目　次

付録 A
R を始める、ヘルプを見る

はじめに

この章は、他の章の基礎です。R のダウンロード方法、インストール方法、実行方法について説明します。

それだけではなく、あなたの疑問に対する答えを見つける方法についても説明します。R コミュニティでは豊富なドキュメントとヘルプを用意しています。あなたの仲間は大勢います。以下はよく使われている役に立つ情報源です。

ローカルにインストールされたドキュメント

R を自分の PC にインストールすると大量のドキュメントも一緒にインストールされます。ローカルドキュメントはもちろん読めますし（**レシピ A.6**）、検索もできます（**レシピ A.8**）。私はすでにインストールされているドキュメントを、わざわざ Web で探した経験が何度もあります。

タスクビュー

タスクビューは、計量経済学、医療画像処理、心理測定学、空間統計学といった統計処理の分野に特化されたパッケージのことです。タスクビューはそれぞれ分野の専門家によって開発され、メンテナンスされています。現在 28 のタスクビューがあり、中には読者の関心がある分野もあるでしょう。私は少なくとも、すべての初心者が 1 つのタスクビューを見つけて、一読することをお勧めします。R の可能性について感覚的にわかってもらうためです（**レシピ A.11**）。

パッケージドキュメント

ほとんどのパッケージは便利なドキュメントを用意しています。また多くは概要と、R コミュニティでは「ビニエット」と呼ばれるチュートリアルも用意しています。このようなドキュメントは、CRAN（http://cran.r-project.org/）のようなパッケージレポジトリからパッケージと一緒に利用でき、パッケージをインストールするとドキュメントも一緒にインストールされます。

メーリングリスト

ボランティアたちが多くの時間を費やして、R メーリングリストに投稿された初心者の質問に親切に答えてくれます。アーカイブを検索して、疑問の答えを探すこともできます（**レシピ A.12**）。

Q&A サイト

誰でも Q&A サイトに質問を投稿できます。質問には知識豊富な人たちが答えを返してくれます。その答えは人気投票できるようになっていて、ベストアンサーが自動的にわかるようになります。すべての情報はタグ付けされ、検索用にアーカイブされています。このようなサイトは、メーリングリストとソーシャルネットワークの中間のようなもので、Stack Overflow（http://stackoverflow.com/）などが代表的です。

Web

Web には R に関する情報がたくさんあり、検索用の R 専用ツールもあります（**レシピ A.10**）。Web の情報は随時更新されるので、新しいもの、まとめられた改善された方法、検索情報に目を光らせるようにします。

レシピ A.1 R のダウンロードとインストール

問題

R を自分の PC にインストールしたい。

解決策

Windows ユーザと OS X ユーザは CRAN（Comprehensive R Archive Network）からダウンロードします。Linux ユーザと Unix ユーザはパッケージ管理ツールを使って R パッケージをインストールします。

Windows

1. ブラウザで http://www.r-project.org/ を開く。
2. 「CRAN」をクリックすると、ミラーサイトの一覧が国別に表示される。
3. 近場のサイトを選択する。
4. 「Download and Install R」の「Windows」をクリックする。
5. 「base」をクリックする。
6. R の最新版のダウンロードリンク（.exe ファイル）をクリックする。
7. ダウンロードが完了したら、.exe ファイルをダブルクリックしてお決まりの質問に答える。

OS X

1. ブラウザで http://www.r-project.org/ を開く。
2. 「CRAN」をクリックすると、ミラーサイトの一覧が国別に表示される。
3. 近場のサイトを選択する。

4. 「MacOS X」をクリックする。
5. R の最新版の「Files:」をクリックして .pkg ファイルをインストールする。
6. ダウンロードが完了したら、.pkg ファイルをダブルクリックしてお決まりの質問に答える。

Linux または Unix

主な Linux ディストリビューションは R インストール用のパッケージを用意しています。以下に例を挙げます。

ディストリビューション	パッケージ名
Ubuntu または Debian	r-base
Red Hat または Fedora	R.i386
Suse	R-base

システムのパッケージマネージャを使って R パッケージをダウンロードし、インストールします。通常は root パスワードか sudo 権限が必要になります。それ以外の場合は、システム管理者に問い合わせてください。

解説

R を Windows や OS X にインストールする場合は、ビルド済みのバイナリが用意されているため、特に難しいことはありません。インストール画面の指示に従うだけです。CRAN の Web ページは、FAQ や特殊な場合のヒント（Windows Vista に R をインストールする方法など）といったインストール関連のリソースもあり、重宝します。

Linux や Unix にインストールする方法は、実は 2 通りあります。ディストリビューションパッケージをインストールする方法と、スクラッチからビルドする方法です。実際には、パッケージをインストールするほうがおすすめです。一番最初にインストールする場合も、その後アップデートするときも、ディストリビューションパッケージを使うと効率的だからです。

Ubuntu や Debian では、apt-get を使って R のダウンロードとインストールを行います。sudo 権限で実行します。

```
$ sudo apt-get install r-base
```

Red Hat や Fedora では yum を使います。

```
$ sudo yum install R.i386
```

ほとんどのプラットフォームは、さらに使いやすいグラフィカルパッケージマネージャを用意しています。

標準パッケージの他に、ドキュメントパッケージのインストールもお勧めします。例えば、私は自分の

Ubuntu マシンに、r-base-html をインストールしていますし、またハイパーリンクの付いたドキュメントを読むのが好きなので、r-doc-html もインストールして、重要な R のマニュアルを手元の PC に置いています。

```
$ sudo apt-get install r-base-html r-doc-html
```

Linux のレポジトリの中には、CRAN で利用できる R パッケージのビルド済みのコピーを置いているものもあります。しかし CRAN から直接最新バージョンのものを使いたいので、私は使いません。

あまり一般的ではありませんが、スクラッチから R をビルドする必要もあるでしょう。バージョンがはっきりしない、またはサポートされていないバージョンの Unix を使っていたり、性能や設定に関して特別な配慮をしたりする場合です。Linux や Unix でビルドを行う手続きは、標準的です。CRAN のミラーサイトから tar ボールをダウンロードします。例えば、R-2.12.1.tar.gz のような名前のもので、2.12.1 は最新バージョンの番号に置き換えられます。tar ボールを展開して、INSTALL という名前のファイルを探し、指示に従います。

関連項目

『R in a Nutshell』（O'Reilly）には、ダウンロードとインストールに関してさらに詳しい説明があります。また Windows と OS X でビルドする手順についても説明があります。決定版はおそらく「R Installation and Administration」（http://cran.r-project.org/doc/manuals/R-admin.html）というタイトルの、CRAN から利用できるガイドでしょう。さまざまなプラットフォーム上で R をビルド／インストールする方法を詳しく説明しています。

このレシピは標準パッケージのインストールについて述べています。CRAN から追加パッケージをインストールするには、**レシピ C.9** を参照してください。

A.2 R の起動

問題

自分の PC で R を実行したい。

解決策

Windows

「スタート」→「すべてのプログラム」→「R」をクリックするか、デスクトップ上の R のアイコンをクリックします（インストールするアイコンが自動的に生成されます）。

OS X

アプリケーションディレクトリにあるアイコンをクリックするか、R アイコンをドックに置き、イコンをクリックします。シェルの Unix コマンドラインで R と入力するだけでも起動できます。

Linux または Unix

シェルプロンプトからRコマンドを使ってRプログラムを起動します（Rは大文字です）。

解説

起動方法は使用するプラットフォームに依存します。

Windows での起動

Rを起動すると、新しいウィンドウが開きます。ウィンドウはRコンソールと呼ばれるテキスト領域があり、Rの式を入力します（**図 A-1**）。

Windows のスタートメニューには不思議なところがあります。新しいバージョンにアップグレードすると、スタートメニューは必ず新しいバージョンだけでなくすべての古いバージョンも表示されます。そのため、アップグレードすると、「R 2.8.1」「R 2.9.1」「R 2.10.1」といった複数のRのバージョンから選べるようになるので、最新のものを選択するようにします（不要なものを減らしたいので、古いバージョンンはアンインストールしたいと思うかもしれません）。

スタートメニューは使いにくいので、別の2つの方法を提案します。デスクトップにショートカットを作るか、.RData ファイルをダブルクリックするかのどちらかを私はお勧めします。

インストールすると、デスクトップアイコンが作成されたと思います。もしアイコンができていなかったら、ショートカットを作れば簡単です。スタートメニューからRプログラムを選択しますが、左クリックでRを実行するのではなく、プログラム名の上でマウスの右ボタンを押し、プログラム名をデスクトップ

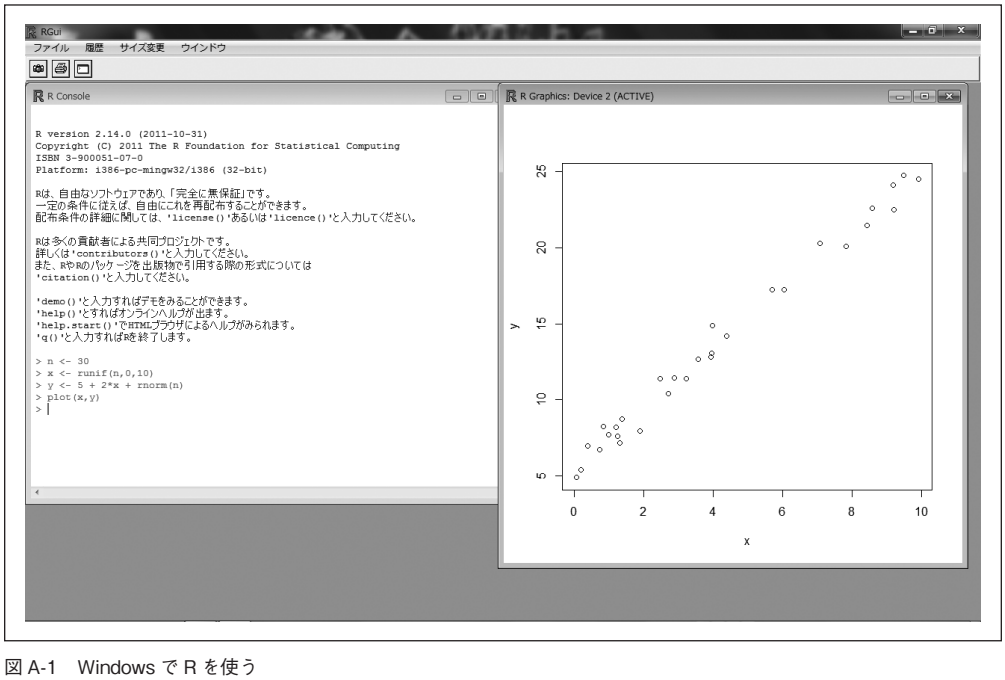


図 A-1 Windows で R を使う

にドラッグして、マウスのボタンを離します。するとWindowsは「ここにコピー」「ここに移動」と聞いてくるので、「ここにコピー」を選びます。これでデスクトップにショートカットが現れます。

作業ディレクトリにある.RDataファイルをダブルクリックしてもRは起動できます。.RDataファイルは、Rがワークスペースを保存するために生成するファイルです。ディレクトリを作成したら、Rを起動してそのディレクトリに変更します。ワークスペースをそこに移動し、「終了」または`save.image`関数を使ってワークスペースをそこに保存します。そうすると、.RDataファイルが作られます。その後、Windows Explorerでディレクトリを開き、.Rdataファイルをダブルクリックするだけでが起動します。

多分、WindowsでRを使う上で最も戸惑うのは、次の一問に集約されているでしょう。「Rをスタートしたときの作業ディレクトリはどこ？」。答えは、もちろん「場合によります」です。

- Rをスタートメニューから起動した場合、作業ディレクトリは普通は、C:\Documents and Settings\<ユーザ名>\My Documents（Windows XP）またはC:\Users\<ユーザ名>\Documents（Windows Vista、Windows 7）となります。環境変数`R_USER`を設定すると、作業ディレクトリを変更できます。
- デスクトップのショートカットからRを起動した場合、代替となるスタートアップディレクトリを指定でき、それがRを起動したときの作業ディレクトリになります。代替ディレクトリを指定するには、ショートカットを右クリックし、プロパティを選んで、「作業フォルダ」と書かれたボックスにディレクトリパスを入力し、OKをクリックします。
- .RDataファイルをクリックしてRを起動する方法が一番手軽にこの小さな問題を解決します。Rは自動的に作業ディレクトリを.RDataファイルのあるディレクトリに変更します。ユーザはたいてい.Rdataファイルのディレクトリが作業ディレクトリとなるようにしたいはずです。

どんな場合でも、`getwd`関数を使うと現在の作業ディレクトリが確認できます（**レシピ C.1**）。

記録するだけならば、WindowsにはRterm.exeと呼ばれるコンソールバージョンのRもあります。Rインストールファイルのbinサブディレクトリに入っています。GUIバージョンのほうが使いやすいので、私は使ったことがありません。Windowsスケジューラからジョブを走らせるようなバッチ（インタラクティブではない）処理を使うことをお勧めします。この本では、読者はコンソールバージョンではなく、GUIバージョンのRを使っていることを前提としています。

OS Xでの起動

アプリケーションフォルダ中のRアイコンをクリックしてRを実行します。頻繁にRを使うようでしたら、アイコンをドックに移動しておくとよいでしょう。すると、GUIバージョンのRが起動します。コンソールバージョンよりもGUIのほうがいくらか便利です。GUIバージョンは起動すると最初にホームディレクトリである作業ディレクトリを表示します。

OS Xではシェルプロンプトで`R`と入力すれば、コンソールバージョンが使えます。

Linux と Unix での起動

Unix シェルプロンプトでプログラム名の R を入力するだけでコンソールバージョンが起動します。小文字の r ではなく、大文字の R を入力するように注意します。

R プログラムは戸惑うほど多くのコマンドラインオプションを持っています。--help オプションを使って、その一覧のすべてを確認してみてください。

関連項目

レシピ A.4 では R の終了方法を、**レシピ C.1** では現在の作業ディレクトリについての詳細を、**レシピ C.2** ではワークスペースの保存の仕方について、**レシピ C.11** では、起動メッセージを隠す方法を説明しています。詳しくは『R in a Nutshell』の 2 章を参照してください。

レシピ A.3　コマンドの入力

問題

R を起動させ、コマンドプロンプトが表示された。次に何をすればよいか？

解決策

コマンドプロンプトに式を入力すればいいだけです。R は入力された式を評価し、結果を表示します。入力支援のコマンドライン編集が使えます。

解説

R のプロンプトは「>」です。まず、R を大きな電卓のように使ってみましょう。式を入力すると、R は式を評価して結果を表示します。

```
> 1+1
[1] 2
```

コンピュータは 1 と 1 を足して 2 を求め、結果を表示します。

2 の前の [1] は、何だろうと思うでしょう。R にとっては、たとえ 1 つしか要素がなくても結果はベクトルです。R は値に [1] というラベルを付け、これがベクトルの第 1 番目の要素であることを示します。ベクトルの唯一の要素なので、当然ですね。

R は完全な式が入力されるまで、プロンプトを表示します。式 max(1,3,5) は完全な式なので、R は入力の読み込みをやめて評価します。

```
> max(1,3,5)
[1] 5
```

一方、「max(1,3,」は不完全な式なので、R プロンプトは、更なる入力を促します。プロンプトは、大なり記号（>）からプラス記号（+）に変わり、R が更なる入力を促していることがわかります。

```
> max(1,3,
+ 5)
[1] 5
```

入力は間違いやすく、再入力は面倒でイライラするものです。そのため、人生をラクにするために R はコマンドライン編集を用意しています。コマンドライン編集はたった 1 つのキーストロークで、コマンドの呼び出し、修正、再実行を可能にします。私独自のコマンドライン編集の使い方は次のような感じです。

1. スペルミスのある R の式を入力する。
2. R が間違いだと文句を言う。
3. ↑ キーを押して、スペルミスのある行を呼び出す。
4. → キー、← キーを使ってカーソルをエラーの箇所に移動。
5. Delete キーで間違いの文字を削除。
6. 正しい文字をコマンドラインに入力する。
7. Enter キーを押して正しいコマンドを再実行する。

これはほんの一例です。R は**表 A-1** に挙げるような呼び出しや編集に関する通常のキーストロークもサポートしています。

表 A-1 コマンドライン編集のキーストローク

キー	Ctrl キーの組合せ	効果
↑	Ctrl-P	以前実行したコマンドの呼び出し
↓	Ctrl-N	1 つ次のコマンド履歴に移動
バックスペース	Ctrl-H	カーソルの左側の文字を削除
削除（Del）	Ctrl-D	カーソルの右側の文字を削除
Home	Ctrl-A	カーソルを行の先頭に移動
End	Ctrl-E	カーソルを行の末尾に移動
→	Ctrl+F	カーソルを 1 文字右に（先に）移動
←	Ctrl-B	カーソルを 1 文字左に（前に）移動
	Ctrl-K	カーソル位置から行末までを削除
	Ctrl-U	行をすべて削除しはじめからやり直す
タブ		名前の補完（プラットフォームによってはできない）

Windows と OS X では、マウスを使ってコマンドをハイライトさせ、通常の方法でコピーしてから新しいコマンドラインにペーストすることもできます。

関連項目

レシピ B.13 を参照。Windows 版のメインメニューから「ヘルプ」→「コンソール」を選択すると、コマンドライン編集に便利なキーストロークの一覧が表示されます。

レシピ A.4　R の終了

問題

R を終了したい。

解決策

Windows

メインメニューからファイル→終了を選びます。またはウィンドウフレームの右上の赤い X をクリックします。

OS X

Command-q（アップルマーク -q）を押下します。またはウィンドウフレームの左上の赤い X をクリックします†。

Linux または Unix

コマンドプロンプトで Ctrl-D を押下します。

どのプラットフォームでもプログラムの終了には q 関数（quit の q）を使います。

```
> q()
```

括弧の中は空ですが、関数を呼び出すために必要です。

解説

終了の際、必ず R にワークスペースを保存するか尋ねられます。次の 3 つの選択肢があります。

- ワークスペースを保存して終了。
- ワークスペースは保存しないが、とにかく終了。
- 終了をキャンセルしてコマンドプロンプトに戻る。

ワークスペースを保存すると、R はワークスペースの内容を現在の作業ディレクトリの .RData というファイルに書き込みます。すでに保存されていた前のワークスペースの内容は上書きされます。ワークスペースを変更したくなければ、保存してはいけません（例えば、誤ってクリティカルデータを消去してし

† 訳注：アップルマークは、古い Macintosh のキーボードのみです。

まったときなどです)。

関連項目

現在の作業ディレクトリについては**レシピ C.1** を、ワークスペースの保存については**レシピ C.2** を参照。『R in a Nutshell』の 2 章を参照。

レシピ A.5　R の中断

問題

長時間の計算を中断して、R を終了せずにコマンドプロンプトに戻りたい。

解決策

Windows または OS X

Esc キーまたはメニューバーの「STOP」マークの赤いアイコンをクリックする。

Linux または Unix

Ctrl-C を押下すると、R を終了させずに中断する。

解説

R の中断は、変数を途中の状態のままにしておくことになります。そしてその状態は、どのくらい計算処理が進んだのかに依存します。中断のあと、ワークスペースを確認してください。

関連項目

レシピ A.4 を参照。

レシピ A.6　提供されたドキュメントを読む

問題

R 用のドキュメントを読みたい。

解決策

`help.start` 関数を使ってドキュメントの目次を確認します。

```
> help.start()
```

ここではインストールされたドキュメントのリンクが表示されます。

解説

R の標準ディストリビューションは豊富なドキュメントを用意しています。文字通り何千ページにもなります。追加パッケージをインストールするとそのパッケージのドキュメントもインストールされます。

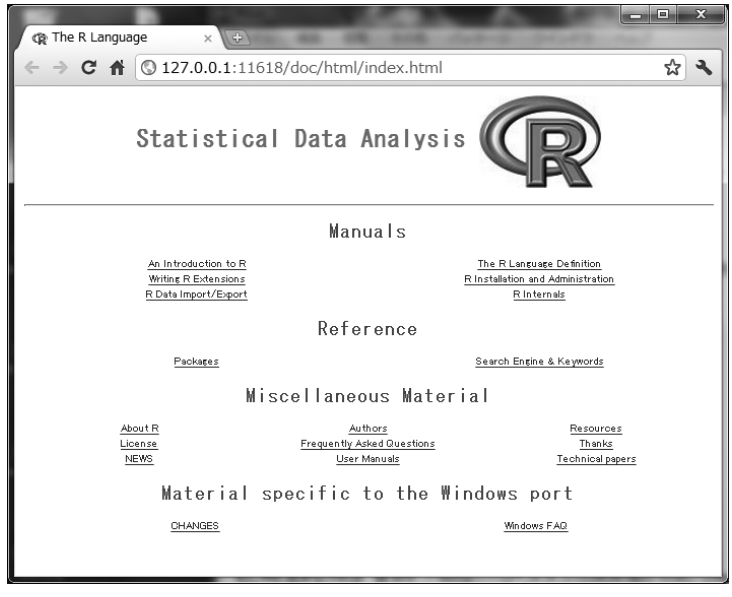


図 A-2　ドキュメントの目次

help.start 関数を使えば、**図 A-2** のように目次のトップの項目のウィンドウが開くので、簡単にドキュメントを読むことができます。

以下のリファレンスセクションの 2 つのリンクが特に便利です。

Packages（パッケージ）
: クリックすると、インストールされたすべてのパッケージの一覧が表示されます。標準ディストリビューションと追加パッケージの両方ともです。パッケージ名をクリックすると、そのパッケージの関数とデータセットの一覧が表示されます。

Search Engine & Keywords（検索エンジンとキーワード）
: ここをクリックすると、簡易検索エンジンにアクセスできます。簡易検索エンジンでは、キーワードまたはフレーズでドキュメントを検索できます。ここにはトピックごとに分類された一般的なキーワードのリストもあります。その 1 つをクリックすれば関連するページが読めます。

関連項目

ローカルドキュメントは R プロジェクト Web サイト（http://www.r-project.org）からコピーしたものです。Web サイトのドキュメントは更新されているかもしれません。

レシピ A.7　関数のヘルプを入手する

問題

インストールされているある関数についての詳細を知りたい。

解決策

help でその関数に関するドキュメントを表示します。

```
> help(functionname)
```

args でその関数の引数に関する情報を確認します。

```
> args(functionname)
```

example でその関数の使用例を確認します。

```
> example(functionname)
```

解説

私はこの本でたくさんの R 関数について説明しています。しかし、どの関数についても、書き切れなかった注意やヒントがたくさんあります。興味がある関数があったら、その関数のヘルプページを読むことを強く勧めます。注意は読者にとってとても役に立つかもしれません。

例えば、mean 関数についてもっと知りたくなったとしたら、help 関数を次のように使います。

```
> help(mean)
```

こうすると、ウィンドウが開いて関数のドキュメントが表示されるか、またはコンソールにドキュメントが表示されます（プラットフォームに依存します）。入力を簡単にするために、ショートカットを使いたい場合は、次のようにします。

```
> ?mean
```

ある関数の引数が何だったか、どういう順番だったかについて思い出したいこともあるでしょう。その場合は次のようにします。

```
> args(mean)
function (x, ...)
NULL
> args(sd)
function (x, na.rm = FALSE)
NULL
```

arg による出力の 1 行目は、関数呼び出しの概要です。mean の概要は、引数は数値ベクトルの x が 1 つ、sd については、同じくベクトルの x と na.rm と呼ばれるオプション引数があることを示しています（NULL と出力されることが多い 2 行目は無視してください）。

関数の多くのドキュメントは、その終わり近くに使用例を載せています。R には、その使用例を実行して、関数の能力についてちょっとしたデモを紹介するという優れた機能があります。例えば mean 関数のドキュメントにも使用例が載っていますが、それをわざわざ入力し直す必要はありません。example を使うだけで実行できます。

```
> example(mean)

mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.1))
[1] 8.75 5.50

mean> mean(USArrests, trim = 0.2)
  Murder  Assault UrbanPop     Rape
    7.42   167.60    66.20    20.16
```

ただ example(mean) と入力するだけで、ヘルプページの例を実行して、結果を表示する、ということを R がすべてやってくれます。

関連項目

関数の検索は**レシピ A.8** を、サーチパスについては、**レシピ C.5** を参照してください。

レシピ A.8　提供されているドキュメントを検索する

問題

インストールされた関数の詳細を知りたいが、help 関数ではそのような関数のドキュメントは見つからないと表示される。または、キーワードでインストール済みのドキュメントを検索したい。

解決策

help.search を使って、自分の PC 上の R ドキュメントを検索します。

```
> help.search("pattern")
```

pattern に入るのは関数名やキーワードです。必ず引用符でくくります。

もっと簡単に 2 つの疑問符を頭につけて検索することもできます（この場合は引用符は必要ありません）。

```
> ??pattern
```

解説

help で関数の検索を探しても、何も見つからないと言われることがあるかもしれません。

```
> help(adf.test)
No documentation for 'adf.test' in specified packages and libraries:
you could try 'help.search("adf.test")'
```

関数が実際にインストールされているのに、このように表示されるとイライラするかもしれません。これは、関数パッケージが現在読み込まれていないことが原因です。そして、読者はどのパッケージにこの関数が入っているのかわかりません。これはある種のジレンマです。現在はサーチパスにそのパッケージがないため、R がヘルプファイルを見つけることができないことを示しています。詳細は**レシピ C.5** を参照してください。

インストールされたパッケージをすべて検索すれば解決します。エラーメッセージの中で勧められたように help.search を使います。

```
> help.search("adf.test")
```

こうすると、この関数を含むすべてのパッケージの一覧が表示されます†。

```
Help files with alias or concept or title matching 'adf.test' using
regular expression matching:

tseries::adf.test       Augmented Dickey-Fuller Test

Type '?PKG::FOO' to inspect entry 'PKG::FOO TITLE'.
```

例えば次の出力は、adf.test 関数が tseries パッケージにあることを示しています。今度は help に明示的に関数を含むパッケージなのか指定することで、ドキュメントを読むことができます。

```
> help(adf.test, package="tseries")
```

この他にも、tseries パッケージを検索リストに入れて、独自の help コマンドを繰り返しても、同様に関数を探してドキュメントを表示してくれます。

キーワードを使って、幅広く検索することもできます。R は指定したキーワードを含むすべてのドキュメントを探してくれます。例えば拡張ディッキー＝フラー（ADF）検定に関連する関数をすべて見つけたい場合は次のようにします。

```
> help.search("dickey-fuller")
```

† 訳注：プラットフォームによって動作は異なります。

私は拡張ディッキー＝フラー（ADF）検定を実装する 2 つの追加パッケージ（fUnitRoots と urca）をインストールしていたので、私の PC では、結果はこのようになります。

```
Help files with alias or concept or title matching 'dickey-fuller' using
fuzzy matching:

fUnitRoots::DickeyFullerPValues
                        Dickey-Fuller p Values
tseries::adf.test       Augmented Dickey-Fuller Test
urca::ur.df             Augmented-Dickey-Fuller Unit Root Test

Type '?PKG::FOO' to inspect entry 'PKG::FOO TITLE'.
```

関連項目

ドキュメントブラウザ経由でローカルの検索エンジンにもアクセスできます。詳細は**レシピ A.6** を参照してください。サーチパスの詳細は**レシピ C.5** を、関数のヘルプを見る方法については**レシピ 1.4** を参照してください。

レシピ A.9　パッケージのヘルプを見る

問題

PC にインストールしたパッケージについて詳しく知りたい。

解決策

help 関数を使って（関数名ではなく）パッケージ名（*packagename*）を指定します。

```
> help(package="packagename")
```

解説

パッケージ（関数とデータセット）の内容を知りたいこともあるでしょう。新しいパッケージをダウンロード／インストールした後は特にこの必要性が高いでしょう。help 関数にパッケージ名を指定すると、内容だけでなく、他の情報も知ることができます。

このヘルプの呼び出しは標準ディストリビューションの標準的パッケージ、tseries パッケージの情報を表示します。

```
> help(package="tseries")
```

パッケージ情報はまず概要で始まり、次に関数とデータセットのインデックスが続きます。私の PC では、最初の数行はこのようになっています。

```
                Information on package 'tseries'
Description:

Package:            tseries
Version:            0.10-22
Date:               2009-11-22
Title:              Time series analysis and computational finance
Author:             Compiled by Adrian Trapletti
                    <a.trapletti@swissonline.ch>
Maintainer:         Kurt Hornik <Kurt.Hornik@R-project.org>
Description:        Package for time series analysis and computational
                    finance
Depends:            R (>= 2.4.0), quadprog, stats, zoo
Suggests:           its
Imports:            graphics, stats, utils
License:            GPL-2
Packaged:           2009-11-22 19:03:45 UTC; hornik
Repository:         CRAN
Date/Publication:   2009-11-22 19:06:50
Built:              R 2.10.0; i386-pc-mingw32; 2009-12-01 19:32:47 UTC;
                    windows

Index:
NelPlo                  Nelson-Plosser Macroeconomic Time Series
USeconomic              U.S. Economic Variables
adf.test                Augmented Dickey-Fuller Test
arma                    Fit ARMA Models to Time Series
.
. (etc.)
.
```

パッケージの中には、ビニエットを含むものもあります。ビニエットはその概要、チュートリアル、参照カードのような追加ドキュメントです。これらもパッケージドキュメントの一部としてインストールされたものです。パッケージのヘルプページの最後のほうには、ビニエットの一覧があります。

PC 上のビニエットの一覧を確認するには、vignette 関数を使います。

```
> vignette()
```

ある特定のパッケージに含まれているビニエットの名前を確認したいときは、次のようにします。

```
> vignette(package="packagename")
```

すべてのビニエットには名前がついていますので、名前からもビニエットを探せます。

```
> vignette("vignettename")
```

関連項目

パッケージ内のある特定の関数のヘルプは**レシピ A.7** を参照してください。

レシピ A.10　Web 上のヘルプを探す

問題

R に関する情報と回答を Web 上で探したい。

解決策

R 内では、RSiteSearch 関数を使ってキーワードやフレーズから検索します。

```
> RSiteSearch("key phrase")
```

Web ブラウザでは、次のサイトから検索してみてください。

http://rseek.org
: R 専用の Web サイトだけを対象とする Google のカスタムサーチです。

http://stackoverflow.com/
: Stack Overflow は検索可能な Q&A サイトです。データ構造、コーディング、グラフィックスといったプログラミングの問題を多く取り扱います。

http://stats.stackexchange.com/
: Stack Exchange の統計解析の分野も検索可能な Q&A サイトですが、プログラミングよりも統計の色合いが濃くなっています。

解説

RSiteSearch 関数は、ブラウザウィンドウを開き、R プロジェクト Web サイトの検索エンジンに直接アクセスします（http://search.r-project.org/）。すると、絞り込み検索ができるような検索の初期画面が表示されます。例えば、次のようにすると、「canonical correlation（正準相関）」の検索が始まります。

```
> RSiteSearch("canonical correlation")
```

RSiteSearch は R を離れずに Web の簡易検索ができるのでとても便利ですが、検索範囲が R のドキュメントとメーリングリストのアーカイブに限られます。

rseek.org サイトは、もっと幅広い対象を検索できます。このサイトのよいところは、R 関連のサイトに対象を絞って Google の機能を利用できるところです。通常の Google の検索の結果から無関係なものを取

り除いてくれます。rseek.org の美しさは、結果を使いやすい方法に整理してくれるところにあります。

図 A-3a は rseek.org サイトにアクセスして、「canonical correlation」を調べた結果です†。ページの左側は、R サイトを検索した一般的な結果です。右側は、検索結果をカテゴリ別に構成し、タブを付けて表示しています。

- Introductions（はじめに）
- Task View（タスクビュー）
- Support Lists（サポート一覧）
- Functions（関数）
- Books（書籍）
- Blogs（ブログ）
- Related Tools（関連ツール）

例えば、「Introduction」タブをクリックすると、チュートリアルが表示されます。「Task Views」タブで

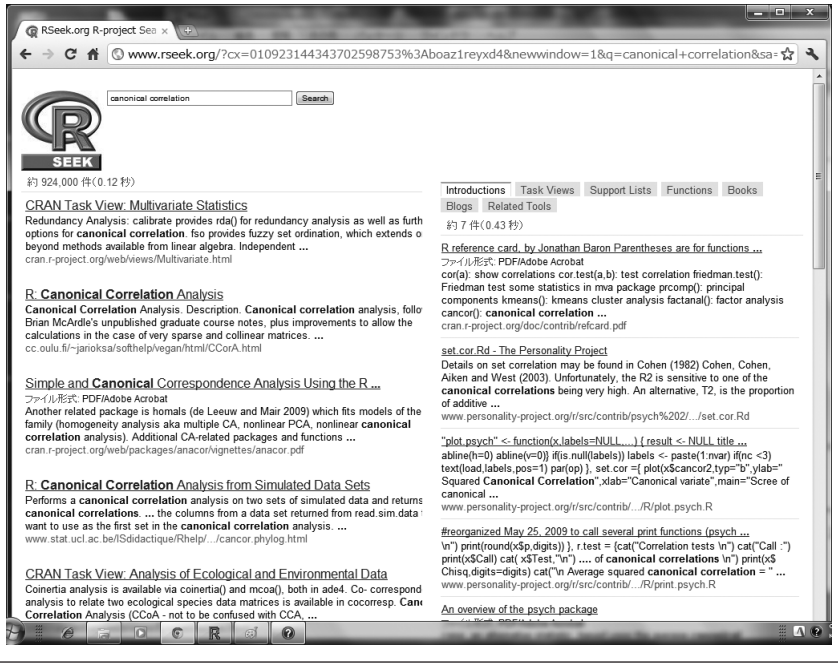


図 A-3a　rseek.org の検索結果

† 訳注：左側の検索結果については、日本語のキーワードでも表示される（図 A-3b 参照）。

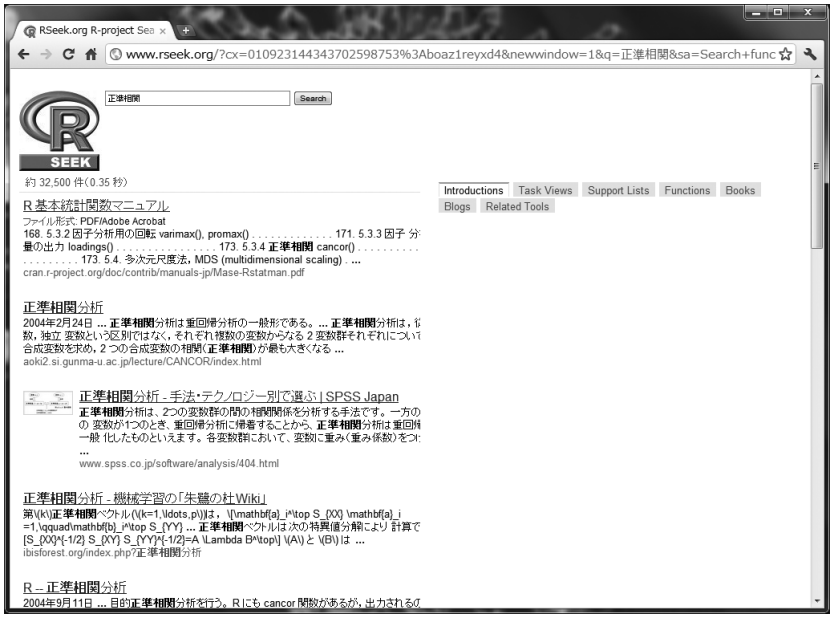


図 A-3b　日本語で検索した結果

は、検索語についての説明があるタスクビューがすべて表示されます。同様に「Functions」タブでは、関連する R 関数が表示されます。検索結果に的を絞るのは優れた方法です。

Stack Overflow（http://stackoverflow.com/）はいわゆる Q&A サイトです。つまり、だれでも質問を投稿でき、経験豊富なユーザが質問に答えてくれます。1 つの質問に対し、複数の答えが寄せられることもあります。読者は、答えに投票し、優れた答えは、上に上がります。これは読者が検索できる充実した Q&A 対話データベースを生成します。Stack Overflow は問題指向が強く、トピックは R のプログラミングの側面に偏っています。

Stack Overflow は多くのプログラミング言語に関する質問を受け付けています。そのため、検索ボックスに言葉を入力すると、「[r]」で始まる言葉は、R のタグがついた質問に焦点を当てます。例えば、「[r] standard error」で検索すると、R のタグがついた質問だけを選び、Python や C++ の質問は除外します。

Stack Exchange（Overflow ではない）は、統計解析に関する Q&A コーナーがあります。このコーナーは、プログラミングよりも統計に焦点を当てています。そのため、このサイトは、R 特有の問題というよりも統計一般について気になる問題の答えを探すときに使います。

関連項目

自分にとって便利なパッケージがわかったら、**レシピ C.9** を参照してその便利なパッケージをインストールしてください。

レシピ A.11 関連する関数とパッケージを探したい

問題

R には 2,000 を超えるパッケージがあり、自分にとって役に立つものがどれだかわからない。

解決策

- タスクビューの一覧（http://cran.r-project.org/web/views/）をながめて、役に立ちそうな分野のタスクビューを見つけてください。関連パッケージへのリンクと説明があります。あるいは http://rseek.org にアクセスしてキーワードで検索し、タスクビューのタブをクリックし、利用できそうなタスクビューを選択します。
- crantastic サイト（http://crantastic.org/) にアクセスし、キーワードでパッケージを検索します。
- 関連する関数を探すために http://rseek.org にアクセスして、名前またはキーワードで検索し、「Functions」タブをクリックします。

解説

この問題は特に初心者にとって頭痛の種です。R は自分の抱える問題を解決してくれると思っていますが、どのパッケージと関数が自分の役に立つのかわかりません。メーリングリストに共通する質問は、「問題 X を解決するパッケージはありますか」です。これは、R の海でおぼれている人に共通の静かな叫びでもあるのです。

これを書いている時点で、2000 以上のパッケージが CRAN からダウンロード可能です。それぞれのパッケージには要約ページがあり、短い説明とパッケージドキュメントへのリンクがあります。興味があるパッケージがあったら、「Reference manual（リファレンスマニュアル）」のリンクをクリックして全詳細が載っている PDF ドキュメントを開きます。（要約ページには、パッケージをインストールするためのダウンロードリンクも含んでいますが、このやり方でパッケージをインストールすることはほとんどないでしょう。**レシピ C.9** を参照）

例えば、ベイズ解析とか、計量経済学、最適化、グラフィックスのようなものに、ちょっと興味を持っただけの場合もあるでしょう。CRAN は役に立ちそうなパッケージを示すタスクビューページを含んでいます。タスクビューは、何が利用できるのかの概要がわかり、スタート地点としては最適です。タスクビューページの一覧は、http://cran.r-project.org/web/views/ または、「解決策」で説明した方法で探します。

偶然、例えば、オンラインで触れられているのを見たなどで、役に立つパッケージの名前を知ったとしましょう。全パッケージのアルファベット順の一覧は http://cran.r-project.org/web/packages/ にあり、パッケージ要約ページへのリンクもついています。

関連項目

他の方法でパッケージが検索できる `sos` と呼ばれる強力な R パッケージをダウンロード、インストールすることもできます。http://cran.r-project.org/web/packages/sos/vignettes/sos.pdf のビニエットを参照してください。

レシピ A.12　メーリングリストを検索する

問題

質問をしたい。同じ質問が過去にされているか、メーリングリストのアーカイブを探したい。

解決策

- Web ブラウザで、http://rseek.org を表示し、キーワードまたは他の検索語で探します。検索結果が現れたら、「Support Lists」タブをクリックします。
- RSiteSearch 関数を使って、R 内で検索することも可能です。

```
> RSiteSearch("keyphrase")
```

初期の検索結果はブラウザウィンドウに表示されます。「Target」の下の R-help リソースを選択し、他のソースをクリアし、再検索をします†。

解説

このレシピは、**レシピ A.10** の応用です。しかし重要です。メーリングリストに新たな質問を投稿する前にメーリングリストのアーカイブをまず探すべきだからです。過去に同じような質問がされ、回答が得られている可能性はあります。

関連項目

CRAN には Web 検索用のリソースもあります。http://cran.r-project.org/search.html を確認してください。

レシピ A.13　メーリングリストに質問を投稿する

問題

R-help メーリングリスト経由で R コミュニティに質問を投稿したい。

解決策

メーリングリスト（http://www.r-project.org/mail.html）のページは R-help メーリングリストを使う上での一般的な情報と使い方が書いてあります。一般的な手順を挙げます。

1. R メーリングリストの R-help リストに登録する（https://stat.ethz.ch/mailman/listinfo/r-help）。
2. 効果的な投稿について書かれた投稿ガイドを読む（http://www.r-project.org/posting-guide.html）

† 訳注：現在は、R-help は、R-help 1997-2001、R-help 2002-2007、R-help 2008-2009、R-help 2010- の 4 つの項目があります。

3. 質問を注意深く正確に書く。可能ならば、エラーや問題点を再現できるような最小限の例も載せる。

4. 質問を r-help@r-project.org 宛てにメールする。

解説

R メーリングリストは強力な情報源です。しかし最後の手段として使ってください。まずはヘルプページを読み、ドキュメントを読み、R-help メーリングリストのアーカイブを検索し、Web を検索します。あなたの質問はすでに回答されている場合がほとんどです。新しい質問はほとんどないのが現実です。

質問を書いたら、r-help@r-project.org 宛てに送信するだけなので、簡単です。しかし、メーリングリストに登録していないと、投稿しても拒否されます。

あなたの持っている問題点は、R のコードがエラーを発生させていたり、予期しない結果を起こしたりしていることが発端であることもあるかもしれません。そのような場合は、質問に必須な要素は、「最小限」と「完結」の実例（コード）なのです。

最小限

あなたの問題を示す最小限の R のコード片を書きます。関係のないものはすべて排除します。

完結

エラーを再現するために必要なデータを含みます。メーリングリストの読者が、再現できないと、診断できません。複雑なデータ構造の場合は、dump 関数を使ってデータの ASCII 表現を生成し、メッセージに添付します。

例が示された質問は明確になるので、より役に立つ答えを得る可能性が高くなります。

実際には複数のメーリングリストが存在します。R-help は一般的な質問の中心的存在です。他にも SIG（special interest group）メーリングリストがあり、特に遺伝子、金融、R の開発、さらには R の仕事といった分野に特化しています。メーリングリストの一覧は https://stat.ethz.ch/mailman/listinfo で確認できます。あなたの質問がこのような分野の 1 つであれば、適切なリストを選ぶことによって、よりよい答えが得られるでしょう。しかし、R-help の場合は、質問を投稿する前に、SIG リストアーカイブを注意深く検索してください。

関連項目

質問の投稿前に Eric Raymond と Rick Moen による素晴らしいエッセイ「カッコよく質問する方法」（"How to Ask Questions the Smart Way"）（http://www.catb.org/~esr/faqs/smart-questions.html）を一読することをお勧めします。

付録 B
R の基本

はじめに

この章のレシピは、問題解決のヒントとチュートリアルの中間に位置するようなものです。一般的な問題を解決するレシピも確かにありますが、解決策では、一般的なテクニックと、このクックブックで紹介している多くの R のコードで使われている作法を披露しています。R が初めてなら、まずこの章に目を通して、このような作法に慣れておくことをお勧めします。

レシピ B.1　表示する

問題

変数や式の値を表示したい。

解決策

コマンドプロンプトで変数名や式を入力するだけで、R はその値を出力します。print 関数を使えば、あらゆるオブジェクトの一般的な表示ができます。cat 関数を使うと、出力形式をカスタマイズできます。

解説

R で何か表示することはとても簡単です。表示させたいものをコマンドプロンプトに入力すればよいのです。

```
> pi
[1] 3.141593
> sqrt(2)
[1] 1.414214
```

このように式を入力すると、R は式を評価して、自動的に print 関数を呼び出します。つまり、上の例は、次のものと同じです。

```
> print(pi)
[1] 3.141593
```

```
> print(sqrt(2))
[1] 1.414214
```

print は、どのようなRの値であっても、行列やリストのように構造化された値でさえも、どのようにフォーマットして出力すべきか知っているので、美しいと言えます。

```
> print(matrix(c(1,2,3,4), 2, 2))
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> print(list("a","b","c"))
[[1]]
[1] "a"

[[2]]
[1] "b"

[[3]]
[1] "c"
```

print と入力するだけで、いつでもデータを見ることができるので便利です。特別な出力ロジックやデータ構造さえも書く必要はありません。しかし print 関数は1つだけ大きな制限があります。一度に1つのオブジェクトしか表示できません。print で複数の項目を表示しようとすると、次のような退屈なエラーメッセージが表示されます。

```
> print("The zero occurs at", 2*pi, "radians.")
 以下にエラー print.default("The zero occurs at", 2 * pi, "radians.") :
   'quote' 引数が不正です
```

print を使って複数の項目を出力する唯一の方法は、おそらく、できれば避けたい方法でしょう。

```
> print("The zero occurs at"); print(2*pi); print("radians")
[1] "The zero occurs at"
[1] 6.283185
[1] "radians"
```

cat 関数は、print の代わりに複数の項目を組合せて連続した形で出力できます。

```
> cat("The zero occurs at", 2*pi, "radians.", "\n")
The zero occurs at 6.283185 radians.
```

cat はデフォルトで項目間にスペースを挿入しています。また行を終わらせたいところで改行文字（\n）を入れる必要があります。

cat 関数は、単純なベクトルも出力できます。

```
> fib <- c(0,1,1,2,3,5,8,13,21,34)
> cat("The first few Fibonacci numbers are:", fib, "...\n")
The first few Fibonacci numbers are: 0 1 1 2 3 5 8 13 21 34 ...
```

cat を使うと、出力をさらに制御できます。出力の制御は特に R スクリプトで役に立ちます。しかし、cat にも厳格な制限があり、行列やリストのようなデータ構造は表示できません。このようなデータを cat で表示しようとすると、またもや退屈なメッセージが出ます。

```
> cat(list("a","b","c"))
以下にエラー cat(list(...), file, sep, fill, labels, append) :
   引数 1 (タイプ 'list') は 'cat' で取り扱えません
```

関連項目

出力形式の制御については**レシピ 1.2** を参照してください。

レシピ B.2　変数の設定

問題

変数に値を保存したい。

解決策

割り当て演算子（<-）を使います。最初に変数を宣言する必要はありません。

```
> x <- 3
```

解説

「計算モード」で R を使っているとすぐに物足りなくなり、そのうちに変数を定義して、値を保存したくなるでしょう。変数を設定すると、入力の手間が減り、時間が節約され、作業がすっきりします。

R では変数を宣言したり、明示的に作成する必要はありません。変数名に値を割り当てるだけで、R は変数を作成してくれます。

```
> x <- 3
> y <- 4
> z <- sqrt(x^2 + y^2)
> print(z)
[1] 5
```

割り当て演算子は小なり記号（<）とハイフン（-）の組合せです。小なり記号とハイフンの間にスペースは入りません。

コマンドプロンプトで上の例のように変数を定義すると、変数は**ワークスペース**で使えるようになります。ワークスペースはPCのメインメモリ上に置かれ、Rを終了するときに、ディスクに保存することもできます。変数を削除するまで、変数の定義はワークスペース上で有効です。

Rは**動的な型付け言語**です。つまり、変数のデータ型を好きなように変えられます。先ほど示したように、xを数値型にもできるし、気が変わって文字列のベクトルで上書きすることもできます。それでもRは文句は言いません。

```
> x <- 3
> print(x)
[1] 3
> x <- c("fee", "fie", "foe", "fum")
> print(x)
[1] "fee" "fie" "foe" "fum"
```

R関数では、見慣れない割り当て演算子 <<- を使っている割り当て文を目にすることもあるでしょう。

```
x <<- 3
```

これは、ローカル変数ではなく、グローバル変数に割り当てています。

包み隠さず公開するという方針ですので、Rが他に2つの割り当て文の形式もサポートしていることをお話します。等号1つ（=）は、コマンドプロンプトで割り当て演算子として使われることがあります。右向きの割り当て演算子（->）は、左向きの割り当て演算子（<-）が使えるところであればどこでも使えます。

```
> foo = 3
> print(foo)
[1] 3
> 5 -> fum
> print(fum)
[1] 5
```

上記の例で示したような形はこの本では使いません。また、使わないように勧めます。等号の割り当ては、等しいか等しくないかの検証と混同しやすいものです。右向きの割り当て演算子は普通は使いません。さらに、式が長くなりすぎたときに読みにくくなるという欠点もあります。

関連項目

レシピB.4、レシピB.14、レシピC.2を参照。assign関数のヘルプページも参照してください。

レシピ B.3　変数の一覧を表示する

問題

ワークスペースで定義されている関数と変数が知りたい。

解決策

ls 関数を使います。ls.str を使うとそれぞれの変数についてさらに詳しい情報がわかります。

解説

ls 関数はワークスペース内のオブジェクト名を表示します。

```
> x <- 10
> y <- 50
> z <- c("three", "blind", "mice")

> f <- function(n,p) sqrt(p*(1-p)/n)
> ls()
[1] "f" "x" "y" "z"
```

ls は変数または関数の名前の文字列のベクトルを返します。ワークスペースに何もなければ空のベクトルを返します（不思議な出力です）。

```
> ls()
character(0)
```

この出力は「ls が長さが 0 のの文字列ベクトルを返した」と表現する R 特有の言い方になっています。つまり、ワークスペースに何も定義されていないので ls は空のベクトルを返したのです。

名前の一覧だけでなく、さらに詳しい情報が欲しければ ls.str を実行してみます。それぞれの変数についてさらに詳しい情報を教えてくれます。

```
> ls.str()
f : function (n, p)
x : num 10
y : num 50
z : chr [1:3] "three" "blind" "mice"
```

この関数は、変数の一覧の表示と str 関数の適用の両方を行って変数の構造を示すために ls.str と呼ばれます（**レシピ 9.15**）。

普通、ls はドット（.）で始まる名前のものは返しません。ドットで始まる名前は、隠されているもの、ユーザにとって興味がないものと考えられているからです（Unix でのドットから始まる名前を表示しないという規約を反映しています）。すべてを強制的に表示するには、引数 all.names を TRUE に設定します。

```
> .hidvar <- 10
> ls()
[1] "f" "x" "y" "z"
> ls(all.names=TRUE)
[1] ".hidvar" "f" "x" "y" "z"
```

関連項目

変数の削除は**レシピ B.4** を、変数の構造を明らかにするには**レシピ 9.15** を参照してください。

レシピ B.4 変数の削除

問題

必要のない変数や関数をワークスペースから削除したい。あるいは内容を完全に消去したい。

解決策

rm 関数を使います。

解説

ワークスペースはすぐに散らかってしまうものです。rm 関数はワークスペースから 1 つ以上のオブジェクトを永久に削除します。

```
> x <- 2*pi
> x
[1] 6.283185
> rm(x)
> x
エラー： オブジェクト 'x' がありません
```

rm を行うと「undo」は使えません。いったん変数を消してしまうと、完全に失われてしまいます。
いくつかの変数を一度に削除することもできます。

```
> rm(x,y,z)
```

ワークスペースの変数を一度に消去することもできます。rm 関数の引数 list に、消去する変数名のベクトルを指定します。ls 関数は変数名のベクトルを返しましたので、rm と ls を組合せればすべての削除できます。

```
> ls()
[1] "f" "x" "y" "z"
> rm(list=ls())
> ls()
character(0)
```

礼儀正しく

他の人も使うようなコード、例えばライブラリ関数やメーリングリストに投稿するサンプルコードの中に `rm(list=ls())` を絶対に入れてはいけません。誰かのワークスペースのすべての変数を消去してしまいます。これはとんでもなく失礼な行為で、あなたの評判を著しく損ないます。

関連項目

レシピ B.3 を参照。

レシピ B.5　ベクトルを作成する

問題

ベクトルを作成したい。

解決策

`c(...)` 演算子を使って与えられた値からベクトルを作成します。

解説

ベクトルは R の中核となる要素で、他のデータ構造とは少し違います。ベクトルには、数値も、文字列も、論理値も入れることができますが、混在させることはできません。`c(...)` 演算子は単純な要素からベクトルを作成します。

```
> c(1,1,2,3,5,8,13,21)
[1]  1  1  2  3  5  8 13 21
> c(1*pi, 2*pi, 3*pi, 4*pi)
[1] 3.141593 6.283185 9.424778 12.566371
> c("Everyone", "loves", "stats.")
[1] "Everyone" "loves" "stats."
> c(TRUE,TRUE,FALSE,TRUE)
[1]  TRUE  TRUE FALSE  TRUE
```

`c(...)` の引数自体がベクトルの場合は、`c(...)` はベクトルを結合し平坦化して 1 つのベクトルにまとめます。

```
> v1 <- c(1,2,3)
> v2 <- c(4,5,6)
> c(v1,v2)
[1] 1 2 3 4 5 6
```

ベクトルは複数のデータ型を混在させることができません。例えば数値と文字列を混在させることはできません。複数のデータ型の要素が混在したベクトルを作成したい場合、Rは片方の要素を変換しようとします。

```
> v1 <- c(1,2,3)
> v3 <- c("A","B","C")
> c(v1,v3)
[1] "1" "2" "3" "A" "B" "C"
```

この例では数値と文字列のベクトルから新たなベクトルを作ろうとしています。Rベクトルを作成する前にすべての数値を文字列に変換して、データ要素を互換性を保つようにします。

技術的な面から言うと、2つのデータ要素は両者が同じモードを持つ場合に限り共存できます。例えば、3.1415はnumericモード、"foo"はcharacterモードです。

```
> mode(3.1415)
[1] "numeric"
> mode("foo")
[1] "character"
```

両モードは互換性がありません。この2つからベクトルを作成するためにRは3.1415をcharacterモードに変換して、"foo"と互換性を持つようにします。

```
> c(3.1415, "foo")
[1] "3.1415" "foo"
> mode(c(3.1415, "foo"))
[1] "character"
```

cはジェネリック演算子です。つまり、ベクトルに限らず多くのデータ型で使えます。しかし、あなたが期待するものではないかもしれないので、他のデータ型やオブジェクトに対して使う前にその振る舞いを確認するようにしてください。

関連項目

ベクトルと他のデータ構造について詳しくは2章の「はじめに」を参照してください。

レシピ B.6 基本統計量を求める

問題

基本統計量、例えば平均値、中央値、標準偏差、分散、相関、共分散を求めたい。

解決策

次の関数を使います。引数のxとyはベクトルです。

- mean(x)
- median(x)
- sd(x)
- var(x)
- cor(x, y)
- cov(x, y)

解説

私がはじめて R のドキュメントを開いたとき、最初に探したのは、「Procedures for Calculating Standard Deviation」（標準偏差を求めるための手順）という項目でした。このような重要なトピックはこの本全体を通して求められるだろうと思います。

難しいことは何もありません。

標準偏差と他の基本統計量は簡単な関数で求められます。通常、関数の引数は数値のベクトルで、関数は統計値を計算して返します。

```
> x <- c(0,1,1,2,3,5,8,13,21,34)
> mean(x)
[1] 8.8
> median(x)
[1] 4
> sd(x)
[1] 11.03328
> var(x)
[1] 121.7333
```

sd 関数は、標本標準偏差を求め、var は標本分散を求めます。cor 関数は 2 つのベクトルの相関を求め、cov は共分散を求めます。

```
> x <- c(0,1,1,2,3,5,8,13,21,34)
> y <- log(x+1)
> cor(x,y)
[1] 0.9068053
> cov(x,y)
[1] 11.49988
```

cor 関数も cov 関数も、利用できない値（NA 値）に対して寛容ではありません。ベクトル引数中に、NA 値が 1 つでもあると、両者とも NA 値を返すか、または不可解なエラーを出して中断することもあります。

```
> x <- c(0,1,1,2,3,NA)
> mean(x)
[1] NA
> sd(x)
[1] NA
```

Rからのこのような警告は煩わしいですが、正しいことを行うようにします。注意深く現状を考えなければなりません。データ中のNA値は統計値を無効にしているでしょうか。もしそうなら、Rは正しいことを行っています。違う場合は、na.rm=TRUEと設定することによって、この振る舞いを変更できます。この設定により、RにNA値を無視するように指示します。

```
> x <- c(0,1,1,2,3,NA)
> mean(x, na.rm=TRUE)
[1] 1.4
> sd(x, na.rm=TRUE)
[1] 1.140175
```

mean関数とsd関数にはデータフレームの扱いに長けているという優れた一面があります。この2つの関数はいずれも、データフレームの各列が異なる変数であることを理解しており、各列に対して個別に統計値を求めます。次の例では、3列からなるデータフレームの基本統計量を求めています。

```
> print(dframe)
       small    medium       big
1  0.6739635 10.526448  99.83624
2  1.5524619  9.205156 100.70852
3  0.3250562 11.427756  99.73202
4  1.2143595  8.533180  98.53608
5  1.3107692  9.763317 100.74444
6  2.1739663  9.806662  98.58961
7  1.6187899  9.150245 100.46707
8  0.8872657 10.058465  99.88068
9  1.9170283  9.182330 100.46724
10 0.7767406  7.949692 100.49814
> mean(dframe)
   small    medium       big
1.245040  9.560325 99.946003
> sd(dframe)
    small    medium       big
0.5844025 0.9920281 0.8135498
```

mean関数もsd関数も両方とも、各列がデータフレームで定義された3つの値を返しています。（技術的にはどちらの関数もnames属性がデータフレームの列から取ってきた3要素のベクトルを返します。）

var関数もデータフレームを理解しますが、meanやsdとは全く異なる振る舞いをします。var関数はデー

タフレームの列の共分散を求め、共分散行列を返します。

```
> var(dframe)
             small      medium         big
small   0.34152627 -0.21516416 -0.04005275
medium -0.21516416  0.98411974 -0.09253855
big    -0.04005275 -0.09253855  0.66186326
```

同様に、x がデータフレームであっても行列であっても、cor(x) は相関行列を返し、cov(x) は共分散行列を返します。

```
> cor(dframe)
             small     medium         big
small   1.00000000 -0.3711367 -0.08424345
medium -0.37113670  1.0000000 -0.11466070
big    -0.08424345 -0.1146607  1.00000000
> cov(dframe)
             small      medium         big
small   0.34152627 -0.21516416 -0.04005275
medium -0.21516416  0.98411974 -0.09253855
big    -0.04005275 -0.09253855  0.66186326
```

median 関数は、なんとデータフレームを理解しません。データフレーム列の中央値を求めるには、**レシピ 3.4** を使い、median 関数を各列に個別に適応させます。

関連項目

レシピ B.14、レシピ 3.4、レシピ 6.17 を参照

レシピ B.7　数列を作る

問題

数列を作りたい。

解決策

n:m 式を使って、単純な数列 $n, n+1, n+2, ..., m$ を作ります。

```
> 1:5
[1] 1 2 3 4 5
```

増分が 1 以外の単調増加数列を作るには seq 関数を使います。

```
> seq(from=1, to=5, by=2)
[1] 1 3 5
```

同じ値が連続する数列を作るには rep 関数を使います。

```
> rep(1, times=5)
[1] 1 1 1 1 1
```

解説

コロン演算子（n:m）は、$n, n+1, n+2, ..., m$ からなるベクトルを作成します。

```
> 0:9
 [1] 0 1 2 3 4 5 6 7 8 9
> 10:19
 [1] 10 11 12 13 14 15 16 17 18 19
> 9:0
 [1] 9 8 7 6 5 4 3 2 1 0
```

最後に挙げた例（9:0）で、Rが優れていることをよく見てください。9は0よりも大きく、逆順に値を出力しています。

コロン演算子は増分1のみの数列に有効です。seq 関数も同様に数列を作成しますが、省略可能な3番目の引数は増分を意味します。

```
> seq(from=0, to=20)
 [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> seq(from=0, to=20, by=2)
 [1]  0  2  4  6  8 10 12 14 16 18 20
> seq(from=0, to=20, by=5)
[1]  0  5 10 15 20
```

さらに出力列の長さを指定できます。すると、Rは指定した長さだけの増分を求めます。

```
> seq(from=0, to=20, length.out=5)
[1]  0  5 10 15 20
> seq(from=0, to=100, length.out=5)
[1]   0  25  50  75 100
```

増分は整数である必要はありません。Rは小数の数列を作ることもできます。

```
> seq(from=1.0, to=2.0, length.out=5)
[1] 1.00 1.25 1.50 1.75 2.00
```

ある1つの値の連続は「数列」の特殊な場合です。rep 関数を使って1番目の引数の繰り返しの数列を作

れます。

```
> rep(pi, times=5)
[1] 3.141593 3.141593 3.141593 3.141593 3.141593
```

関連項目

日付オブジェクトの列を作成するには**レシピ 4.14** を参照してください。

レシピ B.8　ベクトルを比較する

問題

2 つのベクトルを比較したい。またはスカラとベクトル全体を比較したい。

解決策

比較演算子（==, !=, <, >, <=, >=）を使うと、2 つのベクトルの要素ごとの比較ができます。比較演算子を使ってベクトル要素とスカラを比較することもできます。各要素ごとの比較の結果の値からなる論理値のベクトルを、結果として出力します。

解説

R には 2 つの**論理値**、TRUE と FALSE があります。論理値は、他のプログラミング言語では**ブール値**（あるいは**真偽値**）と呼ばれることもあります。

比較演算子は 2 つの値を比較し、比較の結果に従って TRUE または FALSE を返します。

```
> a <- 3
> a == pi      # 等しいかの検証
[1] FALSE
> a != pi      # 等しくないかの検証
[1] TRUE
> a < pi
[1] TRUE
> a > pi
[1] FALSE
> a <= pi
[1] TRUE
> a >= pi
[1] FALSE
```

一度にベクトル全体を比較することによって、R の威力を体感することができるでしょう。R は要素ごとの比較を行い、論理値のベクトルを返します。

```
> v <- c( 3, pi,  4)
> w <- c(pi, pi, pi)
> v == w                  # 2 つの 3 要素ベクトルの比較
```

```
[1] FALSE TRUE FALSE      # 3 要素ベクトルの比較結果
> v != w
[1] TRUE FALSE TRUE
> v < w
[1] TRUE FALSE FALSE
> v <= w
[1] TRUE TRUE FALSE
> v > w
[1] FALSE FALSE TRUE
> v >= w
[1] FALSE TRUE TRUE
```

ベクトルと 1 つのスカラを比較することもできます。この場合、R はスカラをベクトルの長さに拡張して、要素ごとの比較を行います。前の例は次のように簡単にできます。

```
> v <- c(3, pi, 4)
> v == pi               # 3 要素のベクトルを 1 つの数と比較
[1] FALSE TRUE FALSE
> v != pi
[1] TRUE FALSE TRUE
```

（これはリサイクル規則の適用です。**レシピ 2.3** を参照）

2 つのベクトルを比較するとき、比較の一部が TRUE だったのか、すべての比較が TRUE だったのか知りたいこともあるでしょう。any 関数と all 関数は、このような検証を扱います。どちらも論理ベクトルを検証します。any 関数はベクトル要素に 1 つでも TRUE があれば TRUE を返します。all 関数はベクトル要素がすべて TRUE ならば TRUE を返します。

```
> v <- c(3, pi, 4)
> any(v == pi)            # v のいずれかの要素が pi と等しければ TRUE を返す
[1] TRUE
> all(v == 0)             # v のすべての要素が pi と等しければ TRUE を返す
[1] FALSE
```

関連項目

レシピ B.9 を参照。

レシピ B.9　ベクトルの要素を選択する

問題

ベクトルから 1 つあるいは複数の要素を抽出したい。

解決策

自分の問題に適合したインデックス付けの方法を選びます。

- ベクトル要素を位置で選択するには、角括弧を使います。例えばベクトル v の 3 番目の要素を選択したい場合は、v[3] を使います。
- 要素を除外するには、負のインデックスを使います。
- インデックスのベクトルを使って複数の値を選択します。
- 論理ベクトルを使って条件に基づいた要素を選択します。
- 名前を使って名前要素にアクセスします。

解説

ベクトルからの要素の選択も R の強力な機能の 1 つです。基本的な選択方法はその他多くのプログラミング言語と同様で、角括弧と単純なインデックスを使います。

```
> fib <- c(0,1,1,2,3,5,8,13,21,34)
> fib
[1] 0 1 1 2 3 5 8 13 21 34
> fib[1]
[1] 0
> fib[2]
[1] 1
> fib[3]
[1] 1
> fib[4]
[1] 2
> fib[5]
[1] 3
```

他のプログラミング言語では、1 番目の要素のインデックスが 0 から始まるものもありますが、R では 1 番目の要素のインデックスは 0 ではなくて 1 です。

ベクトルインデックスは、一度に複数の要素を選択できるという優れた機能があります。インデックス自体にもベクトルが使えます。また、インデックスに使うベクトルの要素は、データベクトルから選んだものです。

```
> fib[1:3]            # 1 から 3 までの要素を選択
[1] 0 1 1
> fib[4:9]            # 4 から 9 までの要素を選択
[1]  2  3  5  8 13 21
```

1:3 というインデックスは、文字通り 1, 2, 3 番目の要素を選びます。インデックスに使うベクトルは単純な数列である必要はありませんが、データベクトル内から要素を選ぶ必要があります。次の例の中では、1, 2, 4, 8 番目の要素を選択します。

```
> fib[c(1,2,4,8)]
[1] 0 1 2 13
```

Rは負のインデックスの値を除外するという意味に解釈します。インデックスが−1だと1番目の値を除いたほかのすべての値を返します。

```
> fib[-1]              # 1番目の要素を無視
[1]  1  1  2  3  5  8 13 21 34
```

この方法を拡張して、負のインデックスを持つベクトルを使ってスライス全体を除外することも可能となります。

```
> fib[1:3]             # 先ほどの例
[1] 0 1 1
> fib[-(1:3)]          # 負のインデックスは選択せず除外する
[1]  2  3  5  8 13 21 34
```

その他のインデックス付けを使ったテクニックとして、論理ベクトルを使ってデータベクトルから要素を選択するというものがあります。論理ベクトルがTRUEの要素はいずれも選ばれます。

```
> fib < 10                  # fibが10未満ならこのベクトルはTRUEになります
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE
> fib[fib < 10]             # 上のベクトルを使って10未満の要素を選択します
[1] 0 1 1 2 3 5 8

> fib %% 2 == 0             # fibが偶数ならこのベクトルはTRUEになります
[1] TRUE FALSE FALSE TRUE FALSE FALSE TRUE FALSE FALSE TRUE
> fib[fib %% 2 == 0]        # 上のベクトルを使って偶数の要素を選択します
[1] 0 2 8 34
```

通常、論理ベクトルは、データベクトルと同じ長さであるべきで、各要素を含む／含まないをはっきりさせます（長さが異なる場合は、リサイクル規則を理解する必要があります。**レシピ2.3**を参照）。

ベクトル比較、論理演算子、ベクトルインデックスを組合せ、わずかなRのコードできめ細かな選択を行うことができます。

中央値よりも大きな要素をすべて選択する。

```
v[ v > median(v) ]
```

上位および下位5%に入らない要素をすべて選択する。

```
v[ (v < quantile(v,0.05)) | (v > quantile(v,0.95)) ]
```

平均値から±2の標準偏差の範囲にない要素をすべて選択する。

```
v[ abs(v-mean(v)) > 2*sd(v) ]
```

NA でも NULL でもないすべての要素を選択する。

```
v[ !is.na(v) & !is.null(v) ]
```

最後に紹介するインデックスの機能は、名前で要素を選択するものです。この機能は、ベクトルが names 属性を持つことを仮定しています。names 属性は各要素の名前を定義するもので、文字列のベクトルを属性に割り当てて定義します。

```
> years <- c(1960, 1964, 1976, 1994)
> names(years) <- c("Kennedy", "Johnson", "Carter", "Clinton")
> years
Kennedy Johnson  Carter Clinton
   1960    1964    1976    1994
```

いったん名前が定義されると、各要素を名前で参照できるようになります。

```
> years["Carter"]
Carter
1976
> years["Clinton"]
Clinton
1994
```

この一般化により、名前のベクトルでインデックスが付けられるようになります。つまり R はインデックスに名前の付いた要素を返します。

```
> years[c("Carter","Clinton")]
Carter Clinton
1976 1994
```

関連項目

リサイクル規則についてさらに詳しい説明は**レシピ 2.3** を参照してください。

レシピ B.10　ベクトル演算を行う

問題

一度にベクトル全体の演算をしたい。

解決策

通常の算術演算子はベクトル全体に対し要素ごとの演算を行います。ベクトル全体の演算を行い、ベクトルの結果を返す関数も数多くあります。

解説

ベクトル演算は、Rの優れた強みの1つです。すべての基本算術演算子はベクトルにも適用できます。演算は要素ごとの方法で行います。つまり、演算の対象となる2つのベクトルの対応する要素に演算子を適用します。

```
> v <- c(11,12,13,14,15)
> w <- c(1,2,3,4,5)
> v + w
[1] 12 14 16 18 20
> v - w
[1] 10 10 10 10 10
> v * w
[1] 11 24 39 56 75
> v / w
[1] 11.000000  6.000000  4.333333  3.500000  3.000000
> w ^ v
[1]           1        4096     1594323   268435456 30517578125
```

結果の長さは、もともとのベクトルと同じ長さであることがわかります。各要素は入力したベクトルの対応する値の組であるからです。

被演算数（オペランド）の一方がベクトルで、もう一方がスカラの場合、演算は各ベクトル要素とスカラの間で行われます。

```
> w
[1] 1 2 3 4 5
> w + 2
[1] 3 4 5 6 7
> w - 2
[1] -1  0  1  2  3
> w * 2
[1]  2  4  6  8 10
> w / 2
[1] 0.5 1.0 1.5 2.0 2.5
> w ^ 2
[1]  1  4  9 16 25
> 2 ^ w
[1]  2  4  8 16 32
```

例えば、1つの式で各要素の値から平均値を引くだけで、平均値が0となるようにベクトル全体をシフトすることもできます。

```
> w
[1] 1 2 3 4 5
> mean(w)
[1] 3
> w - mean(w)
[1] -2 -1  0  1  2
```

同様に、平均値を引いて標準偏差で割るという 1 つの式で、ベクトルの Z 値を求めることもできます。

```
> w
[1] 1 2 3 4 5
> sd(w)
[1] 1.581139
> (w - mean(w)) / sd(w)
[1] -1.2649111 -0.6324555  0.0000000  0.6324555  1.2649111
```

ベクトルを対象とする演算の実装は、基本的な計算に留まりません。言語全体にわたってベクトル全体の演算を行う関数は数多くあります。例えば、関数 sqrt と関数 log は、ベクトルのすべての要素に対して演算を行い、結果のベクトルを返します。

```
> w
[1] 1 2 3 4 5
> sqrt(w)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
> log(w)
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
> sin(w)
[1]  0.8414710  0.9092974  0.1411200 -0.7568025 -0.9589243
```

ベクトル演算には 2 つの大きな利点があります。まず最も顕著な特徴は簡単であるということです。他の言語ではループが必要な演算が、R では 1 行で済んでしまいます。2 番目は速さです。ベクトル化された演算はほとんど、C コードで直接実装されています。そのため、同等の R コードと比較してずっと速いのです。

関連項目

ベクトルとスカラ間の演算は、実際にはリサイクル規則の特殊な場合です。**レシピ 2.3** を参照してください。

レシピ B.11 演算子の優先順位を理解する

問題

R式の結果がおかしい。演算子の優先順位が原因ではないかと思う。

解決策

表 B-1 に全演算子の一覧を示します。優先順位の高い順に並べています。指定がない限り、優先度が同じ演算子は左から右に評価されます。

表 B-1 演算子の優先順位

演算子	意味	関連項目
[[[	インデックス付け	レシピ B.9
:: :::	ネームスペースの変数にアクセス	
$ @	構成要素の抽出、スロットの抽出	
^	指数（右から左）	
- +	単項マイナス、単項プラス	
:	数列の作成	レシピ B.7、レシピ 4.14
%any%	特殊演算子	解説
* /	乗算、除算	解説
+ -	加算、減算	
== != < > <= >=	比較	レシピ B.8
!	論理否定	
& &&	論理積、ショートサーキット論理積	
| ||	論理和、ショートサーキット論理和	
~	式（フォーミュラ）	レシピ 8.1
-> ->>	右側の割り当て	レシピ B.2
=	割り当て（右から左）	レシピ B.2
<- <<-	割り当て（右から左）	レシピ B.2
?	ヘルプ	レシピ A.7

解説

Rで演算子の優先順位がおかしくなってしまうという問題はよく起こるものです。私もよく遭遇します。式 `0:n-1` は、0から $n-1$ までの整数列を生成するものと何となくと予想しますが、実際は異なります。

```
> n <- 10
> 0:n-1
[1] -1  0  1  2  3  4  5  6  7  8  9
```

実際には、Rはこの式を `(0:n)-1` と解釈し、-1 から $n-1$ までの数列を生成します。読者は**表 B-1** の %any% という記法を見たことがないかもしれません。Rはパーセント記号 (`%...%`) に囲まれたテキストはす

べて二項演算子として解釈します。

以下に事前に意味が定義されている、%...% の形の演算子を挙げます。

演算子	意味
%%	剰余演算子
%/%	整数の除算
%*%	行列の乗算
%in%	左側の被演算数（オペランド）が右側にも現れるときに TRUE を返す。そうでなければ FALSE を返す。

%...% 記法を使って新しい二項演算子を定義することもできます。詳しくは**レシピ 9.19** を参照してください。ここで強調したいのは、%...% で定義した演算子はすべて同じ優先順位を持つということです。

関連項目

ベクトル演算について詳しくは**レシピ B.10** を、行列演算については**レシピ 2.15** を、独自の演算子の定義については**レシピ 9.19** を参照してください。R のヘルプページの Arithmetic（算術演算）と Syntax（構文）の項目、『R in Nutshell』（O'Reilly）の 5、6 章も参照してください。

レシピ B.12　関数を定義する

問題

R の関数を定義したい。

解決策

function キーワードとそれに続くパラメータのリスト（$param_1$, ..., $param_N$）、関数本体（*expr*）を使って関数を定義します。1 行での関数定義は以下のようになります。

```
function(param1, ...., paramN) expr
```

関数本体は複数行の式でも定義できます。その場合は、関数本体の前後に波括弧を付けます。

```
function(param1, ..., paramN) {
    expr1
    .
    .
    .
    exprM
}
```

解説

関数定義は「これを計算する方法はこうです」と伝えるものです。例えば、R には変動係数を計算する組

み込み関数がありません。しかし、以下のように変動係数の計算を定義することができます。

```
> cv <- function(x) sd(x)/mean(x)
> cv(1:10)
[1] 0.5504819
```

1行目で関数を作成し、cvに割り当てます。2行目はxのパラメータの値として1:10を使って関数を呼び出します。関数はその式sd(x)/mean(x)の値を返します。

いったん関数を定義すると、どこでも使えるようになります。例えば、lappy関数の第2引数として使うこともできます（**レシピ**3.2）。

```
> cv <- function(x) sd(x)/mean(x)
> lapply(lst, cv)
```

複数行の関数は波括弧を使って関数本体の先頭と末尾を区切っています。以下は2つの整数の最大公約数を求めるユークリッドの互除法を実装する関数です。

```
> gcd <- function(a,b) {
+     if (b == 0) return(a)
+     else return(gcd(b, a %% b))
+ }
```

さらにRでは**匿名関数**を使うこともできます。匿名関数は名前がなく、1行のコードに向いています。先ほどのcvとlapplyを使った例は、匿名関数を使えば、それがlappyに直接渡されるので1行にできます。

```
> lapply(lst, function(x) sd(x)/mean(x))
```

この本は、Rプログラミングについてのものではないので、R関数のコーディングの細かな点までカバーすることはできません。しかし、役に立つヒントを次に挙げておきます。

戻り値
: すべての関数は値を返します。通常、関数はその関数本体の最後の式の値を返します。return(*expr*)を使うこともできます。

値による呼び出し
: 関数パラメータは「値により呼び出し」をされます。つまり、パラメータを変更すると、その変更はローカルに限られ、呼び出し側の値に影響しません。

ローカル変数
: 値を割り当てるだけでローカル変数を作成することができます。関数が終了するとローカル変数は失われます。

条件実行

R 構文には if 文もあります。詳しくは help(Control) を参照してください。

ループ

R 構文には for ループ、while ループ、repeat ループもあります。詳しくは help(Control) を参照してください。

グローバル変数

割り当て演算子 <<- を使って、関数内でグローバル変数を変更することができますが、推奨できません。

関連項目

関数定義についてさらに詳しくは、「Introduction to R」（http://cran.r-project.org/doc/manuals/R-intro.pdf）と、『R in a Nutshell』を参照してください。

レシピ B.13　少ない入力で大きな成果を上げる

問題

文字数の多いコマンドの入力は面倒だ。また同じコマンドを何回も入力するのも面倒だ。

解決策

エディタウィンドウを開き、再利用できそうな R コードのブロックを溜めておきます。そうやって集めたコードブロックをあとでウィンドウから直接実行します。入力を少なく済ませるためにコマンドラインを取っておくのです。あるいは、1 回限りのコマンドも取っておきます。

そうしておいて、あとで使えるように、スクリプトファイルに保存します。

解説

R 初心者は通常、1 つの式を入力し、それによって何が起こるかを確認します。慣れてくると、次第に複雑な式も入力するようになります。複数行の式も入力するようになるでしょう。そして、次第に複雑な計算を行うために、同じような複数行の式を何度も何度も入力するようになります。

経験豊富なユーザであれば複雑な式を何度も入力するようなことはしません。一度か二度同じ式を入力して、その式が便利で再利用できるものだと気づくと、その式をカット＆ペーストして、GUI のエディタウィンドウにペーストします。あとでそのコード片を実行する際に、再び入力するようなことはありません。実行したいコード片をエディタウィンドウ内で選択し、R に実行させるだけです。この手法は、コード片が長くなって大きなブロックのコードになってしまうような場合に特に威力を発揮します。

Windows と OS X には、エディタウィンドウを使ったコマンド実行を支援するための GUI 機能がいくつかあります。

エディタウィンドウを開く

メインメニューから「ファイル」→「新しいスクリプト」を選択します。

エディタウィンドウの1行を実行する

実行したい行にカーソルを合わせてCtrl-Rを押すとその行が実行されます。

エディタウィンドウの複数行を実行する

マウスを使って実行したい行を反転させ、Ctrl-Rを押すとその行が実行されます。

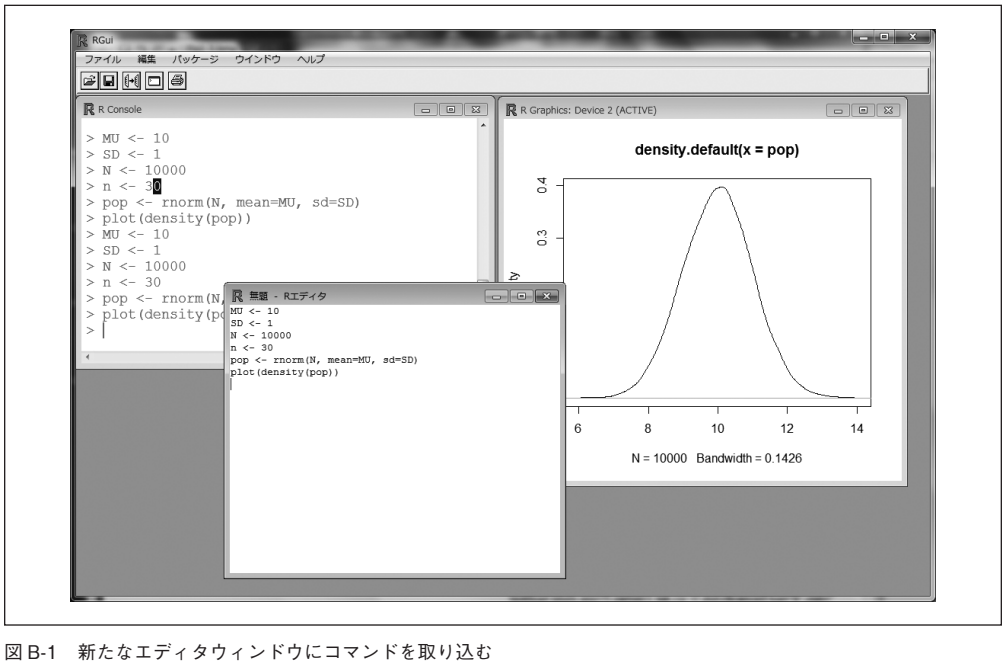
エディタウィンドウの内容すべてを実行する

Ctrl-Aでウィンドウ内のすべての内容を選択し、Ctrl-Rを押します。あるいは、メインメニューから「編集」→「全て実行」を選択します。

ただ単にコピー&ペーストするだけで、コンソールウィンドウからエディタウィンドウに行をコピーすることができます。Rを終了する際、新しいスクリプトを保存するかどうか尋ねられるので、今後の利用のために保存しておくこともできるし、破棄することもできます。

私は最近、この手法を使って、学生向けに中心極限定理の小さな図を作成しました。私は母集団の確率密度の上に標本平均の密度を重ねて表示したいと思いました。

私はRを開いて変数を定義し、関数を実行して母集団の密度のプロットを作成しました。この数行のコマンドはもう一度実行することになるだろうと思ったので、私はRのスクリプトエディタウィンドウを開き、図B-1に示すように、コマンドをペーストしました。



図B-1 新たなエディタウィンドウにコマンドを取り込む

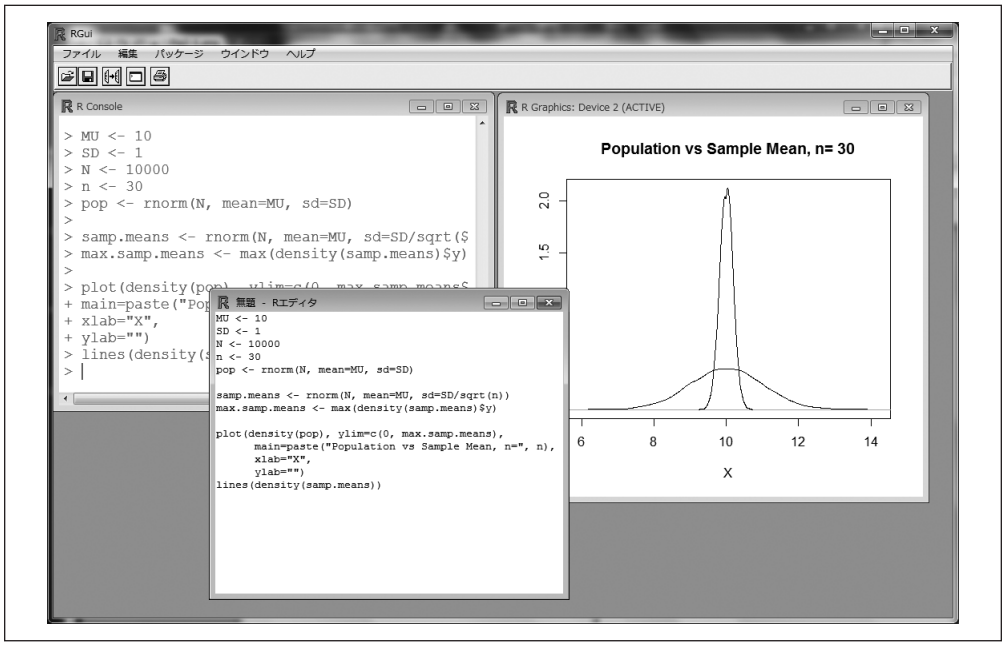


図 B-2　展開した状態のコード

私は先ほど書いたグラフの上に、標本平均の密度を重ねたいと思い、再びコンソールウィンドウにコマンドの入力を始めました。しかし、2 番目の標本平均の密度プロットは、グラフの内部に収まらなかったので、私は関数呼び出しを修正して、すべてを再実行しなければなりませんでした。

しかし、ここで本領発揮です。私はほとんど何も再入力せず、ただコード数行をエディタウィンドウにペーストし、plot 関数の呼び出しを改良し、エディタウィンドウの内容全体を再実行しました。

ほら、これだけで、プロットの大きさの問題は修正されたのです。私はコードを再び改良し（タイトルとラベルを追加）、もう一度ウィンドウ全体の内容を再実行しました。今度は、最終的に図 B-2 で示すような図が生成されました。そのあとエディタの内容をスクリプトファイルに保存し、このグラフがいつでも再作成できるようにしました。

レシピ B.14　よくある間違いをなくす

問題

初心者が犯すようなよくある間違いをなくしたい。さらには経験者が犯すような間違いもなくしたい。

解説

起こりやすいトラブルを挙げます。

関数呼び出しのあとに括弧を忘れる

R 関数の名前のあとに括弧を加えて呼び出します。例えば、次の行は ls 関数を呼び出します。

```
> ls()
[1] "x" "y" "z"
```

しかし、括弧を忘れると、R はこの関数を実行しません。その代わりに関数定義を表示しますが、あまり目にしたいものではないはずです。

```
> ls
function (name, pos = -1, envir = as.environment(pos), all.names = FALSE,
    pattern)
{
    if (!missing(name)) {
        nameValue <- try(name)
        if (identical(class(nameValue), "try-error")) {
            name <- substitute(name)
.
. (etc.)
.
```

Windows のファイルパス中のバックスラッシュを二重にし忘れる

次の関数呼び出しは、F:\research\bio\assay.csv という Windows のファイルを呼び出すように思えますが、実際は呼び出してくれません。

```
> tbl <- read.csv("F:\research\bio\assay.csv")
```

文字列中のバックスラッシュ（\）は特別な意味を持つため、二重にする必要があります。R はこのファイル名を F:researchbioassay.csv と解釈します。これはユーザが期待するものではありません。**レシピ 1.5** に解決につながる答えがあります。

「<-」を「<（空白）-」と間違えて入力する

割り当て演算子は <- であり、< と - の間に空白は入りません。

```
> x <- pi        # x を 3.1415926... に設定する
```

間違えて < と - の間に空白を入れてしまうと、意味がまったく異なってしまいます。

```
> x < - pi       # おっと！ x を設定する代わりに、比較をしています
```

これは、x とマイナス π（-pi）との比較です。x の値は変更しません。運がよければ、x は定義されておらず、R は何かがおかしいと警告を出すでしょう。

```
> x < - pi
 エラー： オブジェクト 'x' がありません
```

x の値が定義されていれば、R は比較を行い、TRUE または FALSE の論理値を表示します。何かおかしいと警告を出してくれればよいのですが、通常、割り当ては何の警告も表示しません。

```
> x <- 0       # x を 0 に初期化
> x < - pi     # おっと！
[1] FALSE
```

複数行の式の正しくない継続

R はユーザが完全な式の入力を完了するまで読み込みます。行数はどんなに多くても構いません。R は + プロンプトを使って、完全な式となるまで後続の入力を促します。この例では、1 つの式を 2 行に分割しています。

```
> total <- 1 + 2 + 3 +      # 次の行に続く
+    4 + 5
> print(total)
[1] 15
```

うっかり式が不完全なうちに終わらせてしまうと、問題が発生しますが、これはよく起こります。

```
> total <- 1 + 2 + 3          # おっと！ R は完全な式とみなします
> + 4 + 5                     # これは新しい式です。R はこの値を表示します
[1] 9
> print(total)
[1] 6
```

この入力がおかしいことを示す証拠が 2 つあります。R は継続プロンプト（+）ではなくて、通常のプロンプト（>）を出して入力を促しています。さらに、4 + 5 の値を表示しています。

ときどきしか R を使わないユーザにとって、このよくある間違いは頭痛の種です。プログラマにとっては悪夢です。R スクリプトに見つけづらいバグを入れ込むもととなるからです。

= を == の代わりに使う

比較には二重等号演算子（==）を使います。間違えて等号 1 つ（=）だけを使うと、変数を上書きしてしまい元に戻せなくなってしまいます。

```
> v == 0      # v と 0 を比較します
> v = 0       # v に 0 を割り当て、前の内容を上書きします
```

1:(n+1) のつもりで 1:n+1 と書く

式 1:n+1 は数列 1, 2, ..., n, $n+1$ を意味すると考えるかもしれませんが、違います。これは数列 1, 2, ..., n の各要素に 1 を足し、2, 3, ..., n, $n+1$ という結果となります。これは、R が 1:n+1 を

(1:n)+1 として解釈するからです。求めたい値を正しく取得したければ、括弧を使います。

```
> n <- 5
> 1:n+1
[1] 2 3 4 5 6
> 1:(n+1)
[1] 1 2 3 4 5 6
```

リサイクル規則で痛い目に遭う

ベクトル演算とベクトル比較は、両方のベクトルが同じ長さのときはうまくいきます。しかし、被演算数の長さが異なる場合、不可解な結果となります。この危険を回避するために、リサイクル規則を理解して覚えます（**レシピ 2.3**）。

パッケージをインストールしたのに、library() や require() を使わない

パッケージを使うには、まずインストールする必要があります。しかし、もう一段階作業が必要です。library か require を使ってサーチパスにパッケージを読み込む必要があります。そうしないと、R はパッケージの関数やデータセットを認識しません。**レシピ C.6** を参照してください。

```
> truehist(x,n)
 エラー： 関数 "truehist" を見つけることができませんでした
> library(MASS)      # MASS パッケージを R に読み込む
> truehist(x,n)
>
```

aList[[i]] と書きたかったのに、aList[i] と書いてしまう、またはその逆を行う

変数 lst がリストの場合、2 通りの方法でインデックス付けができます。lst[[n]] はリストの n 番目の要素です。一方 lst[n] は、lst の n 番目の要素を 1 つだけ持つリストであり、両者は大きく異なります。**レシピ 2.7** を参照してください。

&& ではなく & を使う、またはその逆を行う。| と || についても同様に取り違えてしまう

論理式では & と | を使って、論理値 TRUE と FALSE を呼び出します（**レシピ B.9** を参照）。

if 文と while 文内部のフロー制御式には && と || を使います。

他のプログラミング言語に慣れているプログラマは、反射的に && と || を「速いから」という理由でどこでも使ってしまいます。しかし、これらの演算子は、論理値のベクトルに適用するとおかしな結果となります。そのため、本当に必要なとき以外は、使用を避けるようにします。

複数の引数を 1 つの引数だけを取る関数に渡す

mean(9,10,11) の値は何だと思いますか？ 10 ではありません。9 です。mean 関数は、第 1 引数の平均値を求めます。2 番目と 3 番目の引数はその他の位置引数と解釈されます。

mean のように、1 つの引数しか取らない関数もあります。max や min のような関数は、複数の引数を取り、すべての引数に作用します。引数の数が 1 つなのか複数なのかを必ず把握するようにします。

max が pmax のように振る舞うと考える、または min が pmin のように振る舞うと考える

max 関数と min 関数は複数の引数を取り、1 つの値を返します。max はすべての引数の最大値、min は最小値を返します。

pmax 関数も pmin 関数も、複数の引数を取りますが、引数を要素ごとに比較して得られた結果の値からなるベクトルを返します。**レシピ 9.9** を参照してください。

データフレームを理解しない関数を間違えて使う

データフレームに関してとても賢い関数があります。そういった関数はデータフレームの各列に適用させます。mean 関数と sd 関数は代表的な例です。mean は各列の平均を、sd は各列の標準偏差を計算します。mean も sd も、各列が独立した変数であり、データを混合させることは意味がないと考えるからです。

悲しいかな、すべての関数がこのように賢いわけではありません。median, max, mix 関数はあまり賢いとは言えません。これらの関数は、各列の各値をひとくくりにし、その塊から結果を求めます。これはユーザが期待する結果ではないかもしれません。どの関数がデータフレームに対応していて、対応していないのかに注意してください。

回答を探さずに、メーリングリストへ質問を投稿する

時間をムダにしてはいけません。そして自分以外の人の時間をムダにしてもいけません。メーリングリスト、または Stack Overflow へ質問を投稿する前に、必ず行うべきことがあります。まずアーカイブの検索をしてください。たいてい誰かがすでにあなたの質問に答えているはずです。その場合、質問の論議スレッドで答えを見つけられるでしょう。**レシピ A.12** を参照してください。

関連項目

レシピ A.12、**レシピ B.9**、**レシピ 2.3**、**レシピ 2.7** を参照。

付録 C
R を操作する

はじめに

R はとにもかくにも巨大なソフトの塊です。必然的に、R の機能を使って長い時間を過ごすことになります。つまり、設定し、カスタマイズし、更新し、自分の計算環境に適合させていくのです。この章は、ユーザのこうした作業を支援するものです。数値、統計、グラフについての話題は何もありません。この章では R をソフトウェアとして扱うことについて書かれています。

レシピ C.1　作業ディレクトリを取得し、設定する

問題

作業ディレクトリを変更したい。あるいは、単に作業ディレクトリがどこにあるかを知りたい。

解決策

コマンドライン

getwd を使って作業ディレクトリの場所を確認します。変更には setwd を使います。

```
> getwd()
[1] "/home/paul/research"
> setwd("Bayes")
> getwd()
[1] "/home/paul/research/Bayes"
```

Windows

メインメニューから「ファイル」→「ディレクトリの変更 ...」を選択します。

OS X

メインメニューから「その他」→「作業ディレクトリの変更 ...」を選択します。

Windows も OS X も、メニューを選択すると、ファイルブラウザの中で現在の作業ディレクトリが開きます。そこから新しい作業ディレクトリを選ぶことができます。

解説

作業ディレクトリは、すべての入出力ファイルのデフォルトとなる位置で、データファイルの読み込み／書き込み、スクリプトファイルのオープン／保存、ワークスペースイメージの保存などを行うため、重要です。絶対パスを指定しないでファイルを開く場合、R はそのファイルは作業ディレクトリにあると考えます。

R 起動直後の作業ディレクトリの場所は、R の起動方法によって変わります。**レシピ A.2** を参照してください。

関連項目

Windows のファイル名を操作するには**レシピ 1.5**、ワークスペースを保存するには**レシピ C.2** を参照してください。

レシピ C.2　ワークスペースを保存する

問題

R を終了せずにワークスペースを保存したい。

解決策

`save.image` 関数を呼び出します。

```
> save.image()
```

解説

ワークスペースは R の変数と関数が置かれる場所で、R の起動時に生成されます。ワークスペースはユーザの PC のメインメモリ内に置かれ、R を終了するまで存続します。また、ワークスペースは終了時に保存できます。

しかし、R の終了時でないときでも、ワークスペースを保存したいときがあるかもしれません。例えば、ランチに行きたいときや予期せぬ停電やマシンのクラッシュに備え、作業を保護したいと考えるかもしれません。その場合には、`save.image` 関数を使います。

するとワークスペースは作業ディレクトリの .RData という名前のファイルに書き込まれます。R を起動するとき、R はまず .RData ファイルを探し、ファイルがあればワークスペースを .RData ファイルで初期化します。

しかし悲しいかな、ワークスペースは開かれた（描画された）グラフを保存しません。この画面上のグラフは、R 終了時に失われてしまいます。そして、開かれたグラフを保存して復元する簡単な方法はありません。そのため、終了する前に、グラフを再描画できるようなデータと R コードは保存しておきます。

関連項目

R 終了時にワークスペースを保存する方法については**レシピ A.2** を、作業ディレクトリの設定については

レシピ C.1 を参照してください。

レシピ C.3　コマンド履歴を見る

問題

最近入力した一連のコマンドを確認したい。

解決策

上向きの矢印（ ↑ ）または Ctrl-P を押して遡ります。または history 関数を使って直前の入力を確認します。

```
> history()
```

解説

history 関数は、直前のコマンドを表示します。デフォルトでは、直前の 25 行を表示します。

```
> history(100)          # 直前のコマンド履歴 100 を表示
> history(Inf)          # 保存されている履歴をすべて表示
```

最近入力したコマンドを表示するには、コマンドラインの編集機能を利用して、遡ります。つまり、上向きの矢印（ ↑ ）または Ctrl-P を押すと、前に入力したコマンドが 1 行ずつ再表示されます。

R を終了していても、コマンド履歴を見ることはできます。コマンド履歴は作業ディレクトリ中の .Rhistory ファイルに保存されるので、テキストエディタで .Rhistory ファイルを開き、ファイルの一番下までスクロールします。一番下に直前の入力があります。

レシピ C.4　先に実行したコマンドの結果を保存する

問題

値を求める式を入力したが、結果を変数に保存し忘れた。

解決策

.Last.value という名前の特別な変数が最新の評価された式の値を保存します。次の式を入力する前に、.Last.value の内容を別の変数に保存します。

解説

長い式の入力や実行に長時間かかかる関数の呼び出しはイライラするのに、結果の保存を忘れてしまいました。幸い、式を再入力する必要も、関数を呼び出す必要もありません。結果は .Last.value 変数に保存されています。

```
> aVeryLongRunningFunction()      # おっと！ 結果の保存を忘れてしまった！
[1] 147.6549
> x <- .Last.value                # ここで結果を取得する
> x
[1] 147.6549
```

注意：.Last.value の内容は、別の式を入力するたびに上書きされます。そのため値はただちに取っておくようにします。次の式を評価してからでは遅すぎます。

関連項目

コマンド履歴の呼び出しは**レシピ C.3** を参照してください。

レシピ C.5　サーチパスを表示する

問題

現在 R に読み込まれているパッケージの一覧を見たい。

解決策

search 関数を引数なしで実行します。

```
> search()
```

解説

現在メモリに読み込まれている使用可能なパッケージのリストは**サーチパス**と呼ばれます。多くのパッケージがユーザの PC にインストールされているかもしれませんが、実際に常に R インタプリタに読み込まれているものはわずかでしょう。現在どのパッケージが読み込まれているのか知りたいこともあるでしょう。

引数なしで search 関数を実行すると、読み込まれているパッケージの一覧を返します。次のような出力になります。

```
> search()
[1] ".GlobalEnv" "package:stats" "package:graphics"
[4] "package:grDevices" "package:utils" "package:datasets"
[7] "package:methods" "Autoloads" "package:base"
```

あなたの PC では、インストールされているパッケージにより、これとは違う結果になるかもしれません。search 関数の戻り値は文字列のベクトルです。最初の文字列は ".GlobalEnv" で、これはワークスペースを指します。ほとんどのパッケージは "package:*packagename*" という形をしています。これは、*packagename* という名前のパッケージが R に読み込まれていることを意味しています。この例で読み込まれているパッケージは、stats, graphics, grDevices, utils などです。

R はサーチパスを使って関数を見つけます。関数名を入力すると、上の例で表示された順番に R はパス

を検索します。そして読み込まれたパッケージ内の関数を見つけるまで検索を続けます。関数が見つかればその関数を実行します。見つからなければエラーメッセージを表示して止まります。実際はそれだけではありません。サーチパスはパッケージだけでなく環境も含んでいるので、パッケージ内でオブジェクトによって初期化された場合は検索アルゴリズムは変わることがあります。詳しくは、「R Language Definition」を参照してください。

ワークスペース（.GlobalEnv）は一覧の先頭にあったので、R はまず他のパッケージの検索の前にワークスペース内で関数を探します。もしあなたのワークスペースとパッケージが両方とも同じ名前の関数を含んでいたら、ワークスペースはその関数を「覆い隠し」ます。つまり、関数が見つかると、その後は検索を中止してしまい、パッケージ内の関数は探しません。これは、パッケージの関数を無視したいなら幸いと言えますが、パッケージの関数にアクセスしたいなら悪夢となります。

R データセット（ファイルではない）の検索にも同じような手順でサーチパスを使います。

Unix ユーザは R のサーチパスと Unix のサーチパス（PATH 環境変数）を混同してはいけません。両者は概念的に似ていますが異なるものです。R のサーチパスは R の内部に対するもので、関数やデータセットの場所を表すのに使われます。一方 Unix のサーチパスは実行可能プログラムの位置を知るために使われます。

関連項目

パッケージを R に読み込むには**レシピ C.6** を、（読み込まれているパッケージだけでなく）インストールされているパッケージの一覧の表示には**レシピ C.8** を、データフレームをサーチパスに挿入するには**レシピ 2.31** を参照してください。

レシピ C.6　パッケージの関数にアクセスする

問題

標準パッケージまたは自分でダウンロードしたパッケージが PC にインストールされている。パッケージの関数を使おうとしたが、R がその関数を見つけられない。

解決策

library 関数または require 関数を使って、パッケージ（*packagename*）を R に読み込みます。

```
> library(packagename)
```

解説

R はいくつかの標準パッケージと一緒にインストールされます。しかし、起動時にすべてが自動的に読み込まれるわけではありません。同様に、多くの便利なパッケージを CRAN からダウンロードしてインストールできますが、R を実行する際に自動的に読み込まれるわけではありません。MASS パッケージは R の標準パッケージですが、このパッケージ内の lda 関数を使う際、次のようなメッセージが表示されるかもしれません。

```
> lda(x)
 エラー：  関数 "lda" を見つけることができませんでした
```

R は、現在メモリに読み込まれているパッケージの中から関数の中で lda が見つけられないと文句を言っています。

library 関数や require 関数を使うと、R はメモリにパッケージを読み込み、そのパッケージ内の関数は実行可能になります。

```
> lda(f ~ x + y)
 エラー：  関数 "lda" を見つけることができませんでした
> library(MASS)                    # MASS ライブラリをメモリに読み込む
> lda(f ~ x + y)                   # これで R は関数を見つけられる
Call:
lda(f ~ x + y)

Prior probabilities of groups:
.
. (etc.)
.
```

library 関数を呼び出すまでは R は lda という関数名を認識しません。library によってパッケージ内の関数が利用可能になり、lda 関数の呼び出しも機能します。

パッケージ名を引用符でくくる必要はありません。

require 関数はほぼ library 関数と同一です。require 関数にはスクリプトを書く上で便利な 2 つの機能があります。require 関数はパッケージの読み込みがうまくいっていれば TRUE を返し、うまくいかなければ FALSE を返します。library 関数はエラーを出すだけですが、require 関数は警告を発します。

両関数ともキーとなる重要な機能があります。パッケージがすでに読み込まれている場合は、再読み込みは行いません。そのため、同じパッケージを 2 回呼んでも安全です。この機能はスクリプトを書く際に特に便利です。スクリプトでは、読み込まれたパッケージが再読み込みされないとわかっていても、必要なパッケージを読み込むことを記述することができます。

detach 関数は、現在読み込まれているパッケージをメモリから削除します。

```
> detach(package:MASS)
```

パッケージ名は package:MASS のように指定します。

パッケージをメモリから削除する理由の 1 つは、同じ名前の関数がサーチリストの下位にあるなどして関数名が衝突してしまうからです。このような衝突が起こると、上位の関数は下位の関数を覆い隠してしまうため、ユーザはもはや下位関数を目にすることはありません。上位関数を見つけた時点で R は検索を中止してしまうからです。そのため、上位パッケージをメモリから削除して、下位関数を隠していた覆いを取り除きます。

関連項目

レシピ C.5 を参照。

レシピ C.7　組み込みデータセットにアクセスする

問題

R の組み込みデータセットを使いたい。

解決策

R の標準ディストリビューションにあるデータセットならばすでに利用可能となっています。dataset パッケージはサーチパス上にあるからです。

他のパッケージ内のデータセットにアクセスするには、data 関数を使います。引数にはデータセット名とパッケージ名を指定します。

```
> data(dsname, package="pkgname")
```

解説

R をインストールする際、多くの組み込みデータセットも一緒にインストールされます。このようなデータセットは実際に試すことのできるデータなので、R を学ぶ際に便利です。

多くのデータセットは、datasets（そのまますね）という名前のパッケージ内にあり、このパッケージは標準ディストリビューションに入っています。サーチパス内にあるので、内容には簡単にアクセスできます。例えば、pressure という組み込みデータセットは次のように使います。

```
> head(pressure)
  temperature pressure
1           0   0.0002
2          20   0.0012
3          40   0.0060
4          60   0.0300
5          80   0.0900
6         100   0.2700
```

pressure について詳しく知りたければ help 関数を使います。pressure だけでなく他のデータセットの情報もわかります。

```
> help(pressure)          # pressure データセットについてのヘルプページを表示
```

data 関数を引数なしで呼び出すと、datasets の目次が表示されます。

```
> data()                  # datasets の一覧を表示
```

他の R パッケージを追加して、datasets にないデータセットを利用することができます。 例えば、MASS パッケージには、多くの面白いデータセットがあります。パッケージのデータセットにアクセスするには、data 関数を使って package 引数で指定します。MASS パッケージには Cars93 というデータセットがあります。次のようにアクセスできます。

```
> data(Cars93, package="MASS")
```

このように data 関数を呼び出すと Cars93 データセットは利用できるようになります。続いて、summary(Cars93)、head(Cars93)、のように実行できます。

サーチリストにパッケージを追加すると、（例えば library(MASS) のように）data を呼び出す必要がありません。追加すればそのデータセットは自動的に利用できるようになります。

data 関数を使って MASS 内あるいは他のパッケージ内で利用可能なデータセットの一覧を見ることもできます。その際、引数にはデータセット名ではなくパッケージ名（*pkgname*）を指定します。

```
> data(package="pkgname")
```

関連項目

サーチパスについて詳しくは**レシピ C.5** を、パッケージや library 関数について詳しくは**レシピ C.6** を参照してください。

レシピ C.8　インストールされているパッケージの一覧を見る

問題

PC にどんなパッケージがインストールされているのか知りたい。

解決策

引数なしで library 関数を実行すると、標準パッケージの一覧が表示されます。installed.packages を実行すると、パッケージについてさらに詳しい情報を確認できます。

解説

引数なしの library 関数は、インストールされているパッケージの一覧を表示します。この一覧はかなり長いものです。Linux では、最初の数行はこのような出力になるでしょう。

```
> library()
Packages in library '/usr/local/lib/R/site-library':
boot                    Bootstrap R (S-Plus) Functions (Canty)
CGIwithR                CGI Programming in R
class                   Functions for Classification
cluster                 Cluster Analysis Extended Rousseeuw et al.
DBI                     R Database Interface
```

```
expsmooth              Data sets for "Forecasting with exponential
                       smoothing"
.
. (etc.)
.
```

Windows と OS X では、一覧はポップアップウィンドウ内に表示されます。

installed.packages 関数ではさらに詳しい情報が得られます。この関数は、マシンのパッケージに関する情報を行列にして返します。行列の各行はインストールされている各パッケージを意味しています。また各列はパッケージ名、ライブラリパス、バージョンなどの情報です。情報はインストールされたパッケージの内部データベースから取得します。

この行列から特定の情報を抽出するには、通常インデックスメソッドを使います。この Windows のコード片は installed.packages と呼ばれるもので、Package と Version の両方の列を抽出し、それぞれのパッケージについてどのバージョンがインストールされているか確認します。

```
> installed.packages()[,c("Package","Version")]
             Package         Version
acepack      "acepack"       "1.3-2.2"
alr3         "alr3"          "1.0.9"
base         "base"          "2.4.1"
boot         "boot"          "1.2-27"
bootstrap    "bootstrap"     "1.0-20"
calibrate    "calibrate"     "0.0"
car          "car"           "1.2-1"
chron        "chron"         "2.3-12"
class        "class"         "7.2-30"
cluster      "cluster"       "1.11.4"
.
. (etc.)
.
```

関連項目

パッケージをメモリに読み込む方法は**レシピ C.6** を参照してください。

レシピ C.9　CRAN からパッケージをインストールする

問題

CRAN で見つけたパッケージを PC にインストールしたい。

解決策

コマンドライン

install.packages 関数の引数にパッケージ名を引用符で括って指定します。

```
> install.packages("packagename")
```

Windows
: メニューから「パッケージ」→「パッケージのインストール ...」を選んでダウンロード／インストールを行います。

OS X
: メニューから「パッケージとデータ」→「パッケージインストーラ」を選んでダウンロード／インストールを行います。

プラットフォームが何であれ、CRAN のミラーサイトを選択するように求められます。
また、Windows と OS X では、ダウンロードするパッケージを選択するように求められます。

Linux/Unix のシステム全体にインストール

Linux または Unix システムでは、システムライブラリへのパッケージのインストールには root 権限が必要です。Linux や Unix のディレクトリは通常は手作業では書き込み不可だからです。

そのため、通常はシステム管理者がインストールを行います。管理者がシステム全体へのインストールに積極的でない、あるいはできない場合、個人用ライブラリにパッケージをインストールします。

root 権限を持っている場合は次のように行います。

1. su あるいは sudo を実行し、root シェルを起動する。
2. root シェルで R セッションを開始する。
3. そこで install.packages 関数を実行する。

root 権限あるかどうかはすぐにわかります。root 権限がなければ install.packages 関数は停止し、必要なディレクトリに書き込めないと警告を出します。 それからユーザに個人用ライブラリを代わりに作成するかと聞かれます。作成する場合は、ホームディレクトリに必要がディレクトリが作成され、そこにパッケージがインストールされます。

解説

パッケージを使う前にはまず、ローカルへのインストールが必要です。インストーラはパッケージをダウンロードできる CRAN のミラーサイトを選ぶよう促します。

```
--- Please select a CRAN mirror for use in this session ---
```

そして、CRAN ミラーサイトの一覧を表示します。その中から近いサイトを選択します。
公式 CRAN サーバには、比較的高性能のマシンが使われています。オーストリアのウィーンにあるウィー

ン経済大学統計数学部がホストしてくれています。すべての R ユーザが公式サーバからダウンロードしたら、負荷が高すぎておかしくなってしまう恐れがあるため、世界中にある数多くのミラーサイトから近くのミラーサイトを見つけ、そこを利用するようにします。

もし、新しいパッケージがまだローカルにはインストールされていない他のパッケージに依存しているなら、R インストーラは自動的に必要なパッケージをダウンロード／インストールしてくれます。これは優れた機能です。依存関係を見つけ、解決するという退屈な作業からユーザを解放してくれます。

Linux や Unix にインストールするときは特別な配慮が必要です。パッケージはシステム用ライブラリか、個人用ライブラリかどちらかにインストールできます。システム用ライブラリのパッケージは誰もが使えますが、個人用ライブラリのパッケージは通常はあなただけしか使えません。そのため、人気がありよくテストされたパッケージはシステム用ライブラリに入れるとよいでしょう。一方、あまり有名でなくテストされていないパッケージは個人用ライブラリに入れることをおすすめします。

デフォルトでは、install.packages はシステム用インストールに使われることを前提としています。個人用ライブラリにインストールするには、まず最初にライブラリのディレクトリを作成します。例えば、~/lib/R のように作るときは、次のようにします。

```
$ mkdir ~/lib
$ mkdir ~/lib/R
```

R_LIBS 環境変数を設定してから R を起動します。そうしないとライブラリの場所がわかりません。

```
$ export R_LIBS=~/lib/R        # bash の場合
$ setenv R_LIBS ~/lib/R        # csh の場合
```

それから、lib 引数に、ライブラリのディレクトリを指定して install.packages を呼び出します。

```
> install.packages("packagename", lib="~/lib/R")
```

関連項目

関連パッケージを探すには**レシピ A.11** を、パッケージをインストールして使うには**レシピ C.6** を参照してください。

レシピ C.10　デフォルトの CRAN ミラーサイトを設定する

問題

パッケージをダウンロードしている。R に毎回催促されないように、デフォルトの CRAN ミラーサイトを設定したい。

解決策

ここでは**レシピ C.16** で説明したように、.Rprofile ファイルがあると仮定しています。

1. chooseCRANmirror 関数を呼び出す。

   ```
   > chooseCRANmirror()
   ```

   すると R は CRAN ミラーサイトの一覧を表示します。

2. リストから CRAN ミラーサイトを選択して、OK をクリックする。

3. ミラーサイトの URL を取得し、repos オプションの 1 番目の要素を確認する。

   ```
   > options("repos")[[1]][1]
   ```

4. この行を手元の .Rprofile ファイルに追加する。

   ```
   options(repos="URL")
   ```

   ここで、*URL* はミラーサイトの URL です。

解説

パッケージをインストールする際、あなたはおそらく毎回同じ CRAN ミラーを使うでしょう（つまり、ミラーは一番近いところです）。そして、R が繰り返しミラーサイトを選ぶよう催促することにうんざりしていることでしょう。もし、あなたがこの解決策に従えば、デフォルトミラーを持つことになるので、R はミラーサイトの催促をやめます。

repos オプションは、デフォルトのミラーサイト名です。chooseCRANmirror 関数には、repos オプションをユーザの選択に従って設定するという深刻な副作用があります。問題は、終了時に R が設定を忘れ、永続的なデフォルトを持たないままにしてしまうことです。.Rprofile ファイルの中の repos を設定すれば、R 起動時に毎回設定を復元できます。

関連項目

.Rprofile ファイルと options 関数について詳しくは**レシピ C.16** を参照してください。

レシピ C.11　起動メッセージを隠す

問題

長々とした R の起動メッセージに辟易している。

解決策

起動時に --quiet コマンドラインオプションを使います。

解説

R の起動メッセージには、R プロジェクトやヘルプの見方についての役に立つ情報も表示されるので、初心者にとっては便利です。しかし、目新しさもすぐに色あせてしまいます。

そんなときは --quiet オプションを使って R をシェルプロンプトから起動し、起動メッセージを隠します。

```
$ R --quiet
>
```

私の Linux マシンでは、次のように alias コマンドを使っているので、起動メッセージを目にすることはありません。

```
$ alias R="/usr/bin/R --quiet"
```

Windows でショートカットから R を起動しているなら、ショートカットに --quiet オプションを埋め込むことができます。ショートカットアイコンを右クリックし、「プロパティ」を選択し、「ショートカット」タブを選択します。「リンク（T）:」の最後に --quiet オプションを追加します。プログラムパスと --quiet の間は必ず 1 つスペースを空けます。

関連項目

レシピ A.2 を参照

レシピ C.12　スクリプトを実行する

問題

テキストファイルに保存しておいた一連の R コマンドを、ここで実行したい。

解決策

source 関数は、R にテキストファイルを読み込み内容を実行するように命令します。

```
> source("myScript.R")
```

解説

長くて入力が面倒、あるいは頻繁に使う R のコード片があれば、テキストに保存しておきます。こうしておけば、コードを再入力することなく、簡単に再実行できるようになります。source 関数を使ってコードの読み込みと実行を行います。R コンソールへの入力のときと同様です。

hello.R ファイルの内容は以下の見慣れた挨拶文の 1 行だとします。

```
print("Hello, World!")
```

そして、source 関数でファイルを読み込み、内容を実行します。

```
> source("hello.R")
[1] "Hello, World!"
```

echo=TRUE と設定すると、実行前にスクリプト行をエコー、つまりRプロンプトが表示されてから、実行結果が表示されます。

```
> source("hello.R", echo=TRUE)

> print("Hello, World!")
[1] "Hello, World!"
```

関連項目

GUI内でRのコードブロックを実行するには**レシピB.13**を参照してください。

レシピC.13 バッチスクリプトを走らせる

問題

Unix や OS X のシェルスクリプトや Windows の BAT スクリプトのようなコマンドスクリプトを書いている。スクリプト中でRスクリプトを実行したい。

解決策

CMD BATCH サブコマンドを使って、スクリプト（*scriptfile*）と、出力ファイル（*outputfile*）を指定してRを実行します。

```
$ R CMD BATCH scriptfile outputfile
```

標準出力したい場合、あるいはコマンドライン引数をスクリプトに渡したい場合は Rscript コマンドを使います。

```
$ Rscript scriptfile arg1 arg2 arg3
```

解説

Rは通常は対話型のプログラムで、ユーザに入力を促し結果を表示します。しかし、コマンドをスクリプトから読み込むようなバッチモードで実行したいこともあるでしょう。バッチモードは統計解析を含むスクリプトのようなシェルスクリプト内では特に便利です。

CMD BATCH サブコマンドは、Rをバッチモード内に置きます。 バッチモードでは、*scriptfile* から読み込み、*outputfile* に書き出します。ユーザとはやり取りしません。

コマンドラインオプションを使い、バッチの振る舞いを環境に合わせて変えることもできます。例えば、--quiet オプションを使うと、起動メッセージを隠すので、出力がすっきりします。

```
$ R CMD BATCH --quiet myScript.R results.out
```

バッチモードには、他にも便利なオプションがあります。

オプション	機能
--slave	--quiet と同様ですが、入力のエコーまで隠すのでさらに出力がすっきりします。
--no-restore	起動時にワークスペースを復元しません。空のワークスペースでスクリプトを始めたいときに重要です。
--no-save	終了時、ワークスペースを保存しません。これを指定しないと、R はワークスペースを保存し、作業ディレクトリの .RData ファイルを上書きします。
--no-init-file	.Rprofile ファイルも ~/.Rprofile ファイルも読み込みません。

CMD BATCH サブコマンドは通常、スクリプトが完了すると proc.time を呼び出し、実行時間を表示します。これが鬱陶しいなら、q 関数の引数を runLast=FALSE にして呼び出してスクリプトを実行すると、proc.time を呼び出すことはしません。

CMD BATCH サブコマンドには制限が 2 つあります。まず、出力が常にファイルであること、そしてコマンドライン引数をスクリプトに簡単に渡すことができないことです。 どちらかの制限が問題ならば、R に付属する Rscript プログラムの利用を検討します。最初のコマンドライン引数（myScript.R）が、スクリプト名で、それに続く引数がスクリプトに渡されます。

```
$ Rscript myScript.R arg1 arg2 arg3
```

スクリプト内では、commandArgs を呼び出して、コマンドライン引数にアクセスします。commandArgs は文字列のベクトルとして引数を返します。

```
argv <- commandArgs(TRUE)
```

Rscript プログラムは、さきほど述べたような CMD BATCH と同じコマンドライン引数を取ります。

出力は、標準出力されます。それはもちろんシェルスクリプトの呼び出しを R が受け継いでいるからです。通常のリダイレクションを使って出力をファイルにリダイレクトすることもできます。

```
$ Rscript --slave myScript.R arg1 arg2 arg3 >results.out
```

ここにある短い R スクリプト arith.R は、2 つのコマンドライン引数を取り、4 種類の演算を行います。

```
argv <- commandArgs(TRUE)
x <- as.numeric(argv[1])
y <- as.numeric(argv[2])
cat("x =", x, "\n")
cat("y =", y, "\n")
cat("x + y = ", x + y, "\n")
cat("x - y = ", x - y, "\n")
cat("x * y = ", x * y, "\n")
cat("x / y = ", x / y, "\n")
```

このスクリプトは次のように呼び出します。

```
$ Rscript arith.R 2 3.1415
```

そして次のように出力されます。

```
x = 2
y = 3.1415
x + y = 5.1415
x - y = -1.1415
x * y = 6.283
x / y = 0.6366385
```

Linux や Unix では、Rscript プログラムのパスを先頭行に #! を伴って記述することによって、スクリプトを完結させることができます。Rscript スクリプトがシステムの /usr/bin/Rscrip にインストールされているとします。そしてこの行を arith.R に追加すると、完結したスクリプトになります。

```
#!/usr/bin/Rscript --slave
argv <- commandArgs(TRUE)
x <- as.numeric(argv[1])
.
. (etc.)
.
```

シェルプロンプトで、スクリプトを実行可能にします。

```
$ chmod +x arith.R
```

これで、Rscript 接頭辞がなくてもスクリプトを直接呼び出すことができます。

```
$ arith.R 2 3.1415
```

関連項目

R 内からスクリプトを実行するには**レシピ C.12** を参照してください。

レシピ C.14　環境変数の取得と設定

問題

環境変数の値を知りたい、またはその値を変更したい。

解決策

Sys.getenv 関数を使って値を確認します。Sys.putenv 関数でその値を変更します。

```
> Sys.getenv("SHELL")
SHELL
"/bin/bash"
> Sys.setenv(SHELL="/bin/ksh")
```

解説

Linux と Unix では、ソフトウェアの設定と制御のために環境変数を使うことがあります。各プロセスには自身の環境変数のセットがあります。環境変数は親プロセスから引き継がれたものです。ユーザは振る舞いを理解するために、R プロセス用に設定された環境変数を確認する必要があるかもしれません。また、振る舞いを変更するために、環境変数の設定を変更する必要があるかもしれません。

私は R を 1 か所から起動しますが、実際にはグラフィックスを異なった場所に表示させたいこともあります。例えば、Linux のターミナルで R を実行していても、グラフィックスは、他の人が見やすいように大きな画面に表示させたいこともあります。あるいは、R を自分の Linux ワークステーションで走らせて、グラフィックスを同僚のワークステーションに表示させたいこともあります。Linux では、R はグラフィックスの表示に X Window システムを使います。X Window は表示デバイスを DISPLAY という名前の環境変数にしたがって選択します。Sys.getenv を使って DISPLAY 環境変数の値を確認します。

```
> Sys.getenv("DISPLAY")
DISPLAY
":0.0"
```

すべての環境変数は文字列値です。ここで表示されている値 ":0.0" は、私のワークステーションの R セッションがディスプレイ 0、スクリーン番号 0 に接続されているという意味です。

グラフィックスを 10.0 のローカルディスプレイにリダイレクトするには、DISPLAY 環境変数を次のように変更します。

```
> Sys.putenv(DISPLAY="localhost:10.0")
```

同様に、グラフィックスを zeus という名前のワークステーションのディスプレイ 0、スクリーン 0 にリダイレクトすることもできます。

```
> Sys.putenv(DISPLAY="zeus:0.0")
```

どちらも、グラフィックスを描画前に DISPLAY 環境変数を設定しておく必要があります。

レシピ C.15　R のホームディレクトリの場所を探す

問題

R のホームディレクトリがどこであるか知る必要がある。ホームディレクトリとは、設定ファイルとインストール用ファイルが保存されている場所である。

解決策

Rは R_HOME という環境変数を作成します。R_HOME には Sys.getenv 関数を使ってアクセスできます。

```
> Sys.getenv("R_HOME")
```

解説

ほとんどのユーザは、Rのホームディレクトリが必要になることはありません。しかし、システム管理者やヘビーユーザは、Rのインストール用ファイルを確認したり、変更するために、知っておく必要があります。

Rを起動すると、R_HOME という名前の環境変数が定義されます（Rの変数ではありません）。R_HOME はRホームディレクトリへのパスで、Sys.getenv 関数でその値を呼び出すことができます。プラットフォーム別に R_HOME の例を示します。実際に表示される値はマシンによって異なるでしょう。

Windows

```
> Sys.getenv("R_HOME")
R_HOME
"C:\\PROGRA~1\\R\\R-21~1.1"
```

OS X

```
> Sys.getenv("R_HOME")
"/Library/Frameworks/R.framework/Resources"
```

Linux または Unix

```
> Sys.getenv("R_HOME")
R_HOME
"/usr/lib/R"
```

Windows の結果は、Rは古くさい DOS スタイルの圧縮パス名を返していて何だか変です。 省略されていない、ユーザにわかりやすいパスは C:\Program Files\R\R-2.10.1 となるでしょう。

Unix と OS X では、RHOME サブコマンドを使って R プログラムをシェルから実行し、ホームディレクトリを表示することができます。

```
$ R RHOME
/usr/lib/R
```

Unix と OS X の R のホームディレクトリには、インストール用ファイルが置かれています。しかし、Rの実行可能ファイルである必要はありません。実行可能ファイルは /usr/bin であるかもしれません。一方例えばRのホームディレクトリは、/usr/lib/R です。

レシピ C.16　R をカスタマイズする

問題

例えば、オプションや事前に読み込むパッケージの設定を変更するなどして、R セッションをカスタマイズしたい。

解決策

.Rprofile という名前のスクリプトを作成して R セッションをカスタマイズします。R は起動時に .Rprofile スクリプトを実行します。.Rprofile をどこに置くかはプラットフォームによって異なります。

OS X、Linux、Unix
: ホームディレクトリ (~/.Rprofile）保存する。

Windows
: マイドキュメントフォルダに保存する。

解説

R 起動時にプロファイルスクリプトが実行され、頻繁に使うパッケージの読み込みや R と設定オプションの調整といった繰り返し作業から解放されます。

.Rprofile というプロファイルスクリプトを作成し、それをホームディレクトリ（OS X, Linux, Unix）またはマイドキュメントディレクト（Windows XP）あるいはドキュメントディレクトリ（Windows Vista, Windows 7）に置きます。このプロファイルスクリプトは関数を呼び出してセッションをカスタマイズします。次の簡単なスクリプトは、MASS パッケージを呼び出して、プロンプトを R> に設定します。

```
require(MASS)
options(prompt="R> ")
```

プロファイルスクリプトは、最小の環境で実行するので、できることが制限されます。例えば、グラフィックウインドウを開こうとすると失敗します。グラフィックスパッケージがまだ読み込まれていないからです。また、ユーザは実行時間が長くかかる計算を試みるべきではありません。

ある特定のプロジェクトをカスタマイズするには、.Rprofile ファイルをプロジェクトファイルがあるディレクトリに置きます。そのディレクトリから R を起動すると、R はディレクトリ内の .Rprofile ファイル（ローカルプロファイル）を読み込みます。これにより、プロジェクトごとのカスタマイズを行うことができます（つまり読み込むパッケージは、プロジェクトだけが必要とします)。しかし、もし R がローカルプロファイルを見つけると、R はグローバルプロファイルを読み込みません。これにはイライラするかもしれません。しかし、簡単に直せます。単に、ローカルプロファイルから source 関数でグローバルファイルを実行すればよいのです。例えば、Unix では、このローカルプロファイルは、グローバルプロファイルをまず実行してからローカルプロファイルを実行します。

```
source("~/.Rprofile")
#
# ... remainder of local .Rprofile...
#
```

設定オプション

options関数の呼び出しを通じて行われるカスタマイズもあります。options関数は、Rの設定オプションを指定します。設定オプションはたくさんあります。options関数のヘルプページではオプションのすべての一覧が載っています。

```
> help(options)
```

例をいくつか示します。

オプション	説明
browser="*path*"	デフォルトのHTMLブラウザのパス。
digits=*n*	数値を表示する際の表示桁数。
editor="*path*"	デフォルトのテキストエディタのパス。
prompt="*string*"	入力プロンプト。
repos="*url*"	パッケージ用デフォルトリポジトリのURL。
warn=*n*	警告メッセージの表示制御

パッケージの読み込み

一般的なカスタマイズ方法はもう1つあります。それはパッケージの事前読み込みです。例えば、何回も使うようなパッケージはR実行時に毎回読み込みたいと思うかもしれません。次の呼び出しのように.Rprofileスクリプト内でrequireを呼び出すと簡単です（これはtseriesパッケージを読み込みます）。

```
require(tseries)
```

require関数が警告メッセージを出すと、R起動時の表示がゴチャゴチャしてしまいますが、suppressMessages関数で囲めば警告メッセージは隠れます。

```
suppressMessages(require(tseries))
```

require関数を明示的に呼び出す代わりに、defaultPackagesという名前の設定パラメータを設定することもできます。これはR起動時に読み込まれるパッケージのリストです。初期値はシステム定義のリストです。このリストにパッケージ名を追加すると、Rはそのパッケージ名も読み込むようになり、library関数やrequire関数をわざわざ呼び出す必要がなくなります。

これは私のRプロファイルの抜粋です。読み込まれるパッケージのリストを調節します。私は、ほとんどいつもzooパッケージを使うので、defaultPackagesのリストに追加します。

```
pkgs <- getOption("defaultPackages")      # システムがロードするパッケージのリストを取得
pkgs <- c(pkgs, "zoo")                    # "zoo" をリストに追加
options(defaultPackages = pkgs)           # 設定オプションを更新
rm(pkgs)                                  # 一時変数 pkgs を削除
```

defaultPackages は、サーチリスト内のパッケージの位置を制御できるので便利です。最後に追加されたパッケージは、最後に読み込まれます。そのため、この例では、zoo パッケージは、サーチリストの先頭に現れます（パッケージは読み込まれる際、サーチリストの先頭に挿入されることを思い出してください）。

毎回 zoo パッケージを読み込むと、R の起動が少し遅くなるというトレードオフがあります。それに実際、毎回 zoo を使うとも限りません。私の場合、ほとんど毎回 R を実行するたびに library(zoo) と入力するのにうんざりしていたので、私は少しばかり起動が遅くなっても便利なほうを選びました。

起動シーケンス

R 起動時に何が起こるかをこれからおおまかに説明します（詳細については help(Startup) と入力してヘルプを読んでください）。

1. R は Rprofile.site スクリプトを実行します。Rprofile.site はサイトレベルのスクリプトで、システム管理者がローカライズの一環としてデフォルトオプションを変更できるようにします。このスクリプトのフルパスは、R_HOME/etc/Rprofile.site です（R_HOME は R のホームディレクトリです。**レシピ C.15** を参照）。
   R のディストリビューションには Rprofile.site ファイルは入っていません。正確にはシステム管理者が必要なときに作成するものです。

2. R は作業ディレクトリ内の .Rprofile スクリプトを実行します。作業ディレクトリ内にこのファイルがなければ、ホームディレクトリ内の .Rprofile スクリプトを実行します。このタイミングで、ユーザ用の R のカスタマイズが行われます。ホームディレクトリの .Rprofile スクリプトは、グローバルなカスタマイズに使われます。より下位のディレクトリの .Rprofile スクリプトは、R がそのディレクトリから起動するとき指定したいカスタマイズを行います。例えば、プロジェクト別のディレクトリで R を起動する際にカスタマイズします。

3. R は、作業ディレクトリに .RData ファイルがあれば、そのファイルに保存されているワークスペースを読み込みます。そして、.RData という名前のファイルがあれば、そのファイルにワークスペースを保存します。R は .RData ファイルからワークスペースを再読み込みし、ローカル変数や関数へのアクセスを復元します。

4. R は定義されていれば .First 関数を実行します。.First 関数は起動時の初期化コードを定義するのにユーザやプロジェクトにとって便利です。.Rprofile ファイル内、またはワークスペース内で定義できます。

5. R は .First.sys 関数を実行します。ここでデフォルトパッケージを読み込みます。.First.sys 関数は R 内部にあり、通常はユーザでも管理者でも変更できません。

最後の手順 .First.sys 関数が実行されるまで、デフォルトパッケージは読み込まれていませんね。それまでは標準パッケージしか読み込まれていないからです。手順 5 より前では、標準パッケージ以外のパッケージは使えないということであり、これは重要です。.Rprofile スクリプトがグラフィカルウィンドウを開こうとすると失敗する原因でもあります。グラフィックスパッケージはまだこの時点では読み込まれていないからです。

関連項目

パッケージの読み込みについてさらに詳しくは**レシピ C.6** を参照してください。起動の R ヘルプページ（help(Startup)）と、オプションの R ヘルプページ（help(options)）を参照してください。

索　引

記号・数字

A

C

D

E

F

H

I

L

M

N

O

P

Q

R

S

U、V

W

X、Z

あ行

か行

さ行

た行

な行

ま行

や行

ら行

わ行