に対して作成される関数ハンドルは、これら関数ソースの4つすべてに関するアクセス情報を含んでいます。タイプdoubleを引数として関数を計算する場合、double 引数をもつ組み込み関数が実行されます。

サブ関数やプライベート関数に広くアクセスできる

定義により、すべての MATLAB 関数は、あるスコープをもっています。これらは、そのスコープの中で、他の MATLAB 要素を可視化できますが、その外部では可視化できません。そのスコープの中にある他の関数から関数を直接起動することもできますが、そのスコープの外部の関数からはできません。

サブ関数とプライベート関数は、他の MATLAB 関数に対して、可視性に関して制限を与えるように設計されています。同じ M- ファイルの中で定義された他の関数によってのみサブ関数を起動できます。ユーザは、サブディレクトリ \private のすぐ上位のデレクトリの中の関数からのみプライベート関数を起動できます。

制限のあるスコープをもつ関数へのハンドルを作成する場合、関数ハンドルは、MATLAB環境の中のある位置から関数を計算するために必要な情報をすべて含むことができます。サブ関数がスコープの中に位置していて、サブ関数へのハンドルを作成する場合 (すなわち、サブ関数を定義する M- ファイルの中にそれを作成する)、ユーザは、ハンドルを M- ファイルの外側に位置するコードに渡し、通常のスコープを超えて、サブ関数を計算します。同様なことが、プライベート関数にも適用されます。

関数を計算するときの信頼性の保証

関数ハンドルを使って、定義したスコープ内の関数のみを含ませると、ユーザは、MATLABが関数ハンドルを計算するときに実行される関数全体をコントロールすることができます。これは、計算するときに、スコープの中に位置する関数から知らなくて良いものを分離することができます。ユーザは、関数ハンドルが計算される場合、実行しているどのメソットを得るかのコントロールを行います。

関数を定義するファイル数の増加を押さえる

関数ハンドルを使って、ユーザの関数を定義するために必要な M- ファイルの数を少なくする手助けを行うことができます。一つの M- ファイルの中にいくつかの関数をまとめることに付随する問題は、これらをサブ関数と定義することで、それ

により、MATLAB の中のでそのスコープを減らすことができます。関数ハンドルを使って、サブ関数にアクセスして、この制限を取り去ります。ユーザが、取り扱う必要のあるファイル数を減らしたい場合、関数をグループ化することにより、可能になります。

繰り返し演算の中での性能の向上

MATLAB は、関数ハンドルを作成した時点で関数についてのルックアップを行い、ハンドル自身の中のアクセス情報をストアします。一度定義すると、繰り返し計算の中で、このハンドルを使うことができ、関数のルックアップを毎回行なうことによるパフォーマンスの劣化を起こさないようにします。

関数ハンドルは、配列、構造体、セル配列で操作されます

標準の MATLAB データタイプとして、関数ハンドルは、他の MATLAB データタイプと同じ方法で取り扱い、演算できます、ユーザは、関数ハンドルの配列、構造体、セル配列を作成することができます。数値配列や構造体の要素にアクセスする方法と同様に、これらのデータ構造体の中の個々の関数ハンドルにアクセスします。

MATLAB 配列の他のタイプを作成するために使用する連結メソッド、[]、または、cat のいずれかを使って、n 次元の配列のハンドルを作成します。行列取り扱いを含むすべての演算は、関数ハンドルに対してサポートされています。

簡単な関数ハンドル

関数 repmat は、基本的な行列関数で、単一の M- ファイル repmat.m を使って、MATLABに定義されたものです。関数ハンドルを関数repmatに作成する場合、MATLAB は、ハンドルの中に、後で関数を計算するために必要な情報をストアします。この情報の中に含まれるものは、repmat.m ソースのファイルパス、toolbox\matlab\elmat\repmat.m です。

関数ハンドルを一度作成すると、MATLAB環境の中の変更による影響を受けません。たとえば、関数 repmat へのハンドルを作成して、後で、付加的な repmat.m メソッドを記述して、関数を多重定義した場合、ハンドルは、オリジナルの関数にのみ見ることができます。また、サーチパスから repmat.m へのパスを除去すると、MATLAB は、パスを変更する前に作成したハンドルを使って、関数を配置し、計算することができます。

関数 repmat は、この場合、多重定義したものでないので、このハンドルを通しての関数の計算は、かなり簡単です。関数ハンドルに、feval をコールすることができ、関数が機能するようにいくつかの引数を渡すこともできます。MATLAB は、一つの関数を実行して、そのアクセス情報をハンドルにストアします。

関数ハンドルの構築

関数の前に、符号 @ を使って、MATLAB の中に関数ハンドルを作成します。つぎの例題は、関数 humps に対する関数ハンドルを作成し、それを変数 fhandle に割り当てるものです。

```
fhandle = @humps;
```

ユーザが渡したい引数を同じ方法で他の関数にハンドルを渡します。つぎの例題は、ちょうど作成した関数ハンドルを fminbnd に渡して、区間 [0.3, 1] で最小化するものです。

```
x = fminbnd(fhandle, 0.3, 1)
x =
    0.6370
```

関数 fminbnd は、feval を使って、関数ハンドル @humps を計算するものです。M-ファイル fminbnd の一部を、つぎに示します。ライン 1 で、入力 funfcn パラメータは、渡される関数ハンドル @humps を受け取ります。ステートメント feval のライン 113 は、ハンドルを計算します。

```
1     function [xf,fval,exitflag,output] = ...
          fminbnd(funfcn,ax,bx,options,varargin)
              .
              .
              .
113   fx = feval(funfcn,x,varargin{:});
```

注意　関数ハンドルを作成すると、符号 @ を前に付けた関数 name を使用するだけです。これは、任意のパス情報を含んでいるものではありません。つぎのシンタックスは、正しくありません。fhandle = @\home\user4\humps

---

関数名の最大長

ハンドルに使用できる関数名は、最大 31 キャラクタで、ユニークなものです。関数名がこの長さを超えると、後半を打ち切ります。

```
fhandle = @function_name_that_exceeds_thirty_one_characters
fhandle =
    @function_name_that_exceeds_thir
```

Java コンストラクタに対して作成された関数ハンドルは、パッケージ名あるいはクラス名のセグメントの長さは、31 文字を超えてはいけません (用語 *セグメント* は、ドットの前、ドットの間、ドットの後の名前の任意の部分を意味します。たとえば、java.lang.String には 3 つのセグメントがあります)。パッケージおよびクラスを指定する文字列の全体の長さには制限はありません。

以下のステートメントは、パッケージおよびクラス全体の指定子の長さが 31 文字を超えていますが、有効です。

```
fhandle = @java.awt.datatransfer.StringSelection
```

ハンドルを使って関数を計算

MATLAB コマンド feval を使って、関数ハンドルのターゲット関数を計算します。このコマンドを関数ハンドルと共に使用するシンタックスは、つぎのものです。

```
feval(fhandle, arg1, arg2, ..., argn)
```

これは、fhandle に、引数 arg1 から argn を渡して直接コールするものと同じ挙動をします。根本的な違いは、つぎのことです。

- 関数ハンドルは、ユーザが渡した関数の中から計算されます。
- MATLAB が計算用に選択したコードソースは、ハンドルが作成された時点で、MATLAB パスやスコープ内に存在していた関数を多重定義するメソッドに依存します (引数タイプもメソッド選択に影響します)。パスとスコープは、計算する時点では考えません。
- MATLAB は、関数ハンドルが作成された時点で初期関数ルックアップを行ないます。しかし、ルックアップは、MATLAB がハンドルを計算する度に必要なものではありません。

注意 コマンド feval は、非スカラ関数ハンドル上に操作しません。非スカラ関数ハンドルを feval に渡すと、エラーになります。

関数の計算と多重定義

関数ハンドルと多重定義の間の関係を理解するためには、MATLAB 関数のコールの仕方を簡単にレビューすることが手助けになります。多重定義は、いくつかのコードソースから構成される単一の MATLAB 関数を考える場合に有効になります (たとえば、組み込みや M- ファイル)。feval を使わないで MATLAB 関数をコールした場合、どちらのソースを選択するかは、つぎの 2 つの要因に依存します。

- コール時、パス上に見えるメソッド
- 関数への引数のクラス

MATLAB は、同様な方法で関数ハンドルを計算します。多くの場合、関数ハンドルは、関数を多重定義するメソッドを集めたものを表します。ユーザが、feval を使って、関数ハンドルを計算する場合、コールされる特別なメソッドの選択は、つぎの事柄に依存します。

- ハンドルが作成された時点でのパス上に見えるメソッド
- ハンドルと共に feval に渡される引数のクラス

関数ハンドルの計算の例題

この節は、関数ハンドルの使い方や計算法を示す 2 つの例題を示します。

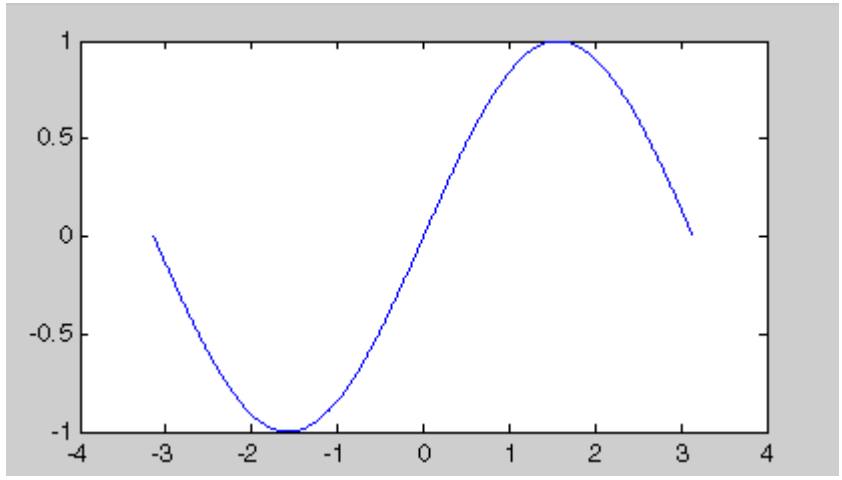
例題 1 - 簡単な関数ハンドル

plot_fhandle と云う名前の関数を定義するつぎの例題は、関数ハンドルとデータを受け取り、そのデータに関して、関数ハンドルの計算を行なうものです。

```
function x = plot_fhandle(fhandle, data)
plot(data, feval(fhandle, data))
```

関数 sin と下に示す引数を関数と共に、plot_fhandle をコールすると、求まる計算結果は、つぎのプロットを作成します。

```
plot_fhandle(@sin, -pi:0.01:pi)
```



例題 2 - 関数ハンドルとサブ関数

この例題の中の M- ファイルは、主関数 fitcurvedemo とサブ関数 expfun です。定義として、サブ関数は、自身の M- ファイルのスコープの中のみで見ることができます。もちろん、その M- ファイルの中の他の関数での使用だけは、可能です。

このコードのオーサは、この M- ファイルの範囲の外で、expfun を使おうとしています。この例題は、関数ハンドルをサブ関数 expfun に作成し、サブ関数に関するアクセス情報を捕らえ、MATLAB環境の中の任意の位置からコールするものです。関数ハンドルは、fminsearch に渡し、通常のスコープの外側でサブ関数をうまく計算するものです。

下に示すコードは、fitcurvedemo とサブ関数 expfun を定義するものです。ライン 6 は、関数ハンドルを expfun に作成し、それを変数 fun に割り当てます。ライン 16 で、fminsearch へのコールで、サブ関数の通常のスコープの外側に関数ハンドルを渡します。関数 fminsearch は、feval を使って、そのハンドルを通して、サブ関数を計算します。

```
1   function Estimates = fitcurvedemo
2   % FITCURVEDEMO
3   % Fit curve to data where user chooses equation to fit.
4
5   % Define function and starting point of fitting routine.
6   fun = @expfun;
7   Starting = rand(1, 2);
8
9   % First, we create the data.
10  t = 0:.1:10;     t=t(:);     % to make 't' a column vector
11  Data = 40 * exp(-.5 * t) + randn(size(t));
12  m = [t Data];
13
14  % Now, we can call FMINSEARCH:
15  options = optimset('fminsearch');   % Use FMINSEARCH defaults 16  Estimates
= fminsearch(fun, Starting, options, t, Data);
17
18  % To check the fit
19  plot(t, Data, '*')
20  hold on
21  plot(t, Estimates(1) * exp(-Estimates(2) * t), 'r')
22  xlabel('t')
23  ylabel('f(t)')
24  title(['Fitting to function ', func2str(fun)]);
25  legend('data', ['fit using ', func2str(fun)])
26  hold off
27
28  % --------------------------------------------------------
29
```

```
function sse = expfun(params, t, Data)
% Accepts curve parameters as inputs, and outputs fitting the
% error for the equation y = A * exp(-lambda * t);
A = params(1);
lambda = params(2);

Fitted_Curve = A .* exp(-lambda * t);
Error_Vector = Fitted_Curve - Data;

% When curve fitting, a typical quantity to minimize is the sum
% of squares error
sse = sum(Error_Vector .^ 2);
```

関数ハンドル情報の表示

コマンド functions は、デバッグ目的に役立つ、関数ハンドルに関する情報を出力します。多重定義された関数のハンドルに対しては、これは、ハンドルによって参照されるすべてのメソッドに関する情報を含みます。

functions コマンドに関する詳細は、オンラインヘルプの以下の部分を参照してください。

```
MATLAB -> Using MATLAB -> Programming and Data Types ->
Function Handles -> Displaying Function Handle Information
```

関数ハンドルの演算

MATLAB は、関数ハンドルと関数名の文字列と間で、変換を行なう 2 つの関数を用意しています。変数が関数ハンドルを保持しているかを調べるテスト用の関数も用意していて、関数ハンドルを比較することもできます。

関数ハンドルを関数名に変換

文字の比較や表示のような文字演算を、関数ハンドルで行なう必要がある場合、文字列フォーマットの中の関数名を得るために、func2strを使うことができます。関数ハンドル sin を文字列に変換するには、つぎのようにします。

```
fhandle = @sin;

func2str(fhandle)
ans =
    sin
```

注意　コマンド func2str は、非スカラの関数ハンドルの演算を行ないません。非スカラ関数ハンドルを func2str に渡すと、エラーになります。

例題 - エラーメッセージの中の関数名表示

ここで示す関数 catcherr は、関数ハンドルとデータ引数を受け入れ、そのハンドルを使って、関数を計算しようとします。関数が計算の実行に失敗すると、catcherr は、sprintf を使って、原因の関数の名前を与えるエラーメッセージを表示します。関数名は、それを表示するために、sprintf 用の文字列でなければなりません。コードは、func2str を使って、関数ハンドルから関数名を引き出します。

```
function catcherr(func, data)
try
    ans = feval(func, data);
    disp('Answer is:');
    ans
catch
    sprintf('Error executing function ''%s''\n', func2str(func))
end
```

下に示すように、catcherr への最初のコールは、ハンドルを round と正しいデータ引数に渡します。これは、正しいコールになり、期待した答えを出力します。二番目のコールは、同じ関数ハンドルと不適切なデータタイプ (MATLAB 構造体) を渡します。このとき、round は失敗して、失敗の原因の関数名を含むエラーメッセージの表示を catcherr を使って行ないます。

```
catcherr(@round, 5.432)
ans =
Answer is 5

xstruct.value = 5.432;
catcherr(@round, xstruct)
Error executing function "round"
```

関数名を関数ハンドルに変換

関数 str2func を使って、MATLAB 関数名を含む文字列から関数ハンドルを作成することができます。文字列 'sin' を、その関数に対するハンドルに変換するために、つぎのようにします。

```
fh = str2func('sin')
fh =
    @sin
```

変数の中の関数名文字列を渡す場合、変数を受ける関数は、関数名を str2func を使って、関数ハンドルに変換します。つぎの例題は、変数 funcname を関数 makeHandle に渡し、そして、関数ハンドルを作成するものです。

```
function fh = makeHandle(funcname)
fh = str2func(funcname);
% -- end of makeHandle.m file --

makeHandle('sin')
ans =
    @sin
```

関数名文字列のセル配列に、str2func 演算を適用することもできます。この場合、str2func は、関数ハンドルの配列を出力します。

```
fh_array = str2func({'sin' 'cos' 'tan'})
fh_array =
    @sin    @cos    @tan
```

例題 - より柔軟性をもつパラメータのチェック

つぎの例題では、関数 myminbnd は、最初の引数に、関数ハンドルか、または、文字列のいずれかを受け取ることを期待しています。文字列を渡す場合、myminbnd は、str2func を使って、それから関数ハンドルを作成し、そして、str2func へのコールの中で、そのハンドルを使います。

```
function myminbnd(fhandle, lower, upper)
if ischar(fhandle)
    disp 'converting function string to function handle ...'
    fhandle = str2func(fhandle);
end
fminbnd(fhandle, lower, upper)
```

ユーザが、関数ハンドルまたは関数名文字列のどちらかと共に、myminbnd をコールする場合、適切に引数を取り扱うことができます。

```
myminbnd('humps', 0.3, 1)
converting function string to function handle ...
ans =
    0.6370
```

データタイプのテスト

関数 isa は、MATLAB 変数またはオブジェクトのデータタイプ、または、クラスを識別します。変数は、タグ function_handle と共に関数 isa で使われることにより、関数ハンドルを保持しているか否かを調べます。つぎの関数は、関数ハンドルを計算しようと試みる前に、関数ハンドルであるか否かを調べるために渡される引数をテストします。

```
function evaluate_handle(arg1, arg2)
if isa(arg1, 'function_handle')
    feval(arg1, arg2);
else
    disp 'You need to pass a function handle';
end
```

等式に対するテスト

関数 isequal を使って、2 つの関数ハンドルが等しいか否かを比較します。たとえば、多重定義された関数に対して、記述された特別なメソッドを実行したいことを考えます。ユーザは、限られたスコープの中の関数ハンドルを作成し、その

メソッドのみにアクセスを与えます。つぎに示す関数 test_myfun は、最初の引数 arg1 に、この関数ハンドルを受け入れ、そして、それを計算します。

arg1 の中でハンドルを計算する前に、test_myfun は、より広い定義域をもった他の関数ハンドルに対して、それをチェックします。この他のハンドル @myfun は、関数に対するすべてのメソッドへのアクセスを与えます。arg1 の中の関数ハンドルが、複数の意図したメソッドを表している場合、エラーメッセージが表示され、関数は計算されません。

```
function test_myfun(arg1, arg2)
if isequal(arg1, @myfun)
    disp 'Function handle holds unexpected context'
else
    feval(arg1, arg2);
end
```

関数ハンドルのセーブとロード

MATLAB の save や load コマンドを使って、MAT ファイルの中に関数ハンドルをセーブしたり、ユーザの MATLAB ワークスペースに読み込んだりできます。この例題は、関数ハンドル配列をファイル savefile にセーブしたり、再度、読み込んだりするものです。

```
fh_array = [@sin @cos @tan];
save savefile fh_array;
clear

load savefile
whos
  Name             Size          Bytes  Class
  fh_array        1x3             48  function_handle array
Grand total is 3 elements using 48 bytes
```

セーブとロードの間で行われる変更の影響

初期の MATLAB セッションで、ユーザがセーブした関数ハンドルをロードする場合、つぎの条件が、予期せぬ挙動の原因になります。

- 関数を定義している M- ファイルのいずれかが移動して、ハンドルの中に保存されているパス上に存在しない。
- セーブしたときの環境と異なる環境に関数ハンドルをロードします。たとえば、関数に対するソースが、存在しない、または、ハンドルがセーブされたシステムのデレクトリと異なるデレクトリに配置されているかのどちらかの場合です。
- save が、既に行われていたので、付加的なメソッドと共に関数を多重定義します。ユーザが今ロードした関数ハンドルは、新しいメソッドに関しては、情報がありません。

最初の 2 つのケースにおいて、関数ハンドルは正しくありません。これは、存在しているソースコードにマッピングされていないからです。ハンドルが正しくないけれども、MATLAB は、まだ、ロードはうまく行ないます。そして、ワーニングの表示もありません。しかし、ハンドルを計算しようとする試みは、エラーになります。

エラー状態の取り扱い

つぎのものは、関数ハンドルの使用に関連したエラー状態です。

存在しない関数へのハンドル

存在していない関数へのハンドルを作成しようとする場合、MATLAB は、ハンドルが feval で計算できるとき、エラーを出力します。MATLAB は、間違ったハンドルを割り当てることができ、func2str のような演算の中で使われますが、ランタイム演算の中でそれを使用しようとした場合、エラーになり、レポートされます。たとえば、

```
fhandle = @no_such_function;

func2str(fhandle)
ans =
no_such_function

feval(fhandle)
??? Error using ==> feval
Undefined function 'no_such_function'.
```

関数ハンドルコンストラクタの中に含まれるパス

関数の前に符号 @ を使うか、または、関数 str2func を使って、関数ハンドルを作成します。どちらの場合でも、簡単な関数名のみを使って、関数を設定します。関数名は、パス情報を含んでいません。つぎのものは、どちらも、ハンドルを関数 deblank にうまく作成したものです。

```
fhandle = @deblank;
fhandle = str2func('deblank');
```

つぎの例題は、パスを deblank.m に含ませ、そして、エラーを出力するものです。

```
fhandle = str2func(which('deblank'))
??? Error using ==> str2func
Invalid function name
    'matlabroot\toolbox\matlab\strfun\deblank.m'.
```

非スカラ関数ハンドルの計算

関数 feval は、関数ハンドルがスカラな場合のみ計算します。非スカラ関数ハンドルを共に feval をコールすると、エラーになります。

```
feval([@sin @cos], 5)
??? Error using ==> feval
Function_handle argument must be scalar.
```

履歴について - 関数名の計算

関数ハンドルを使って関数を計算することは、関数名を含む文字列を通して、関数を計算する以前の MATLAB 方式を置き換えました。たとえば、関数 humps を計算するつぎの 2 つのラインに関して、2 番目のものが最初のものに取り代わり、使用し易いものと考えられています。

```
feval('humps', 0.5674);       % uses a function name string

feval(@humps, 0.5674);        % uses a function handle
```

後位互換をサポートしているので、feval は、最初の引数として関数名文字列を受け入れ、文字列で付けられた関数を計算します。しかし、関数ハンドルは、21-3 ページの " 関数ハンドルを利用する利点 " の節に一覧表示されている、付加的な性能、信頼性、ソースファイルコントロールに関する利点をもたらせます。

22

MATLAB クラスとオブジェクト

クラスとオブジェクト：概要 22-3

MATLAB でのユーザクラス設計 22-9

演算子と関数の多重定義 22-20

例題 - Polynomial クラス 22-23

他のクラスの構築 22-33

例題 - 資産と資産サブクラス 22-36

例題 - ポートフォリオコンテナ 22-52

オブジェクトのセーブとロード 22-58

例題 - saveobj と loadobj の定義 22-59

オブジェクトの優先順位 22-63

MATLAB がメソッドを読み込む優先順位の決定法 . . . 22-65

この章は、MATLAB 内でのクラスの定義法に関する記述を行います。クラスとオブジェクトにより、MATLAB に新しいデータタイプと新しい演算をもちこむができます。変数の *クラス* は、変数の構造を記述し、変数に適用できる演算や関数の種類を示します。*オブジェクト* は、特殊なクラスの例です。*オブジェクト指向のプログラミング*と言う言葉は、クラスやオブジェクトの使用を強調するプログラムの書き方を記述します。

この章では、以下のトピックスと例題が記述されています。

- " クラスとオブジェクト： 概要 "
- "MATLAB でのユーザクラスの設計 "
- " 演算子と関数の多重定義 "
- " 例題 - Polynomial クラス "
- " 他のクラスの構築 "
- " 例題 - 資産と資産サブクラス "
- " 例題 - ポートフォリオコンテナ "
- " オブジェクトのセーブとロード "
- " 例題 - ポートフォリオに対する saveobj と loadobj の定義 "
- " オブジェクトの優先順位 "
- "MATLAB がメソッドを読み込む優先順位の決定法 "

クラスとオブジェクト： 概要

クラスに対して定義される特別な挙動をもつ新しいデータタイプとしてクラスと言うものを考えることができます。たとえば、多項式のクラスは、加算演算子 (+) を、多項式同志の加算を正しく行うものとして、再定義することができます。特別なクラスのオブジェクトと共に機能する演算をクラスの*メソッド* と言います。

クラスを単一エントリとして取り扱う新しいアイテムとして考えることができます。例題には、MATLAB(おそらく、MATLAB のラインやパッチオブジェクトで構成されている) グラフ上に表示され、Handle Graphics オブジェクトのようにプロパティをもつ矢印のオブジェクトがあります。

オブジェクトに対するデータストレージを与え、オブジェクトに適用する M- ファイルを含むクラスディレクトリを作成して、MATLAB 構造体を設定することにより、ユーザの MATLAB 環境の中にクラスを加えることができます。これらの M- ファイルは、クラスに対するメソッドになります。クラスディレクトリには、代数演算、サブスクリプト参照機能、そして、連結を含む種々の MATLAB の命令が、適応する方法を定義する方法も含んでいます。ユーザのクラスに対し、組み込みの演算がどのように機能するのかを再定義することは、演算子の*多重定義*として知られています。

オブジェクト指向のプログラミングの特徴

うまく設計されたクラスを使用することで、オブジェクト指向プログラミングは、コードの再利用を飛躍的に増やし、ユーザのプログラムを容易にし、かつ拡張します。クラスとオブジェクトを使ったプログラミングは、つぎの点で、通常の構造化プログラミングと異なります。

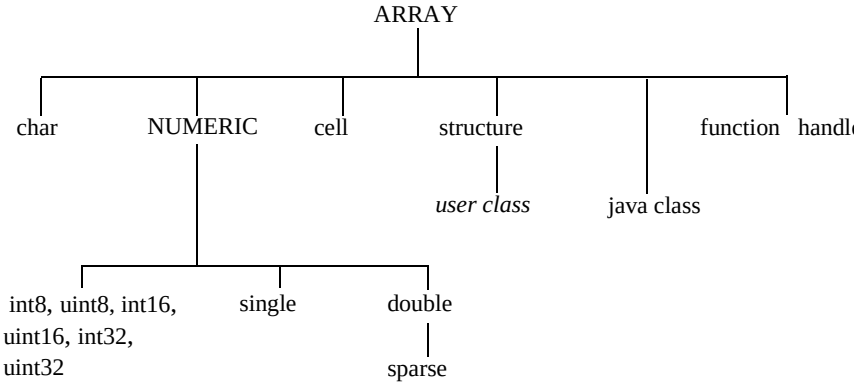
- **関数と演算子の多重定義**．ユーザは、既存の MATLAB 関数を書き換えるメソッドを作成することができます。引数として、ユーザ定義のオブジェクトをもつ関数をコールすると、MATLAB は、まず、オブジェクトのクラスに対して定義されているメソッドが存在するか否かをチェックします。存在すると、MATLAB は、通常の MATLAB 関数ではなく、そのメソッドを呼び込みます。
- **データとメソッドのまとめ**．オブジェクトプロパティは、コマンドラインから見えません。すなわち、クラスメソッドと共にのみ、それらに機能します。このことは、オブジェクトプロパティをオブジェクトのクラスに対して意図していない演算から保護できます。
- **継承**．ユーザは、子クラスは親クラスからデータフィールドやメソッドを継承すると親と子のクラスの階層構造を作成できます。子クラスは、一つの親からの継承 (*単一継承*) であるか、または、複数の親からの継承 (*多重継承*) をすることができます。継承は、一世代、または複数の世代に渡っています。継承は、共通

の親関数を共有でき、すべての子クラス間で共通の挙動をもつことができます。

- **集約 (Aggregation)**. あるオブジェクトが他のオブジェクトを含むような*集約*を使うクラスを作成することができます。これは、あるオブジェクトのタイプが他のオブジェクトのタイプの一部である場合に適切です。たとえば、貯蓄アカウントオブジェクトは、ファイナンシャルポートフォリオオブジェクトの一部になっています。

MATLAB データクラスの階層構造

すべての MATLAB データタイプは、オブジェクト指向のプログラミングの中のクラスとして設計されています。つぎの図は、MATLAB の中の 14 個の基本的なデータタイプを示しています。クラスの階層を拡張することにより、新しいデータタイプを MATLAB に加えることができます。



ダイアグラムは、構造体クラスから継承している*ユーザクラス* を示しています。これは、ユーザがユーザ自身のクラスを挿入できるクラス階層の中の点であるので、ユーザが作成できるすべてのクラスは、構造体をベースにしています (MATLAB のデータタイプに関する詳細は、データタイプの節を参照してください)。

オブジェクトの作成

クラスコンストラクタを呼び、それに適切な入力引数を渡すことで、オブジェクトを作成できます。MATLAB では、コンストラクタは、クラス名と同じ名前をもっています。たとえば、つぎのステートメント

```
p = polynom([1 0 -2 -5]);
```

は、クラスpolynomに属するpと名付けられたオブジェクトを作成します。polynomオブジェクトを作成すると、polynom クラスに対して定義されるメソッドを使うオブジェクトに適用できます。polynom クラスの詳細は、22-23 ページの " 例題 - Polynomial クラス " を参照してください。

オブジェクトにメソッドを適用

クラスメソッドは、一つのオブジェクトを入力引数の一つとして扱う M- ファイル関数です。特別なクラスに対するメソッドは、そのクラスに対応するクラスディレクトリ (*@class_name* ディレクトリ) に配置してください。これは、MATLAB がクラスメソッドを探索する最初の場所です。

オブジェクト上にメソッドを読み込む構文は、関数の呼び込みと同じです。一般に、つぎの型をしています。

```
[out1,out2,...] = method_name(object,arg1,arg2, ...);
```

たとえば、polynom と呼ばれるユーザ定義のクラスは、クラスに対して定義された char メソッドをもっているとします。このメソッドは、polynom オブジェクトをキャラクタ文字列に変換し、文字列を出力するものです。つぎのステートメントは、polynom オブジェクト p に char メソッドを適用します。

```
s = char(p);
```

関数 class を使うことで、戻り値 s がキャラクタ文字列であることがわかります。

```
class(s)

ans =

    char

s

s =

    x^3-2*x-5
```

コマンド methods を使って、あるクラスに対して、定義されているすべてのメソッドの一覧を呼び込むことができます。

プライベートメソッド

プライベートメソッドは、それらのクラスの他のメソッドでのみ読み込むことがで

きるものです。*@class_name* ディレクトリの private サブディレクトリ中に関連したM- ファイルを設定することで、プライベートメソッドを定義できます。たとえば、

```
@class_name/private/update_obj.m
```

は、update_obj メソッドが、*class_name* クラスの中でのみに存在します。これは、update_obj が、*@class_name* ディレクトリで定義される任意のメソッドにより呼び出すことができることを意味します。しかし、これは、親のメソッドを含み、クラスのディレクトリ以外にあるメソッドやMATLABコマンドラインのどちらからも呼び出すことはできません。

プライベートメソッドとプライベート関数は、プライベートメソッドでは、その入力引数の一つとしてオブジェクトをもち、プライベート関数では、そのようなオブジェクトをもたない点が異なります。ユーザは、プライベート関数を、次節で紹介する補助関数として使うことができます。

補助関数

あるクラスの設計において、直接、オブジェクトに適用するのではなく、クラスに対して補助するような関数が必要であることがわかります。これらの関数を、*補助関数*と言います。補助関数は、クラスメソッドファイルや、または、プライベート関数の中のサブ関数と言われます。特別な関数のどのバージョンが呼び出されるかが決定する場合、MATLAB は上で示した図の一覧に従って探索します。MATLAB がどの関数やメソッドを呼び出すかの順番についての詳細は、22-65 ページの "MATLAB がメソッドを読み込む優先順位の決定法 " を参照してください。

クラスメソッドのデバッグ

MATLAB デバッギングコマンドをM- ファイルに使用するのと同様な方法を使って、オブジェクトメソッドに適用することができます。異なる点は、dbstop を使うつぎの例題に示すように、コマンド呼び出しにおいて、メソッド名の前に、クラスのディレクトリ名を含ませることが必要なことです。

```
dbstop @polynom/char
```

クラスメソッドのデバッグ中に、継承されたメソッド、プライベートメソッド、プライベート関数を含むクラスに対して定義されたすべてのメソッドにアクセスします。

クラス定義の変更

クラス定義を変更する場合、たとえば、数値やあるクラスの中のフィールドを変更する場合、MATLAB セッションに変更を伝えるために、つぎのコマンドを実行します。

```
clear classes
```

このコマンドは、ワークスペースからすべてのオブジェクトをクリアします。より詳細は、clear コマンドを参照してください。

クラスディレクトリの設定

あるクラスに対してメソッドを設定する M- ファイルは、クラスディレクトリとして識別されたディレクトリに集められます。ディレクトリ名は、クラス名の前にキャラクタ @ を付けて定義されます。たとえば、この章の中で使用する例題の一つは、単一変数内に多項式をもつクラスです。クラスの名前とクラスコンストラクタの名前は、polynom です。多項式クラスを定義する M- ファイルは、名前 @polynom をもつディレクトリに配置されます。

クラスディレクトリは、MATLAB サーチパス上にあるディレクトリのサブディレクトリになりますが、パスの上にはありません。たとえば、新しい @polynom ディレクトリは、MATLAB のワーキングディレクトリか、または、サーチパスに以前に設定されたユーザ自身のパーソナルディレクトリのサブディレクトリです。

クラスディレクトリを MATLAB パスに付加

クラスディレクトリを作成した後、MATLAB がクラスソースファイルの位置になるように MATLAB パスをアップデートします。クラスディレクトリは、直接 MATLAB パス上にあるものではありません。代わりに、親ディレクトリを MATLAB パスに付加します。たとえば、@polynom クラスディレクトリを

```
c:\my_classes\@polynom
```

に配置する場合、addpath コマンドを使って、クラスディレクトリを MATLAB パスに付加できます。

```
addpath c:\my_classes;
```

他のクラスと同じ名前をもつクラスディレクトリを作成する場合、MATLAB は、クラスメソッドを配置するときに、二つのクラスディレクトリを同じ一つのディレクトリとして取り扱います。詳細は、22-65 ページの "MATLAB がメソッドを読み込む優先順位の決定法 " を参照してください。

データ構造

新しいクラス設計の最初のステップの一つは、クラスにより使われるデータ構造の選択です。オブジェクトは MATLAB 構造体の中に格納されます。構造体のフィールドおよびフィールドに機能する演算の詳細は、クラスに対するメソッドの中にのみ

見られます。適切なデータ構造体の設計は、コードのパフォーマンスに影響します。

C++ と Java プログラミングに対するヒント

他のオブジェクト指向言語、たとえば、C++ や Java のような言語でのプログラミングに慣れている場合、つぎの点が MATLAB プログラミング言語と異なっていることがわかります。

- MATLAB では、メソッドのディスパッチは、C++ や Java のように構文ベースではありません。引数リストが、同じ優先順位のオブジェクトを含んでいる場合、MATLAB は、呼び出すメソッドの順番を、最も左にあるオブジェクトからにします。
- MATLAB では、デストラクタメソッドが異なります。ワークスペースからあるオブジェクトを取り除くためには、関数 clear を使います。
- MATLAB データタイプの生成は、コンパイル時ではなく、むしろ実行時に生じます。関数 class を呼び出すことにより、あるクラスに属するものとしてオブジェクトを割り当てます。
- MATLAB 内の継承を使う場合、継承関係は、親オブジェクトを作成し、その後に、関数 class を呼び出して、子クラスの中で作成されます。継承関係に対するコンストラクタの記述に関する詳細は、22-33 ページの " 他のクラスの構築 " を参照してください。
- MATLAB の中で継承を使用する場合、子オブジェクトは、親クラスの名前をもつプロパティ内の親オブジェクトを継承します。
- MATLAB では、リファレンスによる変数の受け渡しはありません。オブジェクトを更新するメソッドを記述する場合、更新オブジェクトを戻し、代入ステートメントを使わなければなりません。たとえば、set メソッドの呼び出しは、オブジェクト A の name フィールドを更新し、更新したオブジェクトを出力します。

  ```
  A = set(A,'name','John Smith');
  ```

- MATLAB では、抽象クラスに相当するものがありません。
- MATLAB では、C++ スコープ演算子に相当するものがありません。
- MATLAB では、仮想継承または仮想基底クラスはありません。
- MATLAB では、C++ テンプレートに相当するものがありません。

MATLAB でのユーザクラスの設計

この節は、クラス設計法の議論とあるクラス内に当然含まれなければならない基本的なメソッドの集合を記述します。

MATLAB での基準となるクラス

MATLAB クラスを設計する場合、MATLAB 環境内に矛盾のないかつ論理的な方法で、クラスが挙動する標準のメソッド群を含ませる必要があります。定義するクラスの性質に依存しているので、メソッドすべてが必要でない場合もあり、クラスの設計ゴールに達するために、いくつかの他のメソッドを含ませる場合もあります。

つぎの表は、MATLAB のクラス内に含まれる基本的なメソッドです。

クラスメソッド	詳細
class constructor	クラスのオブジェクトの作成
display	MATLAB がオブジェクトの内容を表示する場合 (式表現が、最後にコロン記号を付けないで、呼び込まれた場合) に、呼ばれます。
set と get	クラスプロパティにアクセス
subsref と subsasgn	ユーザオブジェクトに対して、インデックス付きリファレンスと代入が可能
end	あるオブジェクトを使って、インデックス表現の中で end 構文をサポート (たとえば、A(1:end))
subsindex	インデックス表現内のオブジェクトの使用をサポート
converters like double and char	あるオブジェクトを MATLAB データタイプに変換するメソッド

つぎの節は、メソッドの個々のタイプの実現について、この章で使われる例題へのリファレンスとして、議論します。

クラスコンストラクタメソッド

特別なクラスに対する @ ディレクトリは、そのクラスに対するコンストラクタ とする M- ファイルを含んでいます。コンストラクタの名前は、クラスの名前を定義するディレクトリ (接頭記号 @ や .m の拡張子を除いて) の名前と同じです。コンストラクタは、データ構造を初期化したり、クラスオブジェクトを割り当ててオブジェクトを作成します。

コンストラクタを記述するためのガイドライン

クラスコンストラクタは、MATLAB 環境の中で、オブジェクトが正しく機能するような関数です。一般的に、クラスコンストラクタは、入力引数の可能な 3 つの組み合わせを取り扱います。

- 入力引数なし
- 入力引数と同じクラスのオブジェクト
- クラス (一般的には、ある種のデータ) のオブジェクトを作成するために使われる入力引数

入力引数なし . 入力引数がない場合、コンストラクタはデフォルトのオブジェクトを作成します。入力がないために、オブジェクトを作成するためのデータがありません。それで、オブジェクトのデータ構造体を空、または、デフォルト値で、単に初期化し、オブジェクトを示す class 関数を呼び込み、出力引数として、オブジェクトを戻します。この構文に対するサポートは、つぎの 2 つの理由から必要になります。

- オブジェクトをワークスペースにロードする場合、load 関数は、引数なしでクラスコンストラクタを呼びます。
- オブジェクトの配列を作成する場合、MATLAB は、クラスコンストラクタを呼び配列にオブジェクトを付加します。

オブジェクト入力引数 . 引数内の最初の入力引数が同じクラスのオブジェクトの場合、コンストラクタは、単にオブジェクトを戻します。isa 関数を使って、引数がクラスのメンバか否かを決定します。このコンストラクタ構文を使ったメソッドの例題は、22-28 ページの "+ 演算子の多重定義 " を参照してください。

データ入力引数 . 入力引数が存在し、同じクラスのオブジェクトでない場合、コンストラクタは、入力データを使って、オブジェクトを作成します。もちろん、任意の関数内と同じく、ユーザのコンストラクタ関数内でチェックされた適切な引数になります。一般的なアプローチは、varargin 入力引数を使い、プログラムフローをコントロールするため switch ステートメントを使います。これは、3 つの

場合、すなわち、入力なし、オブジェクト入力、オブジェクトを作成するために使われるデータ入力をまとめた簡単な方法です。

オブジェクトのデータ構造体に値を割り当て、関数 class を呼んでオブジェクトと (instantiate) し、出力引数としてオブジェクトを戻すことが、コンストラクタのこの部分にあります。必要ならば、関数 superiorto と inferiorto を使って、オブジェクト階層構造内に、オブジェクトを配置します。

コンストラクタの中で class 関数を使用

コンストラクタメソッドの中で、オブジェクト構造体を特別なクラスに関連付けるために関数 class を使います。これは、関数 class と isa を使ってしかアクセスできない内部クラスタグを使って行います。たとえば、この関数 class を呼び出しは、タイプ polynom の中のオブジェクト p を識別します。

```
p = class(p,'polynom');
```

コンストラクタメソッドの例題

コンストラクタメソッドの例題は、以下を参照してください。

- 22-23 ページの "Polynom コンストラクタメソッド "
- 22-37 ページの " 資産コンストラクタメソッド "
- 22-44 ページの " 株式クラスコンストラクタ "
- 22-53 ページの " ポートフォリオコンストラクタメソッド "

クラスディレクトリ外のオブジェクトの識別

コンストラクタメソッドの中で使われる関数 class と isa は、クラスディレクトリ外でも使えます。つぎの表現

```
isa(a,'class_name');
```

は、a が指定したクラスのオブジェクトかどうかをチェックします。たとえば、つぎの表現は、p が polynom オブジェクトの場合、それぞれ真になります。

```
isa(pi,'double');
isa('hello','char');
isa(p,'polynom');
```

クラスディレクトリ外では、関数 class は一つの引数のみを使います (class が複数の引数をもつのは、コンストラクタの中だけです)。

表現

```
class(a);
```

は、a のクラス名を含んだ文字列を出力します。たとえば、

```
class(pi),
class('hello'),
class(p)
```

は、つぎの出力を行います。

```
'double',
'char',
'polynom'
```

MATLAB のワークスペース内のオブジェクトを見るために、whos コマンドを使用します。

```
whos
```

```
  Name      Size          Bytes  Class
  p         1x1           156 polynom object
```

display メソッド

MATLAB は、オブジェクトがセミコロンで終わらないステートメントの場合は、常に、display と名付けたメソッドを呼びます。たとえば、double である変数 a の作成は、つぎのようにすると、double に対して、MATLAB の display メソッドを呼びます。

```
a = 5

a =

     5
```

MATLAB は、ユーザのクラスからオブジェクト参照した場合、コマンドライン上で値を表示できるように、a と名付けた display メソッドを定義します。多くのクラスの中で、display は、変数名を単に出力し、その後、MATLAB は、文字列である出力を表示するために、char 変換メソッドを使って、内容または変数値を出力します。オブジェクトのデータをキャラクタ文字列に変換するために char メソッドを定義してください。

display メソッドの例題

display メソッドの例題は、つぎの節を参照してください。

- 22-27 ページの "Polynom display メソッド "
- 22-42 ページの " 資産 display メソッド "
- 22-50 ページの " 株式 display メソッド "
- 22-54 ページの " ポートフォリオ display メソッド "

オブジェクトデータのアクセス

ユーザのクラスに対して、オブジェクトデータへのアクセスのためのメソッドを記述する必要があります。Accessor メソッドは種々のアプローチを使いますが、オブジェクトデータを変更するすべてのメソッドは、入力引数として一つのオブジェクトを受け入れ、変更したデータをもつ新しいオブジェクトを出力します。これは、MATLAB がリファレンス (たとえば、ポインタ) により渡される引数をサポートしていないから、必要なことです。関数は、オブジェクトのプライベート、テンポラリコピーのみを変更できます。そのため、存在しているオブジェクトを変更するために、新しいオブジェクトを作成し、古いオブジェクトと置き換える必要があります。

つぎの節は、set, get, subsasgn, subsref メソッドに対する実現手法に対する詳細を示します。

set メソッドと get メソッド

set メソッドと get メソッドは、あるケースで、オブジェクトデータにアクセスする便利な方法です。たとえば、(既存の MATLAB の line オブジェクトと patch オブジェクトから構成されている) グラフ上に表示した矢印のオブジェクトを定義するクラスを作成することを考えましょう。

矛盾しないインタフェースを作成するため、組み込みのグラフィックスオブジェクト上に適用される MATLAB の set 関数と get 関数を使って、矢印オブジェクト上に適用される set メソッドと get メソッドを定義できます。set と get は、それらが実行する演算と同じものですが、ユーザはオブジェクトの内部と切り離されます。

set メソッドと get メソッドの例題

set メソッドと get メソッドの例題は、つぎの節を参照してください。

- 22-38 ページの " 資産 get メソッド " と 22-39 ページの " 資産 set メソッド "
- 22-45 ページの " 株式 get メソッド " と 22-46 ページの " 株式 set メソッド "

プロパティ名メソッド

一般的な set メソッドを変更して、個々のプロパティの代入を取り扱うメソッドを記述することができます。メソッドは、プロパティ名と同じ名前をもっています。

たとえば、雇用データを表わすオブジェクトを作成するクラスを定義する場合、雇用オブジェクトの中に salary と呼ばれるフィールドをもたせます。そして、雇用オブジェクトと値を入力引数とし、設定し値の組をもつオブジェクトを出力させる salary.m と言われるメソッドを定義します。

subsref と subsasgn を使用したインデックス付きリファレンス

ユーザのクラスは、MATLABの中で新しいデータタイプを実現するものです。MATLABの組み込みのデータタイプと同じように、インデックス付きリファレンスを通してオブジェクトデータをアクセスできることは有益なことです。たとえば、A がクラス double の配列の場合、A(i) は、A の i- 番目の要素を戻します。

クラスの設計者として、あるオブジェクトに対するインデックスリファレンスが意味するものを決めることができます。たとえば、多項式オブジェクトを作成し、それらのオブジェクトは、多項式の係数を含んでいるというクラスを定義することを考えましょう。

多項式オブジェクトへのインデックス付きリファレンス

```
p(3)
```

は、x^3 の係数の値、または、$x = 3$ での多項式の値、または、設計に依存して少し異なったもののいずれかを戻します。

2 つのクラスメソッド、subsref と subsasgn を作成することで、特別なクラスに対するインデックスの挙動を定義することができます。MATLAB は、サブスクリプトを使ったリファレンス、または、代入がクラスから一つのオブジェクトの上に適用されると、これらのメソッドが呼ばれます。あるクラスに対して、これらのメソッドを定義しない場合、インデックス機能は、このクラスのオブジェクトに対して未定義となります。

一般に、オブジェクトのインデックス機能に対するルールは、構造体配列にインデックス機能を適用する場合のルールと同じものになります。詳細は、20-1 ページの " 構造体とセル配列 " を参照してください。

サブスクリプトを使ったリファレンスの取り扱い

代入ステートメントの右辺側にあるオブジェクトをサブスクリプト、または、フィールド識別子と共に使うことは、*サブスクリプトを使ったリファレンス (subscripted reference)* として知られています。MATLAB は、このような場合、

subsref と名付けたメソッドを呼びます。オブジェクトのサブスクリプト化されたリファレンスは、3 つの型、配列インデックス、セル配列インデックス、構造体フィールド名からなります。

```
A(I)
A{I}
A.field
```

と名付けたメソッドを呼びます。オブジェクトのサブスクリプト化されたリファレンスは、3 つの型、配列インデックス、セル配列インデックス、構造体フィールド名からなります。

```
B = subsref(A,S)
```

最初の引数は、参照されるオブジェクトです。二番目の引数 S は、2 つのフィールドをもつ構造体配列です。

- S.type は、サブスクリプトタイプを設定する '()', '{}', '.' いずれかを含む文字列です。括弧は数値配列、中括弧はセル配列、点は構造体配列を表わします。
- S.subs は、実際のサブスクリプトを含んだセル配列、または、文字列です。サブスクリプトのように使用するコロンは、文字列 ':' 同じように転送されます。

たとえば、つぎの表現

```
A(1:2,:)
```

は、MATLAB にセル subsref(A,S) を呼び出します。ここで、S はつぎの要素

```
S.type = '()'
S.subs = {1:2,':'}
```

をもつ 1 行 1 列の構造体です。同様に、つぎの表現

```
A{1:2}
```

は、つぎのものを使います。

```
S.type ='{}'
S.subs = {1:2}
```

つぎの表現

```
A.field
```

は、subsref(A,S) を呼びます。ここで、

```
S.type = '.'
S.subs = 'field'
```

この簡単な呼び出しを組み合わせて、より複雑なサブスクリプトを使った表現に適用できます。このような場合、length(S) は、サブスクリプトを使ったレベル数です。たとえば、

```
A(1,2).name(3:4)
```

は、subsref(A,S) を呼びます。ここで、S は、つぎの値をもつ 3 行 1 列の構造体配列です。

```
S(1).type = '()'    S(2).type = '.'      S(3).type = '()'
S(1).subs = '{1,2}' S(2).subs = 'name'   S(3).subs = '{3:4}'
```

subsref の書き方

subsref メソッドは、MATLAB によって渡されるサブスクリプトを使った表現を解釈します。通常のアプローチは、switch ステートメントを使って、使用するインデックス機能のタイプを決定し、実際のインデックスを取得します。つぎの 3 つの分割されたコードは、入力引数の解釈法を示しています。各々の場合、関数は、値 B を出力します。

配列インデックスに対して、

```
switch S.type
case '()'
    B = A(S.subs{:});
end
```

セル配列に対して、

```
switch S.type
case '{}'
    B = A(S.subs{:}); % A is a cell array
end
```

構造体配列に対して、

```
switch S.type
case '.'
    switch S.subs
    case 'field1'
         B = A.field1;
```

```
        case 'field2'
             B = A.field2;
        end
    end
```

subsref メソッドの例題

subsref メソッドの例題は、つぎの節を参照してください。

- 22-27 ページの "Polynom subsref メソッド "
- 22-40 ページの " 資産 subsref メソッド "
- 22-47 ページの " 株式 subsref メソッド "
- 22-62 ページの " ポートフォリオ subsref メソッド "

サブスクリプトを使った代入

代入ステートメントの左辺に存在するオブジェクトと共に、サブスクリプト、または、フィールド識別子を使用することは、*サブスクリプトを使った代入 (subscripted assignment)* として知られています。MATLAB は、このような場面で、subsasgn と呼ばれるメソッドを呼び込みます。オブジェクトのサブスクリプト化された代入は、3 つの型、配列インデックス、セル配列インデックス、構造体フィールド名をもっています。

```
A(I) = B
A{I} = B
A.field = B
```

これらの各々は、つぎの型

```
A = subsasgn(A,S,B)
```

の subsasgn への呼び出しの結果になります。最初の引数 A は、参照されるオブジェクトで、2 番目の引数 S は、subsref と共に使われるものと同じフィールドをもって、3 番目の引数 B は、新しい値です。

subsasgn メソッドの例題

subsasgn メソッドの例題は、つぎの節を参照してください。

- 22-41 ページの " 資産 subsasgn メソッド "
- 22-49 ページの " 株式 subsasgn メソッド "

オブジェクトに対して、end インデックスの定義

オブジェクトのインデックス表現の中で end を使用する場合、MATLAB は、オブジェクトの end クラスメソッドを呼びます。ユーザのクラスのオブジェクトを含むインデックスを使った表現の中で end を使いたい場合、ユーザのクラスに対する end メソッドを定義しなければなりません。

end メソッドは、つぎのような構文で使います。

```
end(a,k,n)
```

ここで、a はユーザのオブジェクト、k は end 構文が使われる表現式の中のインデックス、n は表現式の中のインデックスの総数です。

たとえば、つぎの表現を考えましょう。

```
A(end-1,:)
```

MATLAB は、つぎの引数

```
end(A,1,2)
```

を使ってオブジェクト A に対して定義された end メソッドを呼びます。ここで end ステートメントは、最初のインデックス要素内で生じ、その位置を他の 2 つのインデックス要素で示します。end に対するクラスメソッドは、最初の次元の最後の要素に対するインデックスを戻します。ユーザのクラスに対して end メソッドを実行する場合、オブジェクトに対して適切な値を戻すことを保証する必要があります。

あるオブジェクトを他のオブジェクトを使ってインデックス化

MATLAB は、オブジェクトをインデックスとしてカウントする場合、オブジェクトに対して定義された subsindex メソッドを読み込みます。たとえば、オブジェクト a があり、このオブジェクトを使って、他のオブジェクト b の中でインデックス化を行なうことを考えます。

```
c = b(a);
```

subsindex メソッドは、つぎのサンプルコードに示すように、あるインデックスとして使うことができるようにオブジェクトをdouble形式に変換するようなことを行います。

```
function d = subsindex(a)
%SUBSINDEX
% インデックス表現の中で、一つのインデックスとして、
% 使われる倍精度フォーマットにオブジェクトを変換
```

```
d = double(a);
```

subsindex 値は、0 ベースで、1 ベースではありません。

Converter メソッド

converter メソッドは、たとえば、char、または、double のように他のクラスと同じ名前をもつクラスメソッドです。converter メソッドは、入力として一つのクラスのオブジェクトを受け入れ、他のクラスのオブジェクトを出力します。メソッドは、つぎのことを可能にします。

- 他のクラスに対して定義されたメソッドを使用
- ミックスしたクラスタイプのオブジェクトを含んだ表現が適切に実行していることを確認

関数 converter は、つぎの構文で呼ばれます。

```
b = class_name(a)
```

ここで、a は、*class_name* とは別のクラスのオブジェクトです。この場合、MATLAB は、オブジェクト a に対して、クラスディレクトリの中で *class_name* と呼ばれるメソッドを探索します。入力オブジェクトが既にタイプ class_name になっていると、MATLAB は、コンストラクタを呼び込み、入力引数そのものを戻します。

Converter メソッドの例題

converter メソッドの例題は、つぎの節を参照してください。

- 22-24 ページの "Polynom から double への変換 "
- 22-25 ページの "Polynom から Char への変換 "

演算子と関数の多重定義

多くの場合、引数がオブジェクトのケースに対して、MATLAB 演算子や関数の挙動を変更したい場合があります。これは、基準になる関数を*多重定義* することにより可能になります。多重定義は、入力引数の種々のタイプ、種々の入力数を取り扱う機能をもたせ、最も高い優先順位をもつものに対しても演算を適切に行うことができます。オブジェクトの優先順位に関する情報は、22-63 ページの " オブジェクトの優先順位 " を参照してください。

演算子の多重定義

各々の組み込み MATLAB 演算子は、関連した関数名 (たとえば、+ 演算子が、関連した関数 plus.m) をもっています。ユーザは、クラスディレクトリの中で適切な名前をもつ M- ファイルを作成することにより、任意の演算を多重定義することができます。たとえば、p または q のどちらかがタイプ *class_name* のオブジェクトならば、つぎの式

```
p + q
```

は、関数 @*class_name*/plus.m が存在しているなら、それを読み込みます。p と q が、共に異なるクラスのオブジェクトの場合、MATLAB は、優先順位則を適用して、使用するメソッドを決定します。

多重定義演算子の例題

多重定義演算子の例題に対しては、つぎの節を参照してください。

- 22-28 ページの "+ 演算子の多重定義 "
- 22-29 ページの "- 演算子の多重定義 "
- 22-29 ページの "* 演算子の多重定義 "

つぎの表は、MATLAB のほとんどの演算子に関する関数名の一覧です。

演算	M- ファイル	詳細
a + b	plus(a,b)	バイナリ加算
a - b	minus(a,b)	バイナリ減算
-a	uminus(a)	単項マイナス
+a	uplus(a)	単項プラス
a.*b	times(a,b)	要素単位の乗算
a*b	mtimes(a,b)	行列乗算
a./b	rdivide(a,b)	要素単位の右除算
a.\b	ldivide(a,b)	要素単位の左除算
a/b	mrdivide(a,b)	行列の右除算
a\b	mldivide(a,b)	行列の左除算
a.^b	power(a,b)	要素単位のベキ乗
a^b	mpower(a,b)	行列のベキ乗
a < b	lt(a,b)	より小さい
a > b	gt(a,b)	より大きい
a <= b	le(a,b)	より小さいか等しい
a >= b	ge(a,b)	より大きいか等しい
a ~= b	ne(a,b)	等しくない
a == b	eq(a,b)	等しい
a & b	and(a,b)	論理和
a | b	or(a,b)	論理積
~a	not(a)	論理偽

演算	M- ファイル	詳細
a:d:b a:b	colon(a,d,b) colon(a,b)	コロン演算子
a'	ctranspose(a)	複素共役転置
a.'	transpose(a)	行列転置
command window output	display(a)	表示メソッド
[a b]	horzcat(a,b,...)	水平連結
[a; b]	vertcat(a,b,...)	垂直連結
a(s1,s2,...sn)	subsref(a,s)	サブスクリプトを使った参照
a(s1,...,sn) = b	subsasgn(a,s,b)	サブスクリプトを使った代入
b(a)	subsindex(a)	サブスクリプトを使ったインデックス

関数の多重定義

クラスディレクトリに同じ名前の関数を作ることにより任意の関数を多重定義できます。関数がオブジェクト上で読み込まれるとき、MATLAB はサーチパス上にある他の位置の前にクラスディレクトリ内で関数を探します。たとえば、オブジェクトのあるクラスに対して、関数 plot を多重定義することは、適切なクラスディレクトリの中の plot.m のユーザバージョンに単に置き換えることです。

関数の多重定義の例題

関数の多重定義の例題は、つぎの節を参照してください。

- 22-30 ページの "Polynom クラスに対する関数の多重定義 "
- 22-54 ページの " ポートフォリオ pie3 メソッド "

例題 - Polynomial クラス

この例題は、polynom と名付けた新しいクラスを定義することにより、多項式に対する MATLAB のデータタイプを実現できます。クラスの定義は、データストレージに対するある構造体を設定し、polynom オブジェクト上に演算するメソッドのディレクトリ (@polynom) を定義します。

Polynom データ構造体

polynom クラスは、変数を降べきの順に、並べた係数の行ベクトルで多項式を表わしています。そのために、polynom オブジェクト p は、係数を含む単一フィールド p.cc をもつ構造体です。このフィールドは、@polynom ディレクトリの中のメソッドのみにアクセス可能です。

Polynom メソッド

MATLAB 環境の中で機能するあるクラスを作成し、多項式のデータタイプに対して有効な機能を設定するため、polynom クラスは、つぎのメソッドを実現します。

- コンストラクタメソッド polynom.m
- polynom を double に変換
- polynom を char に変換
- display メソッド
- subsref メソッド
- 多重定義された +, -, * 演算子
- 多重定義された roots, polyval, plot, diff 関数

Polynom コンストラクタメソッド

つぎに、polynom クラスコンストラクタ @polynom/polynom.m を示します。

```
function p = polynom(a)
%POLYNOM 多項式クラスのコンストラクタ
% p = POLYNOM(v) は、x の降べきの順に並べた係数を含んで、
% ベクトル v から多項式オブジェクトを作成
if nargin == 0
   p.c = [];
   p = class(p,'polynom');
elseif isa(a,'polynom')
```

```
    p = a;
else
    p.c = a(:).';
    p = class(p,'polynom');
end
```

呼び出し構文のコンストラクタ

3 つの異なる引数の内の一つを使って、polynom コンストラクタメソッドを呼び出すことができます。

- 入力引数なし - 引数なしでコンストラクタ関数を呼び込む場合、空フィールドをもった polynom オブジェクトを出力します。
- 入力引数がオブジェクト - 既に polynom オブジェクトであるものを入力引数としてコンストラクタ関数に呼び込む場合、MATLAB は入力引数を戻します。関数 isa("is a" を意味します) は、状況をチェックします。
- 入力引数が係数ベクトル - 入力引数が、polynom オブジェクトでない変数の場合、それを行ベクトルに整形し、オブジェクトの構造体の .c フィールドに代入します。関数 class は、polynom オブジェクトを作成し、その後、コンストラクタに戻します。

polynom コンストラクタの使用例を示します。

```
p = polynom([1 0 -2 -5])
```

これは、設定された係数をもつ多項式を作成します。

Polynom クラスに対する converter メソッド

Converter メソッドは、あるクラスのオブジェクトを他のクラスのオブジェクトに変換します。MATLAB に用意されている重要な二つの converter メソッドは、double と char です。double への変換は、MATLAB の従来までの行列を作成します。しかし、いくつかのクラスに対しては、これは適切でない場合もあります。char への変換は、プリント出力の作成に有効なものです。

Polynom から double への変換

polynom クラスに対する double converter メソッドは、非常に簡単な M- ファイル @polynom/double.m です。これは、単に係数ベクトルを戻すだけです。

```
function c = double(p)
% POLYNOM/DOUBLE polynom オブジェクトを係数ベクトルに変換
% c = DOUBLE(p) は、多項式オブジェクトを、
% x の降べきの順に並べた係数を含むベクトル c に変換
c = p.c;
```

オブジェクト p を作成

```
p = polynom([1 0 -2 -5])
```

ステートメント

```
double(p)
```

は、つぎの出力を行います。

```
ans =
    1     0    -2    -5
```

Polynom から Char への変換

Char への変換は、独立変数 x のベキ乗表現を含んだキャラクタ文字列を作成するので、キーのメソッドになります。そのために、x を一度設定すると、戻される文字列は、文法的に正しい MATLAB 表現になり、後で、計算に使用できます。

つぎに、@polynom/char.m を示します。

```
function s = char(p)
% POLYNOM/CHAR
% CHAR(p) は、p.c の文字列表現です。
if all(p.c == 0)
    s = '0';
else
    d = length(p.c) - 1;
    s = [];
    for a = p.c;
        if a ~= 0;
            if ~isempty(s)
                if a > 0
                    s = [s ' + '];
                else
                    s = [s ' - '];
                    a = -a;
                end
```

```
                end
                if a ~= 1 | d == 0
                    s = [s num2str(a)];
                    if d > 0
                        s = [s '*'];
                    end
                end
                if d >= 2
                    s = [s 'x^' int2str(d)];
                elseif d == 1
                    s = [s 'x'];
                end
            end
            d = d - 1;
        end
    end
```

出力の計算

つぎの polynom オブジェクト p を作成します。

```
p = polynom([1 0 -2 -5]);
```

そして、p 上に char メソッドを適用します。

```
char(p)
```

MATLAB は、つぎの結果を出力します。

```
ans =
    x^3 - 2*x - 5
```

char により出力される値は、x に対するスカラ値を設定すると、eval に渡される文字列になります。たとえば、

```
x = 3;
eval(char(p))

ans =
    16
```

多項式を計算するより良いメソッドについては、22-27ページの"Polynom subsref メソッド" を参照してください。

Polynom display メソッド

つぎに、@polynom/display.m を説明します。このメソッドは、多項式の文字列表現を char メソッドに依存し、スクリーン上に行います。このメソッドは、標準の MATLAB 出力と同じ出力を生みます。このメソッドは、変数名に等号記号が続いて表示され、その後にブランク、そして、値を示す新しいラインが続きます。

```
function display(p)
% POLYNOM/DISPLAY Command window display of a polynom
disp(' ');
disp([inputname(1),' = '])
disp(' ');
disp(['   ' char(p)])
disp(' ');
```

ステートメント

```
p = polynom([1 0 -2 -5])
```

は、polynom オブジェクトを作成します。ステートメントがセミコロン付きで終わっていないので、結果の出力は、つぎのようになります。

```
p =

    x^3 - 2*x - 5
```

Polynom subsref メソッド

Polynom クラスの設計は、polynom オブジェクトへのサブスクリプトを使ったリファレンスが、サブスクリプトに等しい独立変数値を使って多項式を計算するように設定されています。polynom オブジェクト p に対して、

```
p = polynom([1 0 -2 -5]);
```

は、つぎのサブスクリプトを使った表現が、x = 3 と x = 4 での多項式の値を、つぎのように戻します。

```
p([3 4])

ans =

    16    51
```

subsref の実現の詳細

この実現は、polynom クラスの中に既に定義されている char メソッドを利用して、表現式を作成し、そして、その値を計算します。

```
function b = subsref(a,s)
% SUBSREF
switch s.type
case '()'
    ind = s.subs{:};
    for i = 1:length(ind)
        b(i) = eval(strrep(char(a),'x',num2str(ind(i))));
    end
otherwise
   error('Specify value for x as p(x)')
end
```

多項式表現が、char メソッドにより作成されると、strrep 関数は、キャラクタ x に対する値に渡されたものをスワップするために使います。eval 関数は、表現式を計算し、出力引数に値を戻します。

代数演算子の多重定義

いくつかの代数演算は、多項式に対して有効で、polynom クラスに対して実現されるものです。代数演算子を多重定義するときは、演算を適用したいデータタイプを常に確認する必要があります。この節で、plus, minus, mtimes メソッドは、演算子 polynom/polynom や polynom/double の組み合わせでの加算、減算、乗算を行うために polynom クラスに対して定義されたものです。

+ 演算子の多重定義

p、または、q のどちらかが polynom の場合、つぎの表現

```
p + q
```

は、@polynom/plus.m 関数が (p と q が、22-63 ページの " オブジェクトの優先順位 " で記述するような、より高い優先順位のオブジェクトでなく) 存在するならば、その関数を呼び込みます。

つぎの M- ファイルは、polynom クラスに対する + 演算子を定義します。

```
function r = plus(p,q)
% POLYNOM/PLUS polynom に対して、p + q を実行
p = polynom(p);
q = polynom(q);
k = length(q.c) - length(p.c);
r = polynom([zeros(1,k) p.c] + [zeros(1,-k) q.c]);
```

関数は、まず、二つの入力引数が多項式であることを保証しなければなりません。たとえば、つぎのような表現

```
p + 1
```

は、polynom と double を含んでいますが、正常に機能します。それで、関数は、二つの係数ベクトルにアクセスし、二つを同じ長さにするために、必要ならばゼロを付け加えます。実際の加算は、2 つの係数ベクトルのベクトル和です。最終的に、関数は、三度 polynom コンストラクタを呼び、適切なタイプの結果を作成します。

- 演算子の多重定義

プラス (+) 演算子と同じアプローチを使って、多重定義されたマイナス (-) 演算子を実現できます。MATLABは、p-qを計算するために@polynom/minus.mを呼びます。

```
function r = minus(p,q)
% POLYNOM/MINUS polynoms に対して、p-q を実行
p = polynom(p);
q = polynom(q);
k = length(q.c) - length(p.c);
r = polynom([zeros(1,k) p.c] - [zeros(1,-k) q.c]);
```

* 演算子の多重定義

MATLAB は、積 p*q を計算するためにメソッド @polynom/mtimes.m を呼び出します。関数名の最初に付いている文字 m は、MATLAB の*行列* 乗算の多重定義であることに起因しています。二つの多項式の乗算は、これらの係数ベクトルのコンボリューションです。

```
function r = mtimes(p,q)
% POLYNOM/MTIMES polynoms に対して、p * q を実行
p = polynom(p);
q = polynom(q);
r = polynom(conv(p.c,q.c));
```

多重定義された演算子の使用

polynom オブジェクトを与え、

```
p = polynom([1 0 -2 -5])
```

を実行するとき、つぎの二つの関数@polynom/plus.mと@polynom/mtimes.mを呼び、

```
q = p+1
r = p*q
```

つぎの結果を出力します。

```
q =
    x^3 - 2*x - 4


r =
    x^6 - 4*x^4 - 9*x^3 + 4*x^2 + 18*x + 20
```

Polynom クラスに対する関数の多重定義

MATLAB は、係数ベクトルで表わされた多項式に適用できるいくつかの関数を既に用意しています。これらは、新しい polynom オブジェクトにも適用できるように多重定義することができます。多くの場合、多重定義されたメソッドは、オリジナルの関数を係数フィールドに適用できます。

Polynom クラスに対する roots の多重定義

メソッド @polynom/roots.m は、polynom オブジェクトの根を探索します。

```
function r = roots(p)
% POLYNOM/ROOTS.  ROOTS(p) は、p の根を含むベクトルです。
r = roots(p.c);
```

ステートメント

```
roots(p)
```

は、つぎの結果を出力します。

```
ans =


    2.0946
   -1.0473 + 1.1359i
   -1.0473 - 1.1359i
```

Polynom クラスに対する polyval の多重定義

関数 polyval は、設定した点群での多項式の値を計算します。@polynom/polyval.m は、入れ子になった乗算、または、x の異なるベキ乗を計算するために使う乗算の演算回数を減らす Horner 法のどちらかを使います。

```
function y = polyval(p,x)
% POLYNOM/POLYVAL  POLYVAL(p,x) は、点 x で p を計算します。
y = 0;
for a = p.c
   y = y.*x + a;
end
```

Polynom クラスに対する plot の多重定義

多重定義された plot 関数は、root と polyval を使います。関数は、すべての実数根を含む範囲よりもわずかに大きな範囲を独立変数に選びます。それで、polyval は、その範囲内で数百点で多項式を計算するために使われます。

```
function plot(p)
% POLYNOM/PLOT  PLOT(p) は、多項式 p をプロットします。
r = max(abs(roots(p)));
x = (-1.1:0.01:1.1)*r;
y = polyval(p,x);
plot(x,y);
title(char(p))
grid on
```

Polynom クラスに対する diff の多重定義

メソッド @polynom/diff.m は、多項式の差分を行い、次数を一つ減らし、そのオリジナルの次数と個々の係数の乗算を行います。

```
function q = diff(p)
% POLYNOM/DIFF  DIFF(p) は、多項式 p の微分を計算
c = p.c;
d = length(c) - 1;  % degree
q = polynom(p.c(1:d).*(d:-1:1));
```

クラスメソッドの一覧

関数

```
methods('class_name')
```

または、コマンド

```
methods class_name
```

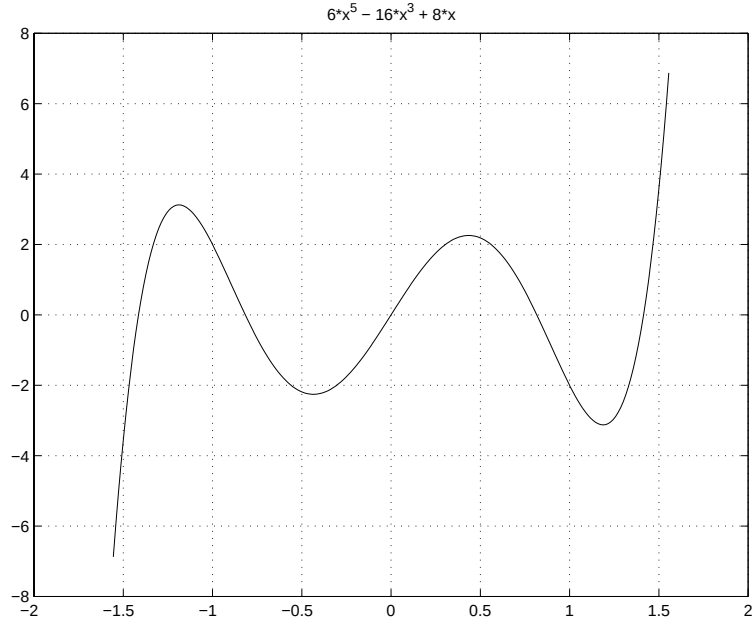
は、特別なクラスに対して有効なすべてのメソッドを表示します。polynom の例題に対して、つぎの出力を得ます。

```
methods polynom

Methods for class polynom:
```

char	display	minus	plot	polynom	roots
diff	double	mtimes	plus	polyval	subsref

二つの polynom オブジェクト x と p をプロットするために、つぎのメソッドを呼びます。

```
x = polynom([1 0]);
p = polynom([1 0 -2 -5]);
plot(diff(p*p + 10*p + 20*x) - 20)
```



他のクラスの構築

MATLAB オブジェクトは、他の MATLAB オブジェクトからプロパティや挙動を*継承*することができます。一つのオブジェクト (子オブジェクト) が、他のオブジェクト (親オブジェクト) から継承されていると、子オブジェクトは、親オブジェクトのすべてのフィールドを含み、親のメソッドを読み込むことができます。親メソッドは、親のクラスから継承した子オブジェクトのフィールドにアクセスできますが、子クラスへの新しいフィールドにはアクセスできません。

継承は、オブジェクト指向プログラミングのキーです。これは、親オブジェクトに対して存在しているコードを利用して、子オブジェクトを使ってコード化するときに再利用が簡単になります。継承は、子のオブジェクトの挙動を親オブジェクトのものと全く同じにすることができます。これは、同様な機能をもつ関係したクラスを開発に有効ですが、別々に実現されます。

継承には 2 種類のものがあります。

- 単一継承、これは、子オブジェクトがある親のクラスから特性を継承するものです。
- 多重継承、これは、子オブジェクトが複数の親クラスから特性を継承するものです。

この節は、関連したトピック、*集約* についても議論します。集約は、そのメンバの一つとして一つのオブジェクトに他のオブジェクトを含ませることができます。

単一継承

他のクラスから派生して、継承しているクラスにそれ自身の新しい要素を加えるクラスは、単一継承を使います。継承は子クラスに属しているオブジェクトが、追加フィールドのように親クラスと同じフィールドをもたせます。そのために、親クラスと関連したメソッドは、子クラスに属するオブジェクト上で働くことができます。しかし、子クラスに関連したメソッドは、親クラスに属しているオブジェクト上で機能しません。子クラスから直接に親のフィールドにアクセスすることはできません。親に対して定義されたアクセスメソッドを使わなければなりません。

他の挙動を継承しているクラスに対するコンストラクタ関数は、2 つの特別な特性をもっています。

- 親クラスに対して、"継承された" フィールドを作るためのコンストラクタ関数を読み込みます。

- class 関数に対して使う構文は、子クラスと親クラスの 2 つを反映するためにわずかに異なります。

関数 class を使って、単一継承関係を作る一般的な構文は、つぎのものです。

```
child_obj = class(child_obj,'child_class',parent_obj);
```

単一継承は、複数の世代にまたがることも可能です。親クラスが、それ自身継承クラスの場合、子オブジェクトは二世代上 (grandparent) のクラスから自動的に継承します。

クラスプロパティとメソッドの可視化

親クラスは、子のプロパティ、または、メソッドに関する知識をもつ必要はありません。子クラスは、親のプロパティに直接アクセスできませんが、親のアクセスメソッド (たとえば、get または subsref メソッド) を使って、親のプロパティにアクセスします。子クラスメソッドから、このアクセスは、子構造体内の親のフィールドを通して行われます。たとえば、コンストラクタは、子オブジェクト c を作成するとき、つぎのようにします。

```
c = class(c,'child_class_name',parent_object);
```

MATLABは、親オブジェクトを含むオブジェクトの構造体内に c.*parent_class_name* フィールドを自動的に作成します。ユーザは、親の display メソッドを呼び、子の display メソッドの中にステートメントを作成します。

```
display(c.parent_class_name)
```

単一継承を使った例題については、22-43 ページの " 株式クラスの設計 " を参照してください。

多重継承

多重継承の場合、オブジェクトのクラスは、複数の親クラスから派生するものを継承します。子オブジェクトは、それ自身のフィールドと同様にすべての親クラスからフィールドを得ます。

多重継承は、一世代以上より、もっと広いものを含みます。たとえば、各々の親オブジェクトは、複数の " ニ世代オブジェクト " 等からメンバを継承します。多重継承は、3 つ以上の引数をもつ class を読み込むことによりコンストラクタの中で実現されます。

```
obj = class(structure,'class_name',parent1,parent2,...)
```

ユーザは、クラス入力リストに希望される親の引数を加えることができます。

多重親クラスは、同じ名前の関連したメソッドをもっています。この場合、MATLAB は、コンストラクタ関数の中のclass関数の中の最初に表れる親に関連したメソッドを呼びます。この名前に続く連続的な親関数にアクセスする方法はありません。

Aggregation(集約)

標準の継承に加え、MATLAB オブジェクトは*包含*または*集約* をサポートしています。これは、一つのオブジェクトが、他のオブジェクトをそのフィールドの一つとして含んでいることを意味します。たとえば、有理数オブジェクトは、分子と分母に対する 2 つの polynom オブジェクトを使います。

外側のオブジェクトに対して、あるメソッド内からのみ含まれているオブジェクトに対するメソッドを呼び込むことができます。呼び出す関数のバージョンを決定する場合、MATLAB は、引数として渡すオブジェクトの一番外のクラスのものを考えます。すなわち、任意の含まれるオブジェクトのクラスを無視します。

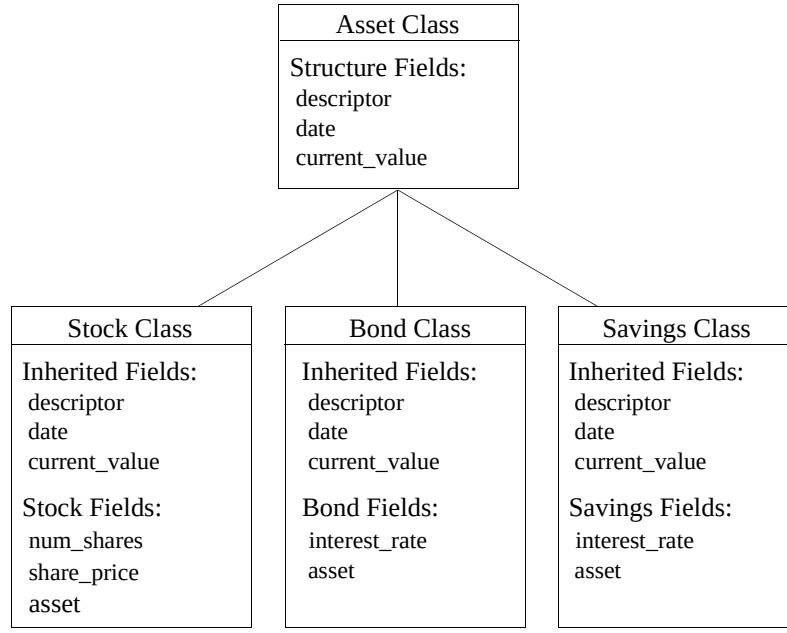
集約の例題は、22-52 ページの " 例題 - ポートフォリオコンテナ " を参照してください。

例題 - 資産と資産サブクラス

単一継承の例題として、通貨価値をもつあるアイテムを表現するために使われる一般的な資産クラスを考えます。資産のいくつかの例題、株式、債券、貯蓄勘定、他のプライスに関するものがあります。クラスのこの集合を考えるに、asset クラスは、特殊な資産サブクラスのすべてに共通なデータをもっています。stock クラスのような個々の資産サブクラスは、asset プロパティを継承し、付加的なプロパティを与えます。サブクラスは、資産の一種です。

資産クラスに対する継承モデル

ある資産親クラスを使った単一継承関係の例題を、つぎの図に示します。



図に示すように、株式 (stock)、債券 (bond)、貯蓄勘定 (saving) クラスは、資産 (asset) クラスから構造体フィールドを継承します。この例題の中で、資産クラスは、すべてのサブクラスに共通するデータに対するストレージを用意するために使われ、これらのサブクラスを使って、資産メソッドを共有できます。この例題は、資産クラスと株式クラスの実現法を示したものです。債券クラスと貯蓄勘定クラスは、資産サブクラスの他のタイプと同じように、株式クラスに非常に似

た方法で実現できます。

資産クラスの設計

資産クラスは、すべての資産の子に共通する情報のストレージを用意し、それにアクセスできます。それは、直接行なおうとするものではなく、そのために、メソッドを拡張する必要もありません。この目的を達するため、クラスは、つぎのメソッドを含むことを必要とします。

- コンストラクタ
- get と set
- subsref と subsasgn
- display

他の資産メソッド

資産クラスは､その子クラスに対して継承されるデータストレージを与えますが､直接、例証されるものではありません。set, get, display メソッドは、ストアされたデータに直接アクセスします。子クラスのみが (データにアクセスするので、コンバータ、エンド、サブインデックスのような) 資産オブジェクトに対して、完全な型でメソッドを実現する必要はありません。

資産コンストラクタメソッド

資産クラスは､つぎの三つのフィールドをもつ構造体配列をベースにしています。

- descriptor - 特別な資産 (たとえば､株式の名前､貯蓄勘定の番号等々) の識別子
- date - オブジェクトが作成された日時 (date コマンドで計算可能)
- type - 資産のタイプ (例 , 貯蓄勘定、債券、株式)
- current_value - 資産の現在の価値 (サブクラスデータから計算可能)

この情報は、資産子オブジェクト (株式、債券、貯蓄勘定) に共通で、各子クラスの中に同じフィールドを定義することを避けるために、親オブジェクトから取り扱われます。これは、子クラスの数が増えると共に便利なものになります。

```
function a = asset(varargin)
% ASSET 資産オブジェクト用のコンストラクタ関数
% a = asset(descriptor, current_value)
switch nargin
case 0
```

```
% 入力引数がない場合、デフォルトオブジェクトを作成
    a.descriptor = 'none';
    a.date = date;
    a.type = 'none';
    a.current_value = 0;
    a = class(a,'asset');
case 1
% クラス asset の引数が単一の場合それを出力
    if (isa(varargin{1},'asset'))
        a = varargin{1};
   else
        error('Wrong argument type')
    end
case 3
% 指定した値を使って、オブジェクトを作成
    a.descriptor = varargin{1};
    a.date = date;
    a.type = varargin{2};
    a.current_value = varargin{3};
    a = class(a,'asset');
otherwise
    error('Wrong number of input arguments')
end
```

関数は、三つの可能なシナリオをまとめて、switch ステートメントを使っています。

- 引数なしで呼び出すと、コンストラクタはデフォルトの資産オブジェクトを戻します。
- 資産オブジェクトを一つ引数として呼び出すと、オブジェクトが単に戻されます。
- 二つの引数 (サブクラス識別子と現在の値) と共に呼び出すと、コンストラクタは新しい資産オブジェクトを出力します。

資産コンストラクタメソッドは、直接呼ばれることはありません。すなわち、その目的が共通のデータに対するストレージを与えることなので、子コンストラクタから呼ばれます。

資産 get メソッド

資産クラスは、資産オブジェクトの中に含まれる日にアクセスするメソッドを必要とします。つぎの関数は、クラスに対する get メソッドを実現するものです。

これは、他の MATLAB オブジェクトと同じようにインタフェースを取るために、厳密なフィールド名でなく、識別可能な型で記述されたプロパティ名を使います。

```
function val = get(a,prop_name)
% GET 指定したオブジェクトから資産プロパティを得、
% 値を戻します。
switch prop_name
case 'Descriptor'
    val = a.descriptor;
case 'Date'
    val = a.date;
case 'CurrentValue'
    val = a.current_value;
otherwise
    error([prop_name,' Is not a valid asset property'])
end
```

この関数は、オブジェクトとプロパティ名を受け入れ、switch ステートメントを使って、アクセスするフィールドを決定します。このメソッドは、継承されたプロパティ内のデータにアクセスするとき、サブクラス get メソッドで呼ばれます。例題は、22-45 ページの " 株式 get メソッド " を参照してください。

資産 set メソッド

資産クラス set メソッドは、サブクラス set メソッドにより呼び込まれます。このメソッドは、資産オブジェクトとプロパティ名 / プロパティ値の可変の組み合わせを表わす引数を受け入れ、修正されたオブジェクトを出力します。

```
function a = set(a,varargin)
% SET 資産プロパティを設定し、アップデートオブジェクトを戻します。
property_argin = varargin;
while length(property_argin) >= 2,
    prop = property_argin{1};
    val = property_argin{2};
    property_argin = property_argin(3:end);
    switch prop
    case 'Descriptor'
         a.descriptor = val;
    case 'Date'
         a.date = val;
    case 'CurrentValue'
```

```
            a.current_value = val;
        otherwise
            error('Asset properties: Descriptor, Date, CurrentValue')
        end
    end
```

サブクラス set メソッドは、資産 set メソッドを呼びますが、MATLAB がリファレンスにより渡される引数をサポートしていないので、修正されたオブジェクトを戻す機能を必要とします。例題については、22-46 ページの " 株式 set メソッド " を参照してください。

資産 subsref メソッド

subsref メソッドは、1 から始まる数値インデックスと構造体フィールド名を使ったインデックスを使って、資産オブジェクトの中に含まれる日にアクセスします。外側の switch ステートメントは、インデックスが、数値、または、フィールド名のどちらかを使ったものであるかを判断するものです。内側の switch ステートメントは、インデックスを適切な値に射影します。

MATLAB は、サブスクリプトを使ったリファレンスをあるオブジェクト (たとえば、A(i), A{i}, A.*fieldname*) にするときに、subsref を呼びます。

```
function b = subsref(a,index)
%SUBSREF 資産オブジェクト用のインデックス付きのフィールド名を定義
switch index.type
case '()'
    switch index.subs{:}
    case 1
        b = a.descriptor;
    case 2
        b = a.date;
    case 3
        b = a.current_value;
    otherwise
        error('Index out of range')
    end
case '.'
    switch index.subs
    case 'descriptor'
        b = a.descriptor;
    case 'date'
```

```
            b = a.date;
        case 'current_value'
            b = a.current_value;
        otherwise
            error('Invalid field name')
        end
    case '{}'
        error('Cell array indexing not supported by asset objects')
    end
```

子 subsref メソッドが親 subsred メソッドを呼び出す方法の例題は、22-47 ページの " 株式 subsref メソッド " を参照してください。

資産 subsasgn メソッド

subsasgn メソッドは、subsref メソッドと同じ代入となります。このバージョンを使って、1から始まる数値インデックスと構造体フィールド名を使ったインデックスを使ってあるオブジェクトに含まれるデータを変更することができます。外側の switch ステートメントは、インデックスが、数値、または、フィールド名のどちらかを使ったものであるかを判断するものです。内側の switch ステートメントは、インデックスを適切な値に射影します。

MATLAB は、代入ステートメント (たとえば、A(i) = val, A{i} = val, または A.*fieldname* = val) を実行する場合、subsasgn を呼びます。

```
function a = subsasgn(a,index,val)
% SUBSASGN 資産オブジェクト用のインデックスの代入を定義
switch index.type
case '()'
    switch index.subs{:}
    case 1
        a.descriptor = val;
    case 2
        a.date = val;
    case 3
        a.current_value = val;
    otherwise
        error('Index out of range')
    end
case '.'
    switch index.subs
```

```
        case 'descriptor'
            a.descriptor = val;
        case 'date'
            a.date = val;
        case 'current_value'
            a.current_value = val;
        otherwise
            error('Invalid field name')
        end
    end
```

subsasgn メソッドは、二つの手法を使って資産オブジェクトデータ構造体に値を代入することができます。たとえば、ユーザは子株式オブジェクト s

```
s = stock('XYZ',100,25);
```

をもっているとします。株式クラスメソッドの中で、つぎのステートメントのどちらかを使って、descriptor フィールドを変更できます。

```
s.asset(1) = 'ABC';
```

または、

```
s.asset.descriptor = 'ABC';
```

子 subsasgn メソッドが親 subsasgn メソッドを呼び込む方法の例題は、22-49 ページの " 株式 subsasgn メソッド " を参照してください。

資産 display メソッド

資産 display メソッドは、子クラス display メソッドから呼び込まれるように設計されています。この目的は、子オブジェクトに対してストアしているデータを表示することです。メソッドは、子の display メソッドの形式と一致する方法に表示用にデータを単に書式化します。

```
function display(a)
% DISPLAY(a) 資産オブジェクトの表示
stg = sprintf(...
    'Descriptor: %s\nDate: %s\nType: %s\nCurrent Value:%9.2f',...
    a.descriptor,a.date,a.current_value);
disp(stg)
```

株式クラス display メソッドは、親クラス内にストアされているデータを表示するためにこのメソッドを呼びます。このアプローチは、資産クラスの変更と株式

display メソッドを分けます。このメソッドの呼び出し方法の例題は、22-50 ページの " 株式 display メソッド " を参照してください。

資産 fieldcount メソッド

資産 fieldcount メソッドは、資産オブジェクトデータ構造体の中のフィールド数を戻します。fieldcount は、子メソッドが、資産クラスを知るために必要であるというよりは、むしろ、子メソッドを使って、実行中に資産オブジェクトの中のフィールド数を決定します。これを使って、子クラスメソッドを変更しないで、資産クラスデータ構造体の中のフィールド数を変更させることができます。

```
function num_fields = fieldcount(asset_obj)
% 資産オブジェクトの中のフィールド数を決定
% 子クラスメソッドで使用されます。
num_fields = length(fieldnames(asset_obj));
```

関数 struct は、あるオブジェクトを等価なデータの構造体に変換します。この構造体の中身にアクセスすることができます。

株式クラスの設計

株式オブジェクトは、ある個人の投資のポートフォリオの中のある特別な資産を表現するために設計されています。このオブジェクトは、それ自身の二つのプロパティと親資産オブジェクトから継承する三つのプロパティを含んでいます。株式プロパティは、つぎの二つです。

株式プロパティ

- NumberShares - 特定の株式オブジェクトに対する株数
- SharePrice - 各株の価値

資産プロパティ

- Descriptor - 特定の資産の識別 (たとえば、資産名、貯蓄勘定数等)
- Date - オブジェクトが作成された日 (date コマンドで計算)
- CurrentValue - 資産の現在の価値

プロパティ名は、株式オブジェクトと資産オブジェクトの内部で使われる構造体配列のフィールド名と同じではありません。プロパティ名インタフェースは、株式と資産の get と set メソッドでコントロールされ、他の MATLAB オブジェクトプロパティのインタフェースに似て設計されます。

株式オブジェクト構造体の asset フィールドは、親資産オブジェクトを含み、親構造体の継承されるフィールドにアクセスするのに使われます。

株式クラスメソッド

株式クラスは、つぎのメソッドを実現します。

- コンストラクタ
- get と set
- subsref と subsasgn
- display

株式クラスコンストラクタ

株式コンストラクタは、三つの入力引数から株式オブジェクトを作成します。

- 株式名
- 株式数
- 株式価格

コンストラクタは、資産オブジェクトを株式コンストラクタの中から作成し、それを親として株式オブジェクトに設定することができます。そのため、株式コンストラクタは、資産コンストラクタを呼びます。株式オブジェクトを作成するために呼ばれる関数 class は、親として資産オブジェクトを定義します。

資産オブジェクトが株式コンストラクタ関数のテンポラリなワークスペースの中で作成され、株式構造体の中にあるフィールド (.asset) としてストアされます。株式オブジェクトは、資産フィールドを継承しますが、資産オブジェクトはベースのワークスペースに戻されません。

```
function s = stock(varargin)
% STOCK 株式クラスコンストラクタ
% s = stock(descriptor, num_shares, share_price)
switch nargin
case 0
% 入力引数がない場合、デフォルトオブジェクトが作成されます。
    s.num_shares = 0;
    s.share_price = 0;
    a = asset('none',0);
    s = class(s, 'stock',a);
case 1
```

```
% クラス stock の引数が一つの場合、それを戻します。
    if (isa(varargin{1},'stock'))
        s = varargin{1};
    else
        error('Input argument is not a stock object')
    end
case 3
% 指定した値を使って、オブジェクトを作成
    s.num_shares = varargin{2};
    s.share_price = varargin{3};
    a = asset(varargin{1},'stock',varargin{2} * varargin{3});
    s = class(s,'stock',a);
otherwise
    error('Wrong number of input arguments')
end
```

コンストラクタの呼び込み構文

株式コンストラクタメソッドは、つぎの 3 つの方法のいずれかを使って呼び込みます。

- 入力引数なし - 入力引数を設定しないで呼ぶ場合、コンストラクタは空のフィールドをもつデフォルトオブジェクトを戻します。
- 入力引数が株式オブジェクト - 株式オブジェクトであるものを単一の入力引数として呼ぶ場合、コンストラクタは入力引数を出力します。株式オブジェクトでないものを単一引数とすると、エラーになります。
- 三つの入力引数をもつ - 三つの入力引数をもつ場合、コンストラクタは、株式オブジェクトを定義するために使います。

その他 - 上の三つの場合のいずれにも当らない場合、エラーになります。

たとえば、つぎのステートメントは、一株 25 ドルの XYZ 社の株式 100 株の所有を記録する株式オブジェクトを作成します。

```
XYZ_stock = stock('XYZ',100,25);
```

株式 get メソッド

get メソッドは、Handle Graphics と同様に " プロパティ名 " を指定するインタフェースを使って株式オブジェクトのデータをアクセスする方法を提供します。この例題の中で、プロパティ名は、構造体のフィールド名と同じですが、全く異なっ

たものを付けることもできます。get メソッドを通してのアクセスからあるフィールドを除外するようにでき、または、このような挙動が適しているならば、種々のプロパティ名に対して、同じフィールドからデータを戻すこともできます。

```
function val = get(s,prop_name)
% GET 指定したオブジェクトから株式プロパティを取得し、
% 値を出力します。プロパティ名：NumberShares
% SharePrice, Descriptor, Date, CurrentValue
switch prop_name
case 'NumberShares'
    val = s.num_shares;
case 'SharePrice'
    val = s.share_price;
case 'Descriptor'
    val = get(s.asset,'Descriptor'); % call asset get method
case 'Date'
    val = get(s.asset,'Date');
case 'CurrentValue'
    val = get(s.asset,'CurrentValue');
otherwise
    error([prop_name ,'Is not a valid stock property'])
end
```

資産オブジェクトは、株式オブジェクトの資産フィールド (s.asset) を通してアクセスされます。MATLAB は、関数 class が親の引数と共に呼ばれた場合、このフィールドを自動的に作成します。

株式 set メソッド

set メソッドは､get メソッド同様に " プロパティ名 " を使ったインタフェースを提供します。これを使って、株式数、株価、項目 (descriptor) を更新できます。カレント値と日付は、自動的に更新されます。

```
function s = set(s,varargin)
% SET 株式プロパティに指定した値を設定
% アップデートオブジェクトを出力
property_argin = varargin;
while length(property_argin) >= 2,
    prop = property_argin{1};
    val = property_argin{2};
    property_argin = property_argin(3:end);
```

```
        switch prop
        case 'NumberShares'
             s.num_shares = val;
        case 'SharePrice'
             s.share_price = val;
        case 'Descriptor'
             s.asset = set(s.asset,'Descriptor',val);
        otherwise
             error('Invalid property')
        end
    end
    s.asset = set(s.asset,'CurrentValue',...
                 s.num_shares * s.share_price,'Date',date);
```

この関数は、新しい値をもった新しい株式オブジェクトを作成し、それを戻して、古い値に上書きします。たとえば、株式オブジェクトを与えて、

```
s = stock('XYZ',100,25);
```

つぎの set コマンドは、株式価格を更新します。

```
s = set(s,'SharePrice',36);
```

MATLAB は、リファレンスで渡される引数をサポートしていないので、オリジナルの株式オブジェクト上にコピーする必要があります (たとえば、出力を s に代入)。そのため、set メソッドは、実際にオブジェクトのコピーに適用されます。

株式 subsref メソッド

subsref メソッドは、株式クラスに対するサブスクリプトを使ったインデックスを定義します。この例題で、subsref は、株式オブジェクトの数値インデックスや構造体フィールド名を使ったインデックスを使うことができます。

```
function b = subsref(s,index)
% SUBSREF 株式オブジェクト用にインデックス付けされたフィールド名を定義
fc = fieldcount(s.asset);
switch index.type
case '()'
    if (index.subs{:} <= fc)
         b = subsref(s.asset,index);
    else
         switch index.subs{:} - fc
```

```
            case 1
                b = s.num_shares;
            case 2
                b = s.share_price;
            otherwise
            error(['Index must be in the range 1 to ',num2str(fc + 2)])
            end
        end
    case '.'
        switch index.subs
        case 'num_shares'
            b = s.num_shares;
        case 'share_price'
            b = s.share_price;
        otherwise
            b = subsref(s.asset,index);
        end
    end
```

外側の switch ステートメントは、インデックスが数値かフィールド名かを決定するものです。

fieldcount 資産メソッドは、資産構造体の中にどのくらいのフィールドがあるかを決めるもので、if ステートメントは、1 から fieldcount までのインデックスに対して、資産 subsref メソッドを呼びます。これらのメソッドの記述については、22-43 ページの " 資産 fieldcount メソッド " と 22-40 ページの " 資産 subsref メソッド " を参照してください。

fieldcount により戻される数字より大きい数値インデックスは、内側の switch ステートメントで取り扱われます。これは、株式構造体の中で適切なフィールドにインデックス値を射影します。

フィールド名を使ったインデックスは、num_shares 以外のフィールド名を想定しています。share_price は資産フィールドで、それは子メソッドにより資産フィールドの知識に対する必要性をなくします。資産 subsref メソッドは、フィールド名のエラーチェックを行います。

このメソッドの実現に対する一般的な情報は、help subsref を参照してください。

株式 subsasgn メソッド

subsasgn メソッドは、数字のインデックス機能や構造体フィールド名インデックス機能を使って株式オブジェクトに含まれるデータを変更することができます。MATLAB は、代入ステートメント (A(i) = val, A{i} = val, または、A.fieldname = val) を実行するときに、subsasgn を呼びます。

```
function s = subsasgn(s,index,val)
% SUBSASGN 株式オブジェクト用にインデックス代入を定
fc = fieldcount(s.asset);
switch index.type
case '()'
    if (index.subs{:} <= fc)
        s.asset = subsasgn(s.asset,index,val);
    else
        switch index.subs{:}-fc
        case 1
            s.num_shares = val;
        case 2
            s.share_price = val;
        otherwise
        error(['Index must be in the range 1 to ',num2str(fc + 2)])
        end
    end
case '.'
    switch index.subs
    case 'num_shares'
        s.num_shares = val;
    case 'share_price'
        s.share_price = val;
    otherwise
        s.asset = subsasgn(s.asset,index,val);
    end
end
```

外側の switch ステートメントは、インデックスが数字かフィールド名かを決定します。

fieldcount 資産メソッドは、資産構造体の中にいくつのフィールドが存在しているかを決めます。そして、if ステートメントは、1 から fieldcount までのインデックスに対して資産 subsasgn メソッドを呼びます。これらのメソッドの詳細

は、22-43 ページの " 資産 fieldcount メソッド " と 22-41 ページの " 資産 subsasgn メソッド " を参照してください。

fieldcount で戻される数字よりも大きい数値インデックスは、内側の switch ステートメントにより取り扱われ、インデックス値を株式構造体の中の適切なフィールドに射影します。

フィールド名のインデックスは、num_shares や share_price とは異なるフィールドが資産フィールドであると仮定します。これにより、子メソッドにより資産フィールドの知識が必要なくなります。資産 subsasgn メソッドは、フィールド名のエラーチェックも行います。

subsasgn メソッドは、二つの手法を使って株式オブジェクトデータ構造体に値を代入することができます。たとえば、つぎの株式オブジェクトを仮定します。

```
s = stock('XYZ',100,25)
```

つぎのステートメントのどちらかを使って、descriptor フィールドを変更できます。

```
s(1) = 'ABC';
```

または、

```
s.descriptor = 'ABC';
```

MATLAB での代入ステートメントに関する一般的な情報については、subsasgn を参照してください。

株式 display メソッド

つぎのステートメント (最後にセミコロンを付けない)

```
XYZStock = stock('XYZ',100,25)
```

を実行するとき、MATLAB は @stock ディレクトリの中で display と呼ばれるメソッドを探します。株式クラスに対する display メソッドは、つぎの出力を行います。

```
Descriptor: XYZ
Date: 17-Nov-1998
Type: stock
Current Value:  2500.00
Number of shares: 100
Share price: 25.00
```

つぎに、株式 display メソッドを示します。

```
function display(s)
% DISPLAY(s) 株オブジェクトの表示
display(s.asset)
stg = sprintf('Number of shares: %g\nShare price: %3.2f\n',...
    s.num_shares,s.share_price);
disp(stg)
```

親資産オブジェクトは、資産 display メソッドに渡され、そのフィールドを表示します (入力引数が資産オブジェクトなので、MATLAB は、資産 display メソッドを呼びます)。株式オブジェクトのフィールドは、書式付きテキスト文字列を使う方法と同じです。

株式クラス display メソッドを実行できない場合、MATLAB は、資産 display メソッドを呼ぶことに注意してください。これは、機能しますが、項目、日付、タイプ、カレント値の表示のみです。

例題 - ポートフォリオコンテナ

集約は、他のクラスを使って、あるクラスを包含します。基本的な関係は、つぎのように表わします。包含クラスは、それぞれ、コンテナクラスの一部です。

たとえば、資産集合 (株式、債券、貯蓄等) に対して一つのコンテナとしてファイナンシャルポートフォリオクラスを考えます。個々の資産が一度まとめられると、それらは解析され、有益な情報を戻すことができます。包含オブジェクトは、直接アクセスできませんが、ポートフォリオクラスメソッドを通してのみアクセスできます。

このポートフォリオクラスにより集められた資産に関する詳細は、22-36 ページの " 例題 - 資産と資産サブクラス " を参照してください。

ポートフォリオクラスの設計

ポートフォリオクラスは、与えられた個々の情報をもつ種々の資産を含むように設計され、個々人の投資のポートフォリオの状態についての情報を提供するように設計できます。この例題は、つぎのように非常に簡単化されたポートフォリオクラスを実行するものです。

- 個々人の資産を集める
- ポートフォリオの内容に関する情報を表示
- ポートフォリオの中に含まれる資産タイプを相対的に示す 3 次元円グラフを表示

必要なポートフォリオメソッド

ポートフォリオクラスは、三つのメソッドのみを実行します。

- portfolio - ポートフォリオコンストラクタ
- display - ポートフォリオの内容についての情報を表示
- pie3 - 引数として単一ポートフォリオオブジェクトをもつように設計された pie3 関数の多重定義バージョン

ポートフォリオオブジェクトは、他のオブジェクトを含んでいるので、ポートフォリオクラスメソッドは、中に含んでいるオブジェクトのメソッドを使うことができます。たとえば、ポートフォリオ display メソッドは、株式クラス display メソッドを呼び込む等々です。

ポートフォリオコンストラクタメソッド

ポートフォリオコンストラクタメソッドは、入力引数としてクライアントの名前と可変長の資産サブクラスオブジェクト (この例題の中の株式、債券、貯蓄オブジェクト) のリストを使います。ポートフォリオオブジェクトは、つぎのフィールドをもつ構造体配列を使います。

- name - クライアントの名前
- ind_assets - 資産サブクラスオブジェクト (株式、債券、貯蓄) の配列
- total_value - すべての資産のトータルの価値。コンストラクタは、引数として渡されるオブジェクトからこの価値を計算します。
- account_number - アカウント番号。このフィールドは、ポートフォリオオブジェクト (22-58 ページの " オブジェクトのセーブとロード " を参照) をセーブしたときのみ値を割り当てます。

```
function p = portfolio(name,varargin)
% PORTFOLIO クライアント名と投資リストを含む
% ポートフォリオオブジェクトを作成
switch nargin
case 0
    % 入力引数がない場合、デフォルトオブジェクトが作成されます。
    p.name = 'none';
    p.total_value = 0;
    p.ind_assets = {};
    p.account_number = '';
    p = class(p,'portfolio');
case 1
    % クラスポートフォリオの引数が一つの場合、それを出力します。
    if isa(name,'portfolio')
        p = name;
    else
        disp([inputname(1) ' is not a portfolio object'])
        return
    end
otherwise
    % 指定した引数を使って、オブジェクトを作成
    p.name = name;
    p.total_value = 0;
    for i = 1:length(varargin)
```

```
            p.ind_assets(i) = {varargin{i}};
            asset_value = get(p.ind_assets{i},'CurrentValue');
            p.total_value = p.total_value + asset_value;
        end
        p.account_number = '';
        p = class(p,'portfolio');
    end
```

コンストラクタ呼び込み構文

ポートフォリオコンストラクタメソッドは、つぎの異なる三つの方法のいずれかで呼び込まれます。

- 入力引数なし - 入力引数なしで呼び込む場合、空フィールドをもつオブジェクトを戻します。
- 入力引数がオブジェクトの場合 - 入力引数が既にポートフォリオオブジェクトの場合、MATLAB は入力引数を戻します。関数 isa は、このケースに対するチェックです。
- 複数の入力引数をもつ場合 - 複数の入力引数をもつ場合、コンストラクタは、最初にクライアント名を、残りに資産サブクラスオブジェクトを配置していることを仮定しています。より十分な実現には、より注意深い入力引数のチェック、たとえば、関数 isa を使って引数がオブジェクトの正しいクラスか否かを決定する等のチェックを行います。

ポートフォリオ display メソッド

ポートフォリオ display メソッドは、オブジェクトの display メソッドを呼ぶことにより各々に含まれるオブジェクトの内容をリストします。そして、クライアント名やトータルの資産価値もリストします。

```
function display(p)
% DISPLAY ポートフォリオオブジェクトを表示
for i=1:length(p.ind_assets)
    display(p.ind_assets{i})
end
stg = sprintf('\nAssets for Client: %s\nTotal Value: %9.2f\n',...
p.name,p.total_value);
disp(stg)
```

ポートフォリオ pie3 メソッド

ポートフォリオクラスは、MATLAB 関数 pie3 を多重定義して、ポートフォリオオ

ブジェクトを受け入れ、クライアントのポートフォリオ全体の中で個々の資産の割合を示す 3 次元円グラフを表示します。MATLAB は、入力引数が単一のポートフォリオオブジェクトの場合、pie3 の @portfolio/pie3.m バージョンを呼び出します。

```
function pie3(p)
% PIE3 ポートフォリオの 3-D pie チャートを作成
stock_amt = 0; bond_amt = 0; savings_amt = 0;
for i=1:length(p.ind_assets)
    if isa(p.ind_assets{i},'stock')
        stock_amt = stock_amt + ...
            get(p.ind_assets{i},'CurrentValue');
    elseif isa(p.ind_assets{i},'bond')
        bond_amt = bond_amt + ...
            get(p.ind_assets{i},'CurrentValue');
    elseif isa(p.ind_assets{i},'savings')
        savings_amt = savings_amt + ...
            get(p.ind_assets{i},'CurrentValue');
    end
end
i = 1;
if stock_amt ~= 0
    label(i) = {'Stocks'};
    pie_vector(i) = stock_amt;
    i = i +1;
end
if bond_amt ~= 0
    label(i) = {'Bonds'};
    pie_vector(i) = bond_amt;
    i = i +1;
end
if savings_amt ~= 0
    label(i) = {'Savings'};
    pie_vector(i) = savings_amt;
end
pie3(pie_vector,label)
set(gcf,'Renderer','zbuffer')
set(findobj(gca,'Type','Text'),'FontSize',14)
cm = gray(64);
colormap(cm(48:end,:))
```

```
stg(1) = {['Portfolio Composition for ',p.name]};
stg(2) = {['Total Value of Assets: $',num2str(p.total_value)]};
title(stg,'FontSize',12)
```

多重定義された pie3 メソッドは、三つの部分から構成されています。

- まず、資産サブクラス get メソッドを使って、個々の包含されているオブジェクトの CurrentValue プロパティにアクセスします。各クラスのトータル価値が加算されます。
- 二番目の部分は、存在しているオブジェクトに依りますが、円グラフのラベルを作成し、グラフデータのベクトルを作成します。
- 三番目の部分は、MATLAB pie3 関数を呼び込み、あるフォントを作成し、カラーマップを調整し、タイトルを加えます。

ポートフォリオの作成

株式クラスの場合と同様に資産サブクラスの選択を行うと仮定します。ポートフォリオオブジェクトを使って、個々のファイナンシャルポートフォリオを表示することができます。たとえば、つぎの資産を与えて、つぎの関数

```
XYZStock = stock('XYZ',200,12);
SaveAccount = savings('Acc # 1234',2000,3.2);
Bonds = bond('U.S. Treasury',1600,12);
```

を使うことにより実現できます。さて、ポートフォリオオブジェクトを作成します。

```
p = portfolio('Gilbert Bates',XYZStock,SaveAccount,Bonds)
```

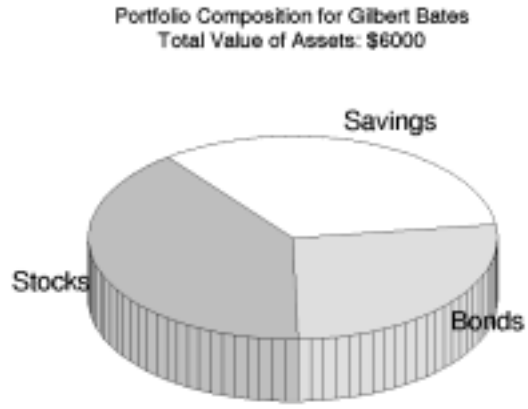
ポートフォリオ display メソッドは、(ここで使っているステートメントは、ラインの最後にセミコロンを付けていないので) ポートフォリオの内容をまとめます。

```
Descriptor: XYZ
Date: 24-Nov-1998
Current Value:   2400.00
Type: stock
Number of shares: 200
Share price: 12.00

Descriptor: Acc # 1234
Date: 24-Nov-1998
Current Value:   2000.00
Type: savings
Interest Rate: 3.2%
```

```
Descriptor: U.S. Treasury
Date: 24-Nov-1998
Current Value:   1600.00
Type: bond
Interest Rate: 12%

Assets for Client: Gilbert Bates
Total Value:   6000.00
```

ポートフォリオ pie3 メソッドは、円グラフを使って、資産の相対的な割合を表示します。

```
pie3(p)
```



オブジェクトのセーブとロード

MATLAB の save コマンドと load コマンドを使って、変数のように、ユーザ定義のオブジェクトを .mat ファイルにセーブしたり、.mat ファイルをロードしたりできます。

オブジェクトをロードするとき、MATLAB は、ワークスペースにオブジェクトを配置させるためにオブジェクトのクラスコンストラクタを呼びます。ユーザがロードするオブジェクトクラス用にコンストラクタ関数は、入力引数をもたないで呼び、デフォルトのオブジェクトを出力できるものでなければなりません。詳細は、22-10 ページの " コンストラクタを記述するためのガイドライン " を参照してください。

セーブ、または、ロード中のオブジェクトの変更

オブジェクト群に save または load コマンドを適用した場合、MATLAB は、クラスディレクトリの中から saveobj と loadobj と呼ばれるクラスメソッドを探します。ユーザは、セーブ演算またはロード演算を行う前にオブジェクトを修正するために、これらのメソッドを多重定義することができます。たとえば、saveobj メソッドに、オブジェクトと共に関連したデータもセーブさせたり、オブジェクトのあるタイプが MATLAB ワークスペースの中にロードされるとき、より新しいバージョンにオブジェクトを更新する loadobj メソッドを記述することもできます。

例題 - ポートフォリオに対する saveobj と loadobj の定義

22-52 ページの " 例題 - ポートフォリオコンテナ " の節で、ポートフォリオオブジェクトは、クライアントの投資ポートフォリオに関する情報を集めるのに使われています。そして、アカウント番号をセーブする個々のポートフォリオオブジェクトに加えることを仮定します。セーブ演算中に自動的にこの仕事を実行するポートフォリオ saveobj メソッドを定義できます。

アカウント番号をもたないポートフォリオオブジェクトの番号を既にセーブしていると仮定します。ユーザはロード中にこれらのオブジェクトをアップデートしようとします。そして、まだ、正しいポートフォリオオブジェクトのままの状態を仮定します。ポートフォリオクラスに対して、loadobj メソッドを定義することにより、これを実行します。

コード変更のまとめ

アカウント番号のシナリオを実行するため、つぎの関数を加えたり、変更する必要があります。

- portfolio - ポートフォリオコンストラクタメソッドは、新しいフィールド account_number を作成するために変更する必要があります。このフィールドは、オブジェクトが作成されるとき、空の文字列に初期化されるものです。
- saveobj - 新しいポートフォリオメソッドが、セーブ作業中に、アカウント番号をポートフォリオオブジェクトに加えるように設計されています。これは、オブジェクトがまだアカウント番号をもっていない場合のみです。
- loadobj - 新しいポートフォリオメソッドが、アカウント番号構造体フィールドが加えられる前のセーブされるポートフォリオオブジェクトのより古いバージョンをアップデートするように設計します。
- subsref - ポートフォリオメソッドの範囲外にあるポートフォリオオブジェクトにサブスクリプトを使ったリファレンスにする新しいポートフォリオメソッドです。
- getAccountNumber - クライアント名の最初の三文字から構成されるアカウント番号を戻す MATLAB 関数

新しいポートフォリオクラスの挙動

この例題の中で、行われる付加や変更で、ポートフォリオクラスは、つぎのようになります。

- アカウント番号用のフィールドを含む

- ポートフォリオオブジェクトが、最初にセーブされる場合のアカウント番号を加える
- これらを、MATLAB ワークスペースにロードすると、自動的に、ポートフォリオオブジェクトの古いバージョンをアップデートします。

saveobj メソッド

MATLAB は、save コマンドがポートフォリオオブジェクトに渡される場合は、常に、ポートフォリオ saveobj メソッドを探します。@portfolio/saveobj が存在する場合、MATLAB は、ポートフォリオオブジェクトを saveobjj に渡し、一つの出力引数として変更されたオブジェクトを戻すはずです。saveobj のつぎの実行は、オブジェクトが、既に前もって用意されているセーブ演算からアカウント番号を割り当てられているか否かを決定します。割り当てられていない場合、saveobj は、番号を得るために getAccountNumber を呼び、それを account_numberr フィールドに割り当てます。

```
function b = saveobj(a)
if isempty(a.account_number)
    a.account_number = getAccountNumber(a);
end
b = a;
```

loadobj メソッド

MATLAB は、load コマンドがロードされる .mat ファイルの中のポートフォリオオブジェクトを調べるときに、ポートフォリオloadobjメソッドを探します。loadobj が存在する場合、MATLAB は、ポートフォリオオブジェクトを loadobj に渡し、一つの出力引数として変更されたオブジェクトを戻すはずです。出力引数は、ワークスペースの中にロードされます。

入力オブジェクトが、コンストラクタ関数により設定されたものと現在の定義が一致しない場合、MATLAB は、それを、同じフィールドとすべての値をそのままそっくりもつオブジェクトの構造体を含む一つの構造体に変換します(ここでは、構造体をもつことになり、オブジェクトをもつのではありません)。

loadobj のつぎの実現は、まず、入力引数がポートフォリオオブジェクトかまたは構造体のどちらかを決定するために、関数 isa を使います。入力がオブジェクトならば、変更は必要ないので、そのまま、何もしないで戻すだけです。入力引数が MATLAB に依って構造体に変換されていると、新しい account_number フィールドが構造体に加えられ、更新されたポートフォリオオブジェクトを作成するために使われます。

```
function b = loadobj(a)
% ポートフォリオクラス用の loadobj
if isa(a,'portfolio')
    b = a;
else % a is an old version
    a.account_number = getAccountNumber(a);
    b = class(a,'portfolio');
end
```

ポートフォリオコンストラクタの変更

ポートフォリオ構造体配列は、アカウント番号を集める付加的なフィールドを必要とします。このフィールドを作成するため、つぎのライン

```
p.account_number = '';
```

を @portfolio/portfolio.m にゼロ引数と可変引数部分をもたせて加えます。

getAccountNumber 関数

この例題で、getAccountNumber は、MATLAB 関数で、クライアント名の最初の三文字とその後に数字を付けてアカウント番号を作成し、それを戻します。実現する技法を示すために、getAccountNumber は、ポートフォリオメソッドではなく、そのため、直接ポートフォリオオブジェクトデータにアクセスできません。それで、ポートフォリオオブジェクト構造体の中の名前のフィールドにアクセスできるポートフォリオ subsref メソッドを定義する必要があります。

この例題に対して、getAccountNumber は、一つの乱数を発生させ、書式化し、ポートフォリオ名フィールドの 1 番目から 3 番目の要素に連結するものです。

```
function n = getAccountNumber(p)
% オブジェクト p 用のアカウント番号を与える
n = [upper(p.name(1:3)) strcat(num2str(round(rand(1,7)*10))')'];
```

ポートフォリオオブジェクトは、フィールド名でインデックスが付加され、その後で、最初の三文字を抽出するために数字でサブスクリプトが使われています。subsref メソッドは、サブスクリプトを使ったリファレンスのこの型をサポートするように記述しなければなりません。

ポートフォリオ subsref メソッド

MATLAB はサブスクリプトを使ったリファレンスを見つけると、たとえば、getAccountNumber 関数の中で、つぎのようにすると、

```
p.name(1:3)
```

MATLAB は、リファレンスを解釈するために、ポートフォリオ subsref メソッドを呼びます。subsref メソッドを定義しない場合、上のステートメントは、ポートフォリオオブジェクトに対して定義されません (ここで、p はオブジェクトで、構造体でないことに注意してください)。

ポートフォリオ subsref メソッドは、ポートフォリオ name フィールドにアクセスするために getAccountNumber 関数に対して、フィールド名と数字をインデックスとして使用することをサポートしています。

```
function b = subsref(p,index)
% SUBSREF ポートフォリオオブジェクト用にインデックスされたフィールド名を定義
switch index(1).type
case '.'
    switch index(1).subs
    case 'name'
        if length(index)== 1
            b = p.name;
        else
            switch index(2).type
            case '()'
                b = p.name(index(2).subs{:});
            end
        end
    end
end
```

subsref のポートフォリオ実現は、name フィールドの特定の要素にアクセスするように設計されています。そして、subref の株式クラス実現のようなすべての構造体データにアクセスする一般的な方法はありません。

インデックス機能とオブジェクトに関する詳細については、subsref を参照してください。

オブジェクトの優先順位

オブジェクトの優先順位は、ある状況の中で、種々のバージョンの演算子や関数が存在するときに、どのバージョンを使うかを決めるものです。オブジェクトの優先順位は、オブジェクトの種々のクラスを含む表現の挙動をコントロールするものです。たとえば、つぎの表現を考えます。

```
objectA + objectB
```

通常、MATLAB は、オブジェクトが同じ優先順位をもつと仮定し、最も左側に位置するものから呼び込みます。しかし、つぎのように二つの例外があります。

- ユーザ定義のクラスは、MATLAB の組み込みクラスより優先順位が高い
- ユーザ定義のクラスは、関数 inferiorto や superiorto を使って、他のユーザ定義のクラスと比べて優先順位を設定することができます。

たとえば、22-23 ページの " 例題 - Polynomial クラス " の節の中で、polynom クラスは、多項式同志の加算が可能になる plus メソッドを定義します。polynom オブジェクト p を与えると、

```
p = polynom([1 0 -2 -5])

p =

    x^3-2*x-5
```

つぎの表現

```
1 + p

ans =

    x^3-2*x-4
```

は、polynom の plus メソッドを呼びます (まず、double の 1 を polynom オブジェクトに変換し、それを p に加えます)。ユーザ定義の polynom クラスは、MATLAB の double クラスよりも優先順位が上です。

ユーザ定義クラスの優先順位の設定

クラスコンストラクタの中で、inferiorto または superiorto 関数を呼び込むことにより、ユーザ定義のクラスの相対的な優先順位を設定することができます。

関数 inferiorto は、優先順位に関する階層構造の中で他のクラスの下にあるクラスを設定するものです。関数 inferiorto に関する呼び込み構文は、つぎのようになります。

```
inferiorto('class1','class2',...)
```

引数のリストの中に複数のクラスを、階層構造の中で多くの他のクラスの下にあるクラスを設定する等の設定ができます。

同様に、関数 superiorto は、優先順位に関する階層構造の中で、他のクラスの上に設定することができます。その読み込み構文は、つぎのようになります。

```
superiorto('class1','class2',...)
```

階層構造の中での位置

objectA が、優先順位に関する階層構造の中で *objectB* より上に位置するとすると、つぎの表現

```
objectA + objectB
```

は、@*classA*/plus.m を呼びます。逆に、*objectB* が優先順位に関する階層構造の中で *objectA* より上に位置する場合、MATLAB は @*classB*/plus.m を呼びます。

関連する情報については、22-65 ページの "MATLAB がメソッドを読み込む優先順位の決定法 " を参照してください。

MATLAB がメソッドを読み込む優先順位の決定法

MATLAB の中で、関数は、コンピュータのファイルシステムの中のディレクトリに存在しています。ディレクトリは、多くの関数 (M- ファイル) を含んでいます。関数名は一つのディレクトリの中で唯一なものです (たとえば、複数のディレクトリに pie3 と呼ばれる関数を含むことができます)。コマンドライン上で関数名をタイプすると、MATLAB は、すべてのディレクトリを探索し、どの関数が呼ばれるかを知ることができます。ディレクトリのこのリストは、*MATLAB パス*と言われます。

関数の探索で、MATLAB はパスの中に記述されている順にディレクトリを探索します。そして、指定した関数名と最初に一致した関数を呼びます。

pie3.m という M- ファイルを記述し、MATLAB が用意している pie3.m を含んでいるディレクトリ specgraph より前のディレクトリに配置したとすると、MATLAB は、ユーザが指定した関数 pie3.m を使います (しかし、必ず最初に位置する組み込み関数、たとえば、plot などはこの規則が適用されません)。

オブジェクト指向プログラミングでは、同じ名前をもつ複数のメソッド (クラスディレクトリ内に位置する MATLAB 関数) をもつことが可能で、関数に渡す変数のクラスやタイプをベースに使用するメソッドがどれかを MATLAB に設定することができます。たとえば、p がポートフォリオオブジェクトの場合、つぎのステートメント

```
pie3(p)
```

は、引数がポートフォリオオブジェクトであるので、@portfolio/pie3.m を呼びます。

メソッド選択手順

同じ名前をもつ種々のバージョンが存在する場合に、その一つを呼び込む場合、MATLAB はつぎの方法でメソッドを決めます。

- 引数リストの中でどの引数が最優先順位をもっているかを決めるためにオブジェクトのクラスを調べます。すなわち、このオブジェクトのクラスは、メソッド選択をコントロールし、*デイスパッチタイプ*と言われます。
- メソッドの中で、可能ないくつかの実行法を決定するために、*関数の優先順位*を適用します。この順序は、関数の配置されている位置、タイプに依存します。

Dispatch タイプの決定

MATLAB は、まず、メソッド選択をコントロールする引数を決定します。この引数のクラスタイプが、MATLABが探索するメソッドの入っているクラスを決定します。

コントロールされる引数は、つぎのいずれかです。

- 優先順位の最も高いものをもつ引数
- 優先順位が等しいものは、最も左にあるものから

ユーザ定義のオブジェクトは、double や char のような MATLAB の組み込みのクラスよりも上に順位を設定できます。22-63 ページの " オブジェクトの優先順位 " に示されるように、関数 inferiorto や superiorto を使って、ユーザ定義のオブジェクトの相対的な優先順位の設定できます。

MATLAB は名前を使って関数を探索します。関数を読み込むと、MATLAB は、名前、引数の数、各引数のタイプを理解します。MATLAB は、同じ名前で設定された複数の関数の中から選択に、ディスパッチタイプを使います。しかし、引数の数は考慮しません。

関数の優先順位

関数の優先順位は、関数のタイプと MATLAB パス上での位置をベースにして、ある関数が他の関数よりも高い順位であるかの決定ができます。メソッド選択の全体を見ると、MATLAB は二つのタイプの関数を含んでいます。すなわち、MATLAB の中に組み込まれたものと、M- ファイルで記述されたものです。MATLAB は、関数の優先順位を決定するときに、これら二つのタイプを別々に取り扱います。

MATLAB は、つぎの関数優先則を適用することにより、与えられた内容に対する正しい関数を選択します。

組み込み関数：

1 多重定義されたメソッド

   MATLAB 関数と同じ名前をもつ dispatch 引数のクラスディレクトリの中にメソッドが存在する場合、このメソッドは、MATLAB 関数の代わりに呼び込まれます。

2 多重定義されていない MATLAB 関数

   多重定義されていないメソッドの場合、MATLAB 組み込み関数が呼び出されます。

MATLAB の組み込み関数は、サブ関数やプライベート関数よりも優先順位が高いものです。そのため、MATLAB 組み込み関数と同名のサブ関数やプライベート関数は、呼び出されません。

非組み込み関数：

1 サブ関数

サブ関数は、パス上にある同じ名前のすべての M- ファイル関数や多重定義されたメソッドよりも優先順位が高いものです。関数が多重定義されたメソッドと一致する引数のタイプを使って呼び出されても、MATLAB はサブ関数を使い、多重定義メソッドを無視します。

2 プライベート関数

プライベート関数は、カレントの範囲内に同名のサブ関数がない場合に呼び出されます。サブ関数と同様に関数が多重定義されたメソッドと一致する引数のタイプを使って呼び出されても、MATLAB はプライベート関数を使い、多重定義メソッドを無視します。

3 クラスコンストラクタ関数

コンストラクタ関数 (@ ディレクトリ、たとえば、@polynom/polynom.m) と同じ名前をもつ関数) は、他の MATLAB 関数より優先順位が上です。そのため、polynom.m と呼ばれる M- ファイルを作成し、コンストラクタ @polynom/polynom.m バージョンの前になるようにパスを設定すると、MATLAB は、コンストラクタバージョンを呼びます。

4 多重定義されたメソッド

MATLAB は、サブ関数、または、プライベート関数でマスクされていなければ、多重定義されたメソッドと言います。

5 カレントディレクトリ

カレントワーキングディレクトリの中の関数は、パス上のある位置の前に選択されます。

6 パス上の位置

最終的に、あるパス上の任意の関数が選択されます。

複数のディレクトリからの選択

同じ名前をもったメソッドを含むパス上にいくつかのディレクトリがあります。MATLAB は、パス上にメソッドの最初の実現が検出された場合に、実現のタイプ (MEX- ファイル、P- コード、M- ファイル) に拘わらず、探索を終了します。

複数の実現タイプからの選択

四つのファイル優先タイプがあります。MATLAB は同じディレクトリ内に同じ名前をもつ関数間で選択する場合、ファイル優先を選択します。ファイルタイプに対する優先順位は、

1. MEX- ファイル
2. MDL- ファイル (Simulink モデル)
3. P- コード
4. M- ファイル

たとえば、MATLAB が、あるクラスの中で、あるメソッドの P- コードと M- ファイルバージョンを探索すると、P- コードバージョンが使われます。そのため、ユーザが M- ファイルがエディトすると、P- コードを再度作成することが重要です。

MATLAB が読み込みメソッドの引用

MATLAB は、コマンド which を使ってどのメソッドを呼ぶかを決定できます。たとえば、

```
which pie3
your_matlab_path/toolbox/matlab/specgraph/pie3.m
```

となりますが、また、p がポートフォリオオブジェクトの場合、

```
which pie3(p)
dir_on_your_path/@portfolio/pie3.m % portfolio method
```

コマンド which は、入力引数としてポートフォリオオブジェクトを渡す場合、MATLAB がどの pie3 バージョンを使うかを決めます。ユーザの MATLAB パス上に存在する設定した関数のすべてのバージョンの一覧は、-all オプションを使ってください。このコマンドに対する詳細は、which のリファレンスページを参照してください。

A

エクスターナルインタフェースとMATLAB API

オンラインヘルプでのドキュメントの検索 A-2

リファレンスドキュメント A-4

オンラインヘルプでのドキュメントの検索

MATLAB は、MATLAB と以下のプログラムおよびデバイスとの通信を行えるインタフェースを提供します。

- 外部の C および Fortran プログラム
- Java や ActiveX のようなオブジェクト指向のテクノロジ
- コンピュータのシリアルポート上のハードウェアデバイス

MATLAB からデータを読み込んだり、MATLAB にデータを書き込むことができます。

これらのインタフェースは、MATLAB Application Program Interface (API) としても参照され、オンラインヘルプに詳しく説明されています。以下のパスを使ってヘルプの位置を探してみてください。

```
MATLAB -> Using MATLAB -> External Interfaces/API
```

MATLAB からの C および Fortran プログラムの呼び出し

MATLAB は、MATLAB 環境の外部のデータがプログラムと相互に機能する C および Fortran 言語で書かれた外部プログラムとのインタフェースを提供します。この節では、MATLAB から、組み込み関数のように C や Fortran サブルーチンを呼び出す方法を説明します。

C 言語 MEX- ファイルの作成

MATLAB の呼び出し可能な C および Fortran プログラムは、MEX- ファイルと呼ばれます。この節では、MEX- ファイルの作成および取り扱い方法を説明します。

Fortran MEX- ファイルの作成

この節では、Fortran MEX- ファイルの作成および取り扱い方法を説明します。

C および Fortran プログラムからの MATLAB の呼び出し

MATLAB は、C および Fortran プログラムからの呼び出しに応答する計算エンジンとして使うことができます。この節では、以下を行うことができる MATLAB 関数について説明します。

- MATLAB プロセスの起動と終了
- コマンドを送信し、MATLAB とデータをやりとりする
- MATLAB エンジンプログラムをコンパイル、リンクする

MATLAB からの Java の呼び出し

この節では、MATLAB と Java クラスおよびオブジェクトとのインタフェースの使用法を説明します。この機能を使って、以下を行うことができます。

- Java クラスを MATLAB 環境に導入
- それらのクラスからオブジェクトを作成
- MATLAB での Java 配列の操作
- Java オブジェクトについてメソッドを呼び出し、MATLAB または Java データタイプを渡す

データの読み込みと書き込み

この節では、MATLAB 環境にデータを読み込んだり、MATLAB 環境からデータを書き込むための MAT- ファイルの使用法を説明します。MAT- ファイルは、異なるプラットフォーム間で MATLAB データを移動するための非常に簡単な方法を提供する便利なメカニズムです。さらに、他のスタンドアロン MATLAB アプリケーションとデータのやりとりを行うための手段も提供します。

ActiveX と DDE のサポート

MATLAB は、ActiveX および Dynamic Data Exchange(DDE) との相互作用を行うことができるインタフェースを提供します。この節では、以下の方法を説明します。

- ActiveX コントロールコンポーネントと MATLAB figure ウィンドウのような ActiveX コントロールコンテナとの統合
- ActiveX Automation コンポーネントを使った MATLAB の制御
- DDE を使って MATLAB と他の Windows アプリケーションとのアクセスを可能にする

シリアルポート I/O

この節では、コンピュータのシリアルポートに接続するモデム、プリンタ、科学機器のような周辺機器への直接アクセスを提供する MATLAB シリアルポート通信を説明します。このインタフェースは、以下を実行可能なシリアルポートオブジェクトによって確立されます。

- シリアルポート通信の設定
- シリアルポート制御ピンの利用
- デバイスへのデータの書き込みとデバイスからの読み込み
- 特定のイベントの発生時にアクションを実行
- シリアルポートセッションの記録を作成

リファレンスドキュメント

オンラインヘルプのリファレンス部は、MATLAB エクスターナルインタフェースで利用可能な MATLAB 関数の詳しい説明を提供します。このドキュメントは、以下のパスにあるオンラインヘルプのコンテンツで見つけることができます。ページ External Interfaces/API Reference をクリックすると、下記のセクションへのリンクが表示されます。

```
MATLAB -> Reference -> External Interfaces/API Reference
```

API Notes
mex スクリプトの利用、MATLAB mxArray とその他のデータタイプ、Fortran でのポインタの渡し方の紹介

C エンジンルーチン
ユーザの C プログラムから MATLAB を呼び出し可能な関数

C MAT- ファイルルーチン
ユーザの C プログラムに MATLAB データを組み込み、利用可能な関数

C MEX-Functions
MATLAB 環境で操作を行うために C ファイル内で利用する関数

C MX-Functions
MATLAB 配列を操作するため、ユーザの C ファイル内で利用する配列アクセス関数および配列作成関数

Fortran エンジンルーチン
ユーザの Fortran プログラムから MATLAB を呼び出し可能な関数

Fortran MAT- ファイルルーチン
ユーザの Fotraran プログラムに MATLAB データを組み込み、利用可能な関数

Fortran MEX-Functions
MATLAB 環境で操作を行うために、ユーザの Fortran ファイル内で利用する関数

Fortran MX-Functions
MATLAB 配列を操作するため、ユーザの Fotraran ファイル内で利用する配列アクセス関数および配列作成関数

Java インタフェース関数
MATLAB から Java クラスおよびオブジェクトを作成、操作することが可能な関数

ActiveX 関数
ActiveX オブジェクトを作成し、それらのインタフェースを操作する関数

DDE 関数

MATLAB とその他の Windows アプリケーションとのアクセスを行う Dynamic Data Exchange 関数

シリアルポート I/O 関数
コンピュータのシリアルポートに接続されているデバイスと通信を行う関数